

# Stateful Protocol Composition and Typing

Andreas V. Hess\*    Sebastian Mödersheim\*    Achim D. Brucker<sup>†</sup>

May 9, 2020

\*DTU Compute, Technical University of Denmark, Lyngby, Denmark  
`{avhe, samo}@dtu.dk`

<sup>†</sup> Department of Computer Science, University of Exeter, Exeter, UK  
`a.brucker@exeter.ac.uk`



## Abstract

We provide in this AFP entry several relative soundness results for security protocols. In particular, we prove typing and compositionality results for stateful protocols (i.e., protocols with mutable state that may span several sessions), and that focuses on reachability properties. Such results are useful to simplify protocol verification by reducing it to a simpler problem: Typing results give conditions under which it is safe to verify a protocol in a typed model where only “well-typed” attacks can occur whereas compositionality results allow us to verify a composed protocol by only verifying the component protocols in isolation. The conditions on the protocols under which the results hold are furthermore syntactic in nature allowing for full automation. The foundation presented here is used in another entry to provide fully automated and formalized security proofs of stateful protocols.

**Keywords:** Security protocols, stateful protocols, relative soundness results, proof assistants, Isabelle/HOL, compositionality



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Preliminaries and Intruder Model</b>	<b>9</b>
2.1	Miscellaneous Lemmata (Miscellaneous)	9
2.2	Protocol Messages as (First-Order) Terms (Messages)	16
2.3	Definitions and Properties Related to Substitutions and Unification (More_Unification)	24
2.4	Dolev-Yao Intruder Model (Intruder_Deduction)	74
<b>3</b>	<b>The Typing Result for Non-Stateful Protocols</b>	<b>93</b>
3.1	Strands and Symbolic Intruder Constraints (Strands_and_Constraints)	93
3.2	The Lazy Intruder (Lazy_Intruder)	137
3.3	The Typed Model (Typed_Model)	150
3.4	The Typing Result (Typing_Result)	187
<b>4</b>	<b>The Typing Result for Stateful Protocols</b>	<b>243</b>
4.1	Stateful Strands (Stateful_Strands)	243
4.2	Extending the Typing Result to Stateful Constraints (Stateful_Typing)	270
<b>5</b>	<b>The Parallel Composition Result for Non-Stateful Protocols</b>	<b>301</b>
5.1	Labeled Strands (Labeled_Strands)	301
5.2	Parallel Compositionality of Security Protocols (Parallel_Compositionality)	306
<b>6</b>	<b>The Stateful Protocol Composition Result</b>	<b>325</b>
6.1	Labeled Stateful Strands (Labeled_Stateful_Strands)	325
6.2	Stateful Protocol Compositionality (Stateful_Compositionality)	339
<b>7</b>	<b>Examples</b>	<b>389</b>
7.1	Proving Type-Flaw Resistance of the TLS Handshake Protocol (Example_TLS)	389
7.2	The Keyserver Example (Example_Keyserver)	393



# 1 Introduction

The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle. The formalization presented in this entry is described in more detail in several publications:

- The typing result (section 3.4 “Typing.Result”) for stateless protocols, the TLS formalization (section 7.1 “Example.TLS”), and the theories depending on those (see Figure 1.1) are described in [2] and [1, chapter 3].
- The typing result for stateful protocols (section 4.2 “Stateful.Typing”) and the keyserver example (section 7.2 “Example.Keyserver”) are described in [3] and [1, chapter 4].
- The results on parallel composition for stateless protocols (section 5.2 “Parallel.Compositionality”) and stateful protocols (section 6.2 “Stateful.Compositionality”) are described in [4] and [1, chapter 5].

Overall, the structure of this document follows the theory dependencies (see Figure 1.1): we start with introducing the technical preliminaries of our formalization (chapter 2). Next, we introduce the typing results in chapter 3 and chapter 4. We introduce our compositionality results in chapter 5 and chapter 6. Finally, we present two example protocols chapter 7.

**Acknowledgments** This work was supported by the Sapere-Aude project “Composec: Secure Composition of Distributed Systems”, grant 4184-00334B of the Danish Council for Independent Research.

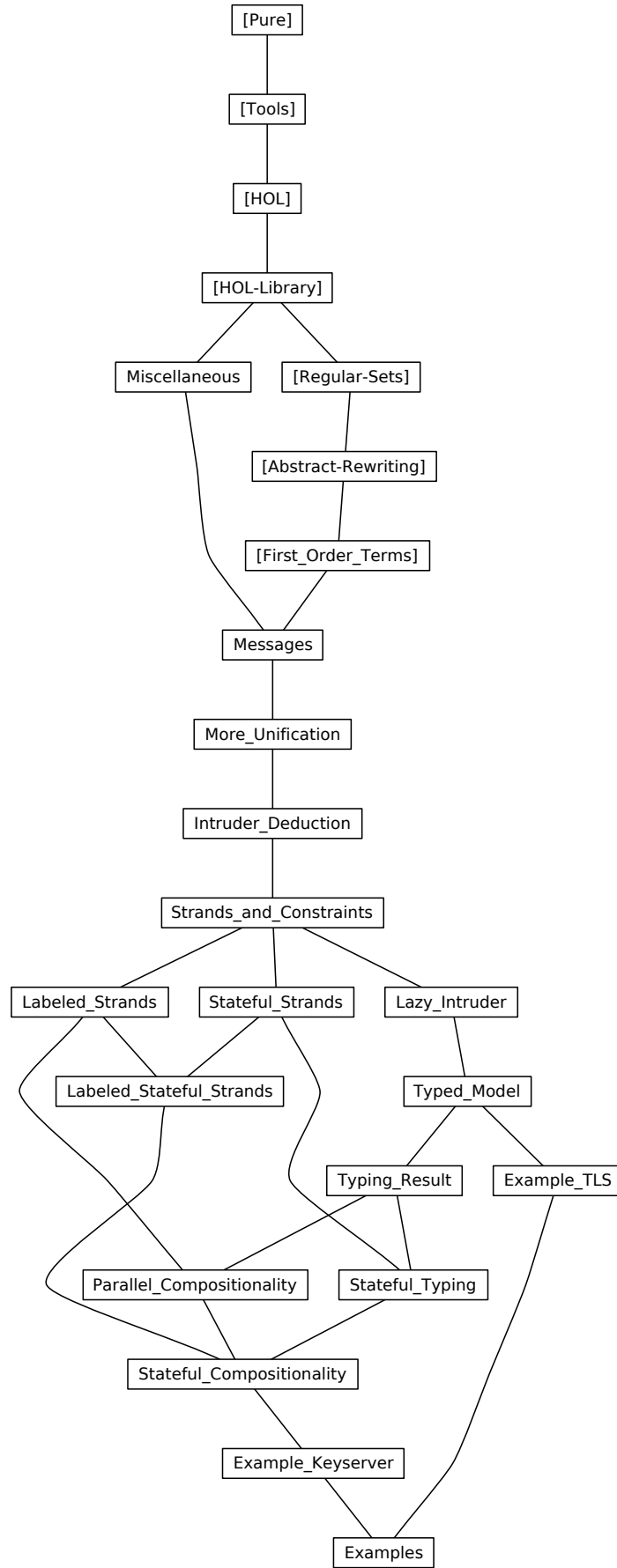


Figure 1.1: The Dependency Graph of the Isabelle Theories.



## 2 Preliminaries and Intruder Model

In this chapter, we introduce the formal preliminaries, including the intruder model and related lemmata.

### 2.1 Miscellaneous Lemmata (Miscellaneous)

```
theory Miscellaneous
imports Main "HOL-Library.Sublist" "HOL-Library.While_Combinator"
begin
```

#### 2.1.1 List: zip, filter, map

```
lemma zip_arg_subterm_split:
  assumes "(x,y) ∈ set (zip xs ys)"
  obtains xs' xs'' ys' ys'' where "xs = xs'@x#xs''" "ys = ys'@y#ys''" "length xs' = length ys'"
proof -
  from assms have "∃zs zs' vs vs'. xs = zs@x#zs' ∧ ys = vs@y#vs' ∧ length zs = length vs"
  proof (induction ys arbitrary: xs)
    case (Cons y' ys' xs)
    then obtain x' xs' where x': "xs = x'#xs'"
    by (metis empty_iff list.exhaust list.set(1) set_zip_leftD)
    show ?case
    by (cases "(x, y) ∈ set (zip xs' ys')",
        metis (xs = x'#xs') Cons.IH[of xs'] Cons_eq_appendI list.size(4),
        use Cons.prems x' in fastforce)
  qed simp
  thus ?thesis using that by blast
qed

lemma zip_arg_index:
  assumes "(x,y) ∈ set (zip xs ys)"
  obtains i where "xs ! i = x" "ys ! i = y" "i < length xs" "i < length ys"
proof -
  obtain xs1 xs2 ys1 ys2 where "xs = xs1@x#xs2" "ys = ys1@y#ys2" "length xs1 = length ys1"
  using zip_arg_subterm_split[OF assms] by mouna
  thus ?thesis using nth_append_length[of xs1 x xs2] nth_append_length[of ys1 y ys2] that by simp
qed

lemma filter_nth: "i < length (filter P xs) ⟹ P (filter P xs ! i)"
using nth_mem by force

lemma list_all_filter_eq: "list_all P xs ⟹ filter P xs = xs"
by (metis list_all_iff filter_True)

lemma list_all_filter_nil:
  assumes "list_all P xs"
  and "⋀x. P x ⟹ ¬Q x"
  shows "filter Q xs = []"
using assms by (induct xs) simp_all

lemma list_all_concat: "list_all (list_all f) P ⟷ list_all f (concat P)"
by (induct P) auto

lemma map_upt_index_eq:
  assumes "j < length xs"
  shows "(map (λi. xs ! is i) [0..
```

```

lemma map_snd_list_insert_distrib:
  assumes " $\forall (i,p) \in \text{insert } x (\text{set } xs). \forall (i',p') \in \text{insert } x (\text{set } xs). p = p' \longrightarrow i = i'$ "
  shows " $\text{map snd (List.insert } x \text{ } xs) = \text{List.insert (snd } x) (\text{map snd } xs)$ "
using assms
proof (induction xs rule: List.rev_induct)
  case (snoc y xs)
  hence IH: " $\text{map snd (List.insert } x \text{ } xs) = \text{List.insert (snd } x) (\text{map snd } xs)$ " by fastforce

  obtain iy py where y: " $y = (iy,py)$ " by (metis surj_pair)
  obtain ix px where x: " $x = (ix,px)$ " by (metis surj_pair)

  have "(ix,px)  $\in$  insert  $x$  (set (y#xs))" "(iy,py)  $\in$  insert  $x$  (set (y#xs))" using y x by auto
  hence *: " $iy = ix$ " when " $py = px$ " using that snoc.prem by auto

  show ?case
  proof (cases "px = py")
    case True
    hence "y = x" using * y x by auto
    thus ?thesis using IH by simp
  next
    case False
    hence "y  $\neq$  x" using y x by simp
    have "List.insert x (xs@[y]) = (List.insert x xs)@[y]"
    proof -
      have 1: "insert y (set xs) = set (xs@[y])" by simp
      have 2: " $x \notin \text{insert } y (\text{set } xs) \vee x \in \text{set } xs$ " using <y  $\neq$  x> by blast
      show ?thesis using 1 2 by (metis (no_types) List.insert_def append_Cons insertCI)
    qed
    thus ?thesis using IH y x False by (auto simp add: List.insert_def)
  qed
qed simp

lemma map_append_inv: " $\text{map } f \text{ } xs = ys@zs \implies \exists vs \text{ } ws. xs = vs@ws \wedge \text{map } f \text{ } vs = ys \wedge \text{map } f \text{ } ws = zs$ "
proof (induction xs arbitrary: ys zs)
  case (Cons x xs')
  note prems = Cons.prem
  note IH = Cons.IH

  show ?case
  proof (cases ys)
    case (Cons y ys')
    then obtain vs' ws where *: " $xs' = vs'@ws$ " " $\text{map } f \text{ } vs' = ys'$ " " $\text{map } f \text{ } ws = zs$ "
    using prems IH[of ys' zs] by auto
    hence " $x\#xs' = (x\#vs')@ws$ " " $\text{map } f \text{ } (x\#vs') = y\#ys'$ " using Cons.prem by force+
    thus ?thesis by (metis Cons *(3))
  qed (use prems in simp)
qed simp

```

### 2.1.2 List: subsequences

```

lemma subseqs_set_subset:
  assumes "ys  $\in$  set (subseqs xs)"
  shows "set ys  $\subseteq$  set xs"
using assms subseqs_powset[of xs] by auto

lemma subset_sublist_exists:
  " $ys \subseteq \text{set } xs \implies \exists zs. \text{set } zs = ys \wedge zs \in \text{set (subseqs } xs)$ "
proof (induction xs arbitrary: ys)
  case Cons thus ?case by (metis (no_types, lifting) Pow_iff imageE subseqs_powset)
qed simp

lemma map_subseqs: " $\text{map (map } f) (\text{subseqs } xs) = \text{subseqs (map } f \text{ } xs)$ "

```

```

proof (induct xs)
  case (Cons x xs)
  have "map (Cons (f x)) (map (map f) (subseqs xs)) = map (map f) (map (Cons x) (subseqs xs))"
    by (induct "subseqs xs") auto
  thus ?case by (simp add: Let_def Cons)
qed simp

```

```

lemma subseqs_Cons:
  assumes "ys ∈ set (subseqs xs)"
  shows "ys ∈ set (subseqs (x#xs))"
by (metis assms Un_iff set_append subseqs.simps(2))

```

```

lemma subseqs_subset:
  assumes "ys ∈ set (subseqs xs)"
  shows "set ys ⊆ set xs"
using assms by (metis Pow_iff image_eqI subseqs_powset)

```

### 2.1.3 List: prefixes, suffixes

```

lemma suffix_Cons': "suffix [x] (y#ys) ⟹ suffix [x] ys ∨ (y = x ∧ ys = [])"
using suffix_Cons[of "[x]"] by auto

```

```

lemma prefix_Cons': "prefix (x#xs) (x#ys) ⟹ prefix xs ys"
by simp

```

```

lemma prefix_map: "prefix xs (map f ys) ⟹ ∃zs. prefix zs ys ∧ map f zs = xs"
using map_append_inv unfolding prefix_def by fast

```

```

lemma length_prefix_ex:
  assumes "n ≤ length xs"
  shows "∃ys zs. xs = ys@zs ∧ length ys = n"
  using assms
proof (induction n)
  case (Suc n)
  then obtain ys zs where IH: "xs = ys@zs" "length ys = n" by moura
  hence "length zs > 0" using Suc.prem(1) by auto
  then obtain v vs where v: "zs = v#vs" by (metis Suc_length_conv gr0_conv_Suc)
  hence "length (ys@[v]) = Suc n" using IH(2) by simp
  thus ?case using IH(1) v by (metis append.assoc append_Cons append_Nil)
qed simp

```

```

lemma length_prefix_ex':
  assumes "n < length xs"
  shows "∃ys zs. xs = ys@xs ! n#zs ∧ length ys = n"
proof -
  obtain ys zs where xs: "xs = ys@zs" "length ys = n" using assms length_prefix_ex[of n xs] by moura
  hence "length zs > 0" using assms by auto
  then obtain v vs where v: "zs = v#vs" by (metis Suc_length_conv gr0_conv_Suc)
  hence "(ys@zs) ! n = v" using xs by auto
  thus ?thesis using v xs by auto
qed

```

```

lemma length_prefix_ex2:
  assumes "i < length xs" "j < length xs" "i < j"
  shows "∃ys zs vs. xs = ys@xs ! i#zs@xs ! j#vs ∧ length ys = i ∧ length zs = j - i - 1"
by (smt assms length_prefix_ex' nth_append append.assoc append.simps(2) add_diff_cancel_left'
    diff_Suc_1 length_Cons length_append)

```

### 2.1.4 List: products

```

lemma product_lists_Cons:
  "x#xs ∈ set (product_lists (y#ys)) ⟷ (xs ∈ set (product_lists ys) ∧ x ∈ set y)"
by auto

```

```

lemma product_lists_in_set_nth:
  assumes "xs ∈ set (product_lists ys)"
  shows "∀ i < length ys. xs ! i ∈ set (ys ! i)"
proof -
  have 0: "length ys = length xs" using assms(1) by (simp add: in_set_product_lists_length)
  thus ?thesis using assms
  proof (induction ys arbitrary: xs)
    case (Cons y ys)
    obtain x xs' where xs: "xs = x#xs'" using Cons.prem1(1) by (metis length_Suc_conv)
    hence "xs' ∈ set (product_lists ys) ⇒ ∀ i < length ys. xs' ! i ∈ set (ys ! i)"
      "length ys = length xs'" "x#xs' ∈ set (product_lists (y#ys))"
      using Cons by simp_all
    thus ?case using xs product_lists_Cons[of x xs' y ys] by (simp add: nth_Cons')
  qed simp
qed

```

```

lemma product_lists_in_set_nth':
  assumes "∀ i < length xs. ys ! i ∈ set (xs ! i)"
  and "length xs = length ys"
  shows "ys ∈ set (product_lists xs)"
using assms
proof (induction xs arbitrary: ys)
  case (Cons x xs)
  obtain y ys' where ys: "ys = y#ys'" using Cons.prem1(2) by (metis length_Suc_conv)
  hence "ys' ∈ set (product_lists xs)" "y ∈ set x" "length xs = length ys'"
    using Cons by fastforce+
  thus ?case using ys product_lists_Cons[of y ys' x xs] by (simp add: nth_Cons')
qed simp

```

### 2.1.5 Other Lemmata

```

lemma inv_set_fset: "finite M ⇒ set (inv set M) = M"
unfolding inv_def by (metis (mono_tags) finite_list someI_ex)

```

```

lemma lfp_eqI':
  assumes "mono f"
  and "f C = C"
  and "∀ X ∈ Pow C. f X = X ⇒ X = C"
  shows "lfp f = C"
by (metis PowI assms lfp_lowerbound lfp_unfold subset_refl)

```

```

lemma lfp_while':
  fixes f::"'a set ⇒ 'a set" and M::"'a set"
  defines "N ≡ while (λA. f A ≠ A) f {}"
  assumes f_mono: "mono f"
  and N_finite: "finite N"
  and N_supset: "f N ⊆ N"
  shows "lfp f = N"
proof -
  have *: "f X ⊆ N" when "X ⊆ N" for X using N_supset monoD[OF f_mono that] by blast
  show ?thesis
    using lfp_while[OF f_mono * N_finite]
    by (simp add: N_def)
qed

```

```

lemma lfp_while'':
  fixes f::"'a set ⇒ 'a set" and M::"'a set"
  defines "N ≡ while (λA. f A ≠ A) f {}"
  assumes f_mono: "mono f"
  and lfp_finite: "finite (lfp f)"
  shows "lfp f = N"
proof -

```

```

have *: "f X ⊆ lfp f" when "X ⊆ lfp f" for X
  using lfp_fixpoint[OF f_mono] monoD[OF f_mono that]
  by blast
show ?thesis
  using lfp_while[OF f_mono * lfp_finite]
  by (simp add: N_def)
qed

lemma preordered_finite_set_has_maxima:
  assumes "finite A" "A ≠ {}"
  shows "∃ a::'a::preorder ∈ A. ∀ b ∈ A. ¬(a < b)"
using assms
proof (induction A rule: finite_induct)
  case (insert a A) thus ?case
    by (cases "A = {}" , simp, metis insert_iff order_trans less_le_not_le)
qed simp

lemma partition_index_bij:
  fixes n::nat
  obtains I k where
    "bij_betw I {0..

```

```

shows "finite {ys. length xs = length ys ∧ (∀y ∈ set ys. ∃x ∈ set xs. P x y)}"
proof -
  define Q where "Q ≡ λys. ∀y ∈ set ys. ∃x ∈ set xs. P x y"
  define M where "M ≡ {y. ∃x ∈ set xs. P x y}"

  have 0: "finite M" using assms unfolding M_def by fastforce

  have "Q ys ⟷ set ys ⊆ M"
    "(Q ys ∧ length ys = length xs) ⟷ (length xs = length ys ∧ Q ys)"
  for ys
  unfolding Q_def M_def by auto
  thus ?thesis
    using finite_lists_length_eq[OF 0, of "length xs"]
    unfolding Q_def by presburger
qed

lemma trancl_eqI:
  assumes "∀(a,b) ∈ A. ∀(c,d) ∈ A. b = c ⟶ (a,d) ∈ A"
  shows "A = A+"
proof
  show "A+ ⊆ A"
  proof
    fix x assume x: "x ∈ A+"
    then obtain a b where ab: "x = (a,b)" by (metis surj_pair)
    hence "(a,b) ∈ A+" using x by metis
    hence "(a,b) ∈ A" using assms by (induct rule: trancl_induct) auto
    thus "x ∈ A" using ab by metis
  qed
qed auto

lemma trancl_eqI':
  assumes "∀(a,b) ∈ A. ∀(c,d) ∈ A. b = c ∧ a ≠ d ⟶ (a,d) ∈ A"
  and "∀(a,b) ∈ A. a ≠ b"
  shows "A = {(a,b) ∈ A+. a ≠ b}"
proof
  show "{(a,b) ∈ A+. a ≠ b} ⊆ A"
  proof
    fix x assume x: "x ∈ {(a,b) ∈ A+. a ≠ b}"
    then obtain a b where ab: "x = (a,b)" by (metis surj_pair)
    hence "(a,b) ∈ A+" "a ≠ b" using x by blast+
    hence "(a,b) ∈ A"
    proof (induction rule: trancl_induct)
      case base thus ?case by blast
    next
      case step thus ?case using assms(1) by force
    qed
    thus "x ∈ A" using ab by metis
  qed
qed (use assms(2) in auto)

lemma distinct_concat_idx_disjoint:
  assumes xs: "distinct (concat xs)"
  and ij: "i < length xs" "j < length xs" "i < j"
  shows "set (xs ! i) ∩ set (xs ! j) = {}"
proof -
  obtain ys zs vs where ys: "xs = ys@xs ! i#zs@xs ! j#vs" "length ys = i" "length zs = j - i - 1"
  using length_prefix_ex2[OF ij] by mouna
  thus ?thesis
    using xs concat_append[of "ys@xs ! i#zs" "xs ! j#vs"]
    distinct_append[of "concat (ys@xs ! i#zs)" "concat (xs ! j#vs)"]
    by auto
qed

```

```

lemma remdups_ex2:
  "length (remdups xs) > 1  $\implies \exists a \in \text{set } xs. \exists b \in \text{set } xs. a \neq b$ "
by (metis distinct_Ex1 distinct_remdups less_trans nth_mem set_remdups zero_less_one zero_neq_one)

lemma trancl_minus_refl_idem:
  defines "cl  $\equiv \lambda ts. \{(a,b) \in ts^+. a \neq b\}$ "
  shows "cl (cl ts) = cl ts"
proof -
  have 0: "(ts+)+ = ts+" "cl ts  $\subseteq$  ts+" "(cl ts)+  $\subseteq$  (ts+)+"
  proof -
    show "(ts+)+ = ts+" "cl ts  $\subseteq$  ts+" unfolding cl_def by auto
    thus "(cl ts)+  $\subseteq$  (ts+)+" using trancl_mono[of _ "cl ts" "ts+"] by blast
  qed

  have 1: "t  $\in$  cl (cl ts)" when t: "t  $\in$  cl ts" for t
    using t 0 unfolding cl_def by fast

  have 2: "t  $\in$  cl ts" when t: "t  $\in$  cl (cl ts)" for t
  proof -
    obtain a b where ab: "t = (a,b)" by (metis surj_pair)
    have "t  $\in$  (cl ts)+" and a_neq_b: "a  $\neq$  b" using t unfolding cl_def ab by force+
    hence "t  $\in$  ts+" using 0 by blast
    thus ?thesis using a_neq_b unfolding cl_def ab by blast
  qed

  show ?thesis using 1 2 by blast
qed

```

### 2.1.6 Infinite Paths in Relations as Mappings from Naturals to States

```

context
begin

private fun rel_chain_fun::"nat  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  ('a  $\times$  'a) set  $\Rightarrow$  'a" where
  "rel_chain_fun 0 x _ _ = x"
| "rel_chain_fun (Suc i) x y r = (if i = 0 then y else SOME z. (rel_chain_fun i x y r, z)  $\in$  r)"

lemma infinite_chain_intro:
  fixes r::"('a  $\times$  'a) set"
  assumes " $\forall (a,b) \in r. \exists c. (b,c) \in r$ " "r  $\neq \{\}$ "
  shows " $\exists f. \forall i::\text{nat}. (f\ i, f\ (\text{Suc } i)) \in r$ "
proof -
  from assms(2) obtain a b where "(a,b)  $\in$  r" by auto

  let ?P = " $\lambda i. (\text{rel\_chain\_fun } i\ a\ b\ r, \text{rel\_chain\_fun } (\text{Suc } i)\ a\ b\ r) \in r$ "
  let ?Q = " $\lambda i. \exists z. (\text{rel\_chain\_fun } i\ a\ b\ r, z) \in r$ "

  have base: "?P 0" using (a,b)  $\in$  r by auto

  have step: "?P (Suc i)" when i: "?P i" for i
  proof -
    have "?Q (Suc i)" using assms(1) i by auto
    thus ?thesis using someI_ex[OF (Q (Suc i))] by auto
  qed

  have " $\forall i::\text{nat}. (\text{rel\_chain\_fun } i\ a\ b\ r, \text{rel\_chain\_fun } (\text{Suc } i)\ a\ b\ r) \in r$ "
    using base step nat_induct[of ?P] by simp
  thus ?thesis by fastforce
qed

end

lemma infinite_chain_intro':

```

```

fixes r:: "('a × 'a) set"
assumes base: "∃ b. (x,b) ∈ r" and step: "∀ b. (x,b) ∈ r+ ⟶ (∃ c. (b,c) ∈ r)"
shows "∃ f. ∀ i::nat. (f i, f (Suc i)) ∈ r"
proof -
  let ?s = "{(a,b) ∈ r. a = x ∨ (x,a) ∈ r+}"

  have "?s ≠ {}" using base by auto

  have "∃ c. (b,c) ∈ ?s" when ab: "(a,b) ∈ ?s" for a b
  proof (cases "a = x")
    case False
    hence "(x,a) ∈ r+" using ab by auto
    hence "(x,b) ∈ r+" using ⟨(a,b) ∈ ?s⟩ by auto
    thus ?thesis using step by auto
  qed (use ab step in auto)
  hence "∃ f. ∀ i. (f i, f (Suc i)) ∈ ?s" using infinite_chain_intro[of ?s] ⟨?s ≠ {}⟩ by blast
  thus ?thesis by auto
qed

lemma infinite_chain_mono:
  assumes "S ⊆ T" "∃ f. ∀ i::nat. (f i, f (Suc i)) ∈ S"
  shows "∃ f. ∀ i::nat. (f i, f (Suc i)) ∈ T"
using assms by auto

end

```

## 2.2 Protocol Messages as (First-Order) Terms (Messages)

```

theory Messages
  imports Miscellaneous "First_Order_Terms.Term"
begin

```

### 2.2.1 Term-related definitions: subterms and free variables

```

abbreviation "the_Fun ≡ un_Fun1"
lemmas the_Fun_def = un_Fun1_def

fun subterms:: "('a,'b) term ⇒ ('a,'b) terms" where
  "subterms (Var x) = {Var x}"
| "subterms (Fun f T) = {Fun f T} ∪ (⋃ t ∈ set T. subterms t)"

abbreviation subterm_eq (infix "⊆" 50) where "t' ⊆ t ≡ (t' ∈ subterms t)"
abbreviation subterm (infix "⊂" 50) where "t' ⊂ t ≡ (t' ⊆ t ∧ t' ≠ t)"

abbreviation "subterms_set M ≡ ⋃ (subterms ` M)"
abbreviation subterm_eq_set (infix "⊆set" 50) where "t ⊆set M ≡ (t ∈ subterms_set M)"

abbreviation fv where "fv ≡ vars_term"
lemmas fv_simps = term.simps(17,18)

fun fv_set where "fv_set M = ⋃ (fv ` M)"

abbreviation fv_pair where "fv_pair p ≡ case p of (t,t') ⇒ fv t ∪ fv t'"

fun fv_pairs where "fv_pairs F = ⋃ (fv_pair ` set F)"

abbreviation ground where "ground M ≡ fv_set M = {}"

```

### 2.2.2 Variants that return lists insteads of sets

```

fun fv_list where
  "fv_list (Var x) = [x]"

```



```

| "fv_list (Fun f T) = concat (map fv_list T)"

definition fv_list_pairs where
  "fv_list_pairs F  $\equiv$  concat (map ( $\lambda(t,t').$  fv_list t@fv_list t') F)"

fun subterms_list:: "('a,'b) term  $\Rightarrow$  ('a,'b) term list" where
  "subterms_list (Var x) = [Var x]"
| "subterms_list (Fun f T) = remdups (Fun f T#concat (map subterms_list T))"

lemma fv_list_is_fv: "fv t = set (fv_list t)"
by (induct t) auto

lemma fv_list_pairs_is_fv_pairs: "fv_pairs F = set (fv_list_pairs F)"
by (induct F) (auto simp add: fv_list_is_fv fv_list_pairs_def)

lemma subterms_list_is_subterms: "subterms t = set (subterms_list t)"
by (induct t) auto

```

### 2.2.3 The subterm relation defined as a function

```

fun subterm_of where
  "subterm_of t (Var y) = (t = Var y)"
| "subterm_of t (Fun f T) = (t = Fun f T  $\vee$  list_ex (subterm_of t) T)"

lemma subterm_of_iff_subtermeq[code_unfold]: "t  $\sqsubseteq$  t' = subterm_of t t'"
proof (induction t')
  case (Fun f T) thus ?case
    proof (cases "t = Fun f T")
      case False thus ?thesis
        using Fun.IH subterm_of.simps(2)[of t f T]
        unfolding list_ex_iff by fastforce
    qed simp
  qed simp

lemma subterm_of_ex_set_iff_subtermeqset[code_unfold]: "t  $\sqsubseteq_{\text{set}}$  M = ( $\exists t' \in M.$  subterm_of t t')"
using subterm_of_iff_subtermeq by blast

```

### 2.2.4 The subterm relation is a partial order on terms

```

interpretation "term": order " $\sqsubseteq$ " " $\sqsubset$ "
proof
  show "s  $\sqsubseteq$  s" for s :: "('a,'b) term"
    by (induct s rule: subterms.induct) auto

  show trans: "s  $\sqsubseteq$  t  $\implies$  t  $\sqsubseteq$  u  $\implies$  s  $\sqsubseteq$  u" for s t u :: "('a,'b) term"
    by (induct u rule: subterms.induct) auto

  show "s  $\sqsubseteq$  t  $\implies$  t  $\sqsubseteq$  s  $\implies$  s = t" for s t :: "('a,'b) term"
  proof (induction s arbitrary: t rule: subterms.induct[case_names Var Fun])
    case (Fun f T)
    { assume 0: "t  $\neq$  Fun f T"
      then obtain u:: "('a,'b) term" where u: "u  $\in$  set T" "t  $\sqsubseteq$  u" using Fun.prem(2) by auto
      hence 1: "Fun f T  $\sqsubseteq$  u" using trans[OF Fun.prem(1)] by simp

      have 2: "u  $\sqsubseteq$  Fun f T"
        by (cases u) (use u(1) in force, use u(1) subterms.simps(2)[of f T] in fastforce)
      hence 3: "u = Fun f T" using Fun.IH[OF u(1) _ 1] by simp

      have "u  $\sqsubseteq$  t" using trans[OF 2 Fun.prem(1)] by simp
      hence 4: "u = t" using Fun.IH[OF u(1) _ u(2)] by simp

      have "t = Fun f T" using 3 4 by simp
      hence False using 0 by simp
    }
  qed

```

```

    }
    thus ?case by auto
qed simp
thus "(s  $\sqsubseteq$  t) = (s  $\sqsubseteq$  t  $\wedge$   $\neg$ (t  $\sqsubseteq$  s))" for s t :: "('a,'b) term"
  by blast
qed

```

## 2.2.5 Lemmata concerning subterms and free variables

```

lemma fv_list_pairs_append: "fv_list_pairs (F@G) = fv_list_pairs F@fv_list_pairs G"
by (simp add: fv_list_pairs_def)

```

```

lemma distinct_fv_list_idx_fv_disjoint:
  assumes t: "distinct (fv_list t)" "Fun f T  $\sqsubseteq$  t"
  and ij: "i < length T" "j < length T" "i < j"
  shows "fv (T ! i)  $\cap$  fv (T ! j) = {}"
using t
proof (induction t rule: fv_list.induct)
  case (2 g S)
  have "distinct (fv_list s)" when s: "s  $\in$  set S" for s
  by (metis (no_types, lifting) s "2.prem" (1) concat_append distinct_append
    map_append split_list fv_list.simps (2) concat.simps (2) list.simps (9))
  hence IH: "fv (T ! i)  $\cap$  fv (T ! j) = {}"
  when s: "s  $\in$  set S" "Fun f T  $\sqsubseteq$  s" for s
  using "2.IH" s by blast

  show ?case
proof (cases "Fun f T = Fun g S")
  case True
  define U where "U  $\equiv$  map fv_list T"

  have a: "distinct (concat U)"
  using "2.prem" (1) True unfolding U_def by auto

  have b: "i < length U" "j < length U"
  using ij (1,2) unfolding U_def by simp_all

  show ?thesis
  using b distinct_concat_idx_disjoint[OF a b ij (3)]
    fv_list_is_fv[of "T ! i"] fv_list_is_fv[of "T ! j"]
    unfolding U_def by force
qed (use IH "2.prem" (2) in auto)
qed force

```

```

lemmas subterm_eqI' [intro] = term.eq_refl

```

```

lemma subterm_eqI'' [intro]: "t  $\in$  set T  $\implies$  t  $\sqsubseteq$  Fun f T"
by force

```

```

lemma finite_fv_set [intro]: "finite M  $\implies$  finite (fv_set M)"
by auto

```

```

lemma finite_fun_symbols [simp]: "finite (funs_term t)"
by (induct t) simp_all

```

```

lemma fv_set_mono: "M  $\subseteq$  N  $\implies$  fv_set M  $\subseteq$  fv_set N"
by auto

```

```

lemma subterms_set_mono: "M  $\subseteq$  N  $\implies$  subterms_set M  $\subseteq$  subterms_set N"
by auto

```

```

lemma ground_empty [simp]: "ground {}"
by simp

```

```

lemma ground_subset: "M  $\subseteq$  N  $\implies$  ground N  $\implies$  ground M"
by auto

lemma fv_map_fv_set: " $\bigcup$  (set (map fv L)) = fvset (set L)"
by (induct L) auto

lemma fv_set_union: "fvset (M  $\cup$  N) = fvset M  $\cup$  fvset N"
by auto

lemma finite_subset_Union:
  fixes A::"'a set" and f::"'a  $\Rightarrow$  'b set"
  assumes "finite ( $\bigcup$  a  $\in$  A. f a)"
  shows " $\exists$  B. finite B  $\wedge$  B  $\subseteq$  A  $\wedge$  ( $\bigcup$  b  $\in$  B. f b) = ( $\bigcup$  a  $\in$  A. f a)"
by (metis asms eq_iff finite_subset_image finite_UnionD)

lemma inv_set_fv: "finite M  $\implies$   $\bigcup$  (set (map fv (inv set M))) = fvset M"
using fv_map_fv_set[of "inv set M"] inv_set_fset by auto

lemma ground_subterm: "fv t = {}  $\implies$  t'  $\sqsubseteq$  t  $\implies$  fv t' = {}" by (induct t) auto

lemma empty_fv_not_var: "fv t = {}  $\implies$  t  $\neq$  Var x" by auto

lemma empty_fv_exists_fun: "fv t = {}  $\implies$   $\exists$  f X. t = Fun f X" by (cases t) auto

lemma vars_iff_subtermeq: "x  $\in$  fv t  $\longleftrightarrow$  Var x  $\sqsubseteq$  t" by (induct t) auto

lemma vars_iff_subtermeq_set: "x  $\in$  fvset M  $\longleftrightarrow$  Var x  $\in$  subtermsset M"
using vars_iff_subtermeq[of x] by auto

lemma vars_if_subtermeq_set: "Var x  $\in$  subtermsset M  $\implies$  x  $\in$  fvset M"
by (metis vars_iff_subtermeq_set)

lemma subtermeq_set_if_vars: "x  $\in$  fvset M  $\implies$  Var x  $\in$  subtermsset M"
by (metis vars_iff_subtermeq_set)

lemma vars_iff_subterm_or_eq: "x  $\in$  fv t  $\longleftrightarrow$  Var x  $\sqsubseteq$  t  $\vee$  Var x = t"
by (induct t) (auto simp add: vars_iff_subtermeq)

lemma var_is_subterm: "x  $\in$  fv t  $\implies$  Var x  $\in$  subterms t"
by (simp add: vars_iff_subtermeq)

lemma subterm_is_var: "Var x  $\in$  subterms t  $\implies$  x  $\in$  fv t"
by (simp add: vars_iff_subtermeq)

lemma no_var_subterm: " $\neg$ t  $\sqsubseteq$  Var v" by auto

lemma fun_if_subterm: "t  $\sqsubseteq$  u  $\implies$   $\exists$  f X. u = Fun f X" by (induct u) simp_all

lemma subtermeq_vars_subset: "M  $\sqsubseteq$  N  $\implies$  fv M  $\subseteq$  fv N" by (induct N) auto

lemma fv_subterms[simp]: "fvset (subterms t) = fv t"
by (induct t) auto

lemma fv_subterms_set[simp]: "fvset (subtermsset M) = fvset M"
using subtermeq_vars_subset by auto

lemma fv_subset: "t  $\in$  M  $\implies$  fv t  $\subseteq$  fvset M"
by auto

lemma fv_subset_subterms: "t  $\in$  subtermsset M  $\implies$  fv t  $\subseteq$  fvset M"
using fv_subset fv_subterms_set by metis

```

```

lemma subterms_finite[simp]: "finite (subterms t)" by (induction rule: subterms.induct) auto

lemma subterms_union_finite: "finite M  $\implies$  finite ( $\bigcup t \in M.$  subterms t)"
by (induction rule: subterms.induct) auto

lemma subterms_subset: "t'  $\sqsubseteq$  t  $\implies$  subterms t'  $\subseteq$  subterms t"
by (induction rule: subterms.induct) auto

lemma subterms_subset_set: "M  $\subseteq$  subterms t  $\implies$  subtermsset M  $\subseteq$  subterms t"
by (metis SUP_least contra_subsetD subterms_subset)

lemma subset_subterms_Union[simp]: "M  $\subseteq$  subtermsset M" by auto

lemma in_subterms_Union: "t  $\in$  M  $\implies$  t  $\in$  subtermsset M" using subset_subterms_Union by blast

lemma in_subterms_subset_Union: "t  $\in$  subtermsset M  $\implies$  subterms t  $\subseteq$  subtermsset M"
using subterms_subset by auto

lemma subterm_param_split:
  assumes "t  $\sqsubseteq$  Fun f X"
  shows " $\exists$  pre x suf. t  $\sqsubseteq$  x  $\wedge$  X = pre@x#suf"
proof -
  obtain x where "t  $\sqsubseteq$  x" "x  $\in$  set X" using assms by auto
  then obtain pre suf where "X = pre@x#suf" "x  $\notin$  set pre  $\vee$  x  $\notin$  set suf"
    by (meson split_list_first split_list_last)
  thus ?thesis using <t  $\sqsubseteq$  x> by auto
qed

lemma ground_iff_no_vars: "ground (M::('a,'b) terms)  $\longleftrightarrow$  ( $\forall v.$  Var v  $\notin$  ( $\bigcup m \in M.$  subterms m))"
proof
  assume "ground M"
  hence " $\forall v.$   $\forall m \in M.$  v  $\notin$  fv m" by auto
  hence " $\forall v.$   $\forall m \in M.$  Var v  $\notin$  subterms m" by (simp add: vars_iff_subtermeq)
  thus "( $\forall v.$  Var v  $\notin$  ( $\bigcup m \in M.$  subterms m))" by simp
next
  assume no_vars: " $\forall v.$  Var v  $\notin$  ( $\bigcup m \in M.$  subterms m)"
  moreover
  { assume " $\neg$ ground M"
    then obtain v and m:: "('a,'b) term" where "m  $\in$  M" "fv m  $\neq$  {}" "v  $\in$  fv m" by auto
    hence "Var v  $\in$  (subterms m)" by (simp add: vars_iff_subtermeq)
    hence " $\exists v.$  Var v  $\in$  ( $\bigcup t \in M.$  subterms t)" using <m  $\in$  M> by auto
    hence False using no_vars by simp
  }
  ultimately show "ground M" by blast
qed

lemma index_Fun_subterms_subset[simp]: "i < length T  $\implies$  subterms (T ! i)  $\subseteq$  subterms (Fun f T)"
by auto

lemma index_Fun_fv_subset[simp]: "i < length T  $\implies$  fv (T ! i)  $\subseteq$  fv (Fun f T)"
using subtermeq_vars_subset by fastforce

lemma subterms_union_ground:
  assumes "ground M"
  shows "ground (subtermsset M)"
proof -
  { fix t assume "t  $\in$  M"
    hence "fv t = {}"
      using ground_iff_no_vars[of M] assms
      by auto
    hence " $\forall t' \in$  subterms t. fv t' = {}" using subtermeq_vars_subset[of _ t] by simp
    hence "ground (subterms t)" by auto
  }

```

```

  thus ?thesis by auto
qed

lemma Var_subtermeq: "t  $\sqsubseteq$  Var v  $\implies$  t = Var v" by simp

lemma subtermeq_imp_funs_term_subset: "s  $\sqsubseteq$  t  $\implies$  funs_term s  $\subseteq$  funs_term t"
by (induct t arbitrary: s) auto

lemma subterms_const: "subterms (Fun f []) = {Fun f []}" by simp

lemma subterm_subtermeq_neq: "[t  $\sqsubseteq$  u; u  $\sqsubseteq$  v]  $\implies$  t  $\neq$  v"
by (metis term.eq_iff)

lemma subtermeq_subterm_neq: "[t  $\sqsubseteq$  u; u  $\sqsubseteq$  v]  $\implies$  t  $\neq$  v"
by (metis term.eq_iff)

lemma subterm_size_lt: "x  $\sqsubseteq$  y  $\implies$  size x < size y"
using not_less_eq size_list_estimation by (induct y, simp, fastforce)

lemma in_subterms_eq: "[x  $\in$  subterms y; y  $\in$  subterms x]  $\implies$  subterms x = subterms y"
using term.antisym by auto

lemma Fun_gt_params: "Fun f X  $\notin$  ( $\bigcup$  x  $\in$  set X. subterms x)"
proof -
  have "size_list size X < size (Fun f X)" by simp
  hence "Fun f X  $\notin$  set X" by (meson less_not_refl size_list_estimation)
  hence " $\forall$  x  $\in$  set X. Fun f X  $\notin$  subterms x  $\vee$  x  $\notin$  subterms (Fun f X)"
    by (metis term.antisym[of "Fun f X" _])
  moreover have " $\forall$  x  $\in$  set X. x  $\in$  subterms (Fun f X)" by fastforce
  ultimately show ?thesis by auto
qed

lemma params_subterms[simp]: "set X  $\subseteq$  subterms (Fun f X)" by auto

lemma params_subterms_Union[simp]: "subtermsset (set X)  $\subseteq$  subterms (Fun f X)" by auto

lemma Fun_subterm_inside_params: "t  $\sqsubseteq$  Fun f X  $\longleftrightarrow$  t  $\in$  ( $\bigcup$  x  $\in$  (set X). subterms x)"
using Fun_gt_params by fastforce

lemma Fun_param_is_subterm: "x  $\in$  set X  $\implies$  x  $\sqsubseteq$  Fun f X"
using Fun_subterm_inside_params by fastforce

lemma Fun_param_in_subterms: "x  $\in$  set X  $\implies$  x  $\in$  subterms (Fun f X)"
using Fun_subterm_inside_params by fastforce

lemma Fun_not_in_param: "x  $\in$  set X  $\implies$   $\neg$  Fun f X  $\sqsubseteq$  x"
using term.antisym by fast

lemma Fun_ex_if_subterm: "t  $\sqsubseteq$  s  $\implies$   $\exists$  f T. Fun f T  $\sqsubseteq$  s  $\wedge$  t  $\in$  set T"
proof (induction s)
  case (Fun f T)
  then obtain s' where s': "s'  $\in$  set T" "t  $\sqsubseteq$  s'" by auto
  show ?case
  proof (cases "t = s'")
    case True thus ?thesis using s' by blast
  next
    case False
    thus ?thesis
      using Fun.IH[OF s'(1)] s'(2) term.order_trans[OF _ Fun_param_in_subterms[OF s'(1), of f]]
      by metis
  qed
qed simp

```

```

lemma const_subterm_obtain:
  assumes "fv t = {}"
  obtains c where "Fun c []  $\sqsubseteq$  t"
using assms
proof (induction t)
  case (Fun f T) thus ?case by (cases "T = []") force+
qed simp

lemma const_subterm_obtain': "fv t = {}  $\implies \exists c. \text{Fun } c [] \sqsubseteq t$ "
by (metis const_subterm_obtain)

lemma subterms_singleton:
  assumes "( $\exists v. t = \text{Var } v$ )  $\vee$  ( $\exists f. t = \text{Fun } f []$ )"
  shows "subterms t = {t}"
using assms by (cases t) auto

lemma subtermeq_Var_const:
  assumes "s  $\sqsubseteq$  t"
  shows "t = Var v  $\implies s = \text{Var } v$ " "t = Fun f []  $\implies s = \text{Fun } f []$ "
using assms by fastforce+

lemma subterms_singleton':
  assumes "subterms t = {t}"
  shows "( $\exists v. t = \text{Var } v$ )  $\vee$  ( $\exists f. t = \text{Fun } f []$ )"
proof (cases t)
  case (Fun f T)
  { fix s S assume "T = s#S"
    hence "s  $\in$  subterms t" using Fun by auto
    hence "s = t" using assms by auto
    hence False
      using Fun_param_is_subterm[of s "s#S" f] (T = s#S) Fun
      by auto
  }
  hence "T = []" by (cases T) auto
  thus ?thesis using Fun by simp
qed (simp add: assms)

lemma funs_term_subterms_eq[simp]:
  " $(\bigcup s \in \text{subterms } t. \text{funs\_term } s) = \text{funs\_term } t$ "
  " $(\bigcup s \in \text{subterms}_{\text{set}} M. \text{funs\_term } s) = \bigcup (\text{funs\_term } ' M)$ "
proof -
  show " $\bigwedge t. \bigcup (\text{funs\_term } ' \text{subterms } t) = \text{funs\_term } t$ "
    using term.order_refl subtermeq_imp_funs_term_subset by blast
  thus " $\bigcup (\text{funs\_term } ' (\text{subterms}_{\text{set}} M)) = \bigcup (\text{funs\_term } ' M)$ " by force
qed

lemmas subtermI'[intro] = Fun_param_is_subterm

lemma funs_term_Fun_subterm: "f  $\in$  funs_term t  $\implies \exists T. \text{Fun } f T \in \text{subterms } t$ "
proof (induction t)
  case (Fun g T)
  hence "f = g  $\vee$  ( $\exists s \in \text{set } T. f \in \text{funs\_term } s$ )" by simp
  thus ?case
  proof
    assume " $\exists s \in \text{set } T. f \in \text{funs\_term } s$ "
    then obtain s where "s  $\in$  set T" " $\exists T. \text{Fun } f T \in \text{subterms } s$ " using Fun.IH by auto
    thus ?thesis by auto
  qed (auto simp add: Fun)
qed simp

lemma funs_term_Fun_subterm': "Fun f T  $\in$  subterms t  $\implies f \in \text{funs\_term } t$ "
by (induct t) auto

```

```

lemma zip_arg_subterm:
  assumes "(s,t) ∈ set (zip X Y)"
  shows "s ⊆ Fun f X" "t ⊆ Fun g Y"
proof -
  from assms have *: "s ∈ set X" "t ∈ set Y" by (meson in_set_zipE)+
  show "s ⊆ Fun f X" by (metis Fun_param_is_subterm[OF *(1)])
  show "t ⊆ Fun g Y" by (metis Fun_param_is_subterm[OF *(2)])
qed

lemma fv_disj_Fun_subterm_param_cases:
  assumes "fv t ∩ X = {}" "Fun f T ∈ subterms t"
  shows "T = [] ∨ (∃ s ∈ set T. s ∉ Var 'X)"
proof (cases T)
  case (Cons s S)
  hence "s ∈ subterms t"
    using assms(2) term.order_trans[of _ "Fun f T" t]
    by auto
  hence "fv s ∩ X = {}" using assms(1) fv_subterms by force
  thus ?thesis using Cons by auto
qed simp

lemma fv_eq_FunI:
  assumes "length T = length S" "∧i. i < length T ⟹ fv (T ! i) = fv (S ! i)"
  shows "fv (Fun f T) = fv (Fun g S)"
using assms
proof (induction T arbitrary: S)
  case (Cons t T S')
  then obtain s S where S': "S' = s#S" by (cases S') simp_all
  thus ?case using Cons by fastforce
qed simp

lemma fv_eq_FunI':
  assumes "length T = length S" "∧i. i < length T ⟹ x ∈ fv (T ! i) ⟷ x ∈ fv (S ! i)"
  shows "x ∈ fv (Fun f T) ⟷ x ∈ fv (Fun g S)"
using assms
proof (induction T arbitrary: S)
  case (Cons t T S')
  then obtain s S where S': "S' = s#S" by (cases S') simp_all
  thus ?case using Cons by fastforce
qed simp

lemma finite_fv_pairs[simp]: "finite (fv_pairs x)" by auto

lemma fv_pairs_Nil[simp]: "fv_pairs [] = {}" by simp

lemma fv_pairs_singleton[simp]: "fv_pairs [(t,s)] = fv t ∪ fv s" by simp

lemma fv_pairs_Cons: "fv_pairs ((s,t)#F) = fv s ∪ fv t ∪ fv_pairs F" by simp

lemma fv_pairs_append: "fv_pairs (F@G) = fv_pairs F ∪ fv_pairs G" by simp

lemma fv_pairs_mono: "set M ⊆ set N ⟹ fv_pairs M ⊆ fv_pairs N" by auto

lemma fv_pairs_inI[intro]:
  "f ∈ set F ⟹ x ∈ fv_pair f ⟹ x ∈ fv_pairs F"
  "f ∈ set F ⟹ x ∈ fv (fst f) ⟹ x ∈ fv_pairs F"
  "f ∈ set F ⟹ x ∈ fv (snd f) ⟹ x ∈ fv_pairs F"
  "(t,s) ∈ set F ⟹ x ∈ fv t ⟹ x ∈ fv_pairs F"
  "(t,s) ∈ set F ⟹ x ∈ fv s ⟹ x ∈ fv_pairs F"
using UN_I by fastforce+

lemma fv_pairs_cons_subset: "fv_pairs F ⊆ fv_pairs (f#F)"
by auto

```

### 2.2.6 Other lemmata

```

lemma nonvar_term_has_composed_shallow_term:
  fixes t:: "('f, 'v) term"
  assumes "¬(∃x. t = Var x)"
  shows "∃f T. Fun f T ⊆ t ∧ (∀s ∈ set T. (∃c. s = Fun c []) ∨ (∃x. s = Var x))"
proof -
  let ?Q = "λS. ∀s ∈ set S. (∃c. s = Fun c []) ∨ (∃x. s = Var x)"
  let ?P = "λt. ∃g S. Fun g S ⊆ t ∧ ?Q S"
  { fix t:: "('f, 'v) term"
    have "(∃x. t = Var x) ∨ ?P t"
    proof (induction t)
      case (Fun h R) show ?case
        proof (cases "R = [] ∨ (∀r ∈ set R. ∃x. r = Var x)")
          case False
            then obtain r g S where "r ∈ set R" "?P r" "Fun g S ⊆ r" "?Q S" using Fun.IH by fast
            thus ?thesis by auto
          case True
            qed force
        qed simp
      } thus ?thesis using assms by blast
    qed
  end

```

## 2.3 Definitions and Properties Related to Substitutions and Unification (More\_Unification)

```

theory More_Unification
  imports Messages "First_Order_Terms.Unification"
begin

```

### 2.3.1 Substitutions

```

abbreviation subst_apply_list (infix ".list" 51) where
  "T .list ∅ ≡ map (λt. t · ∅) T"

abbreviation subst_apply_pair (infixl ".p" 60) where
  "d ·p ∅ ≡ (case d of (t, t') ⇒ (t · ∅, t' · ∅))"

abbreviation subst_apply_pair_set (infixl ".pset" 60) where
  "M ·pset ∅ ≡ (λd. d ·p ∅) ' M"

definition subst_apply_pairs (infix ".pairs" 51) where
  "F ·pairs ∅ ≡ map (λf. f ·p ∅) F"

abbreviation subst_more_general_than (infixl "≼o" 50) where
  "σ ≼o ∅ ≡ ∃γ. ∅ = σ ∘s γ"

abbreviation subst_support (infix "supports" 50) where
  "∅ supports δ ≡ (∀x. ∅ x · δ = δ x)"

abbreviation rm_var where
  "rm_var v s ≡ s(v := Var v)"

abbreviation rm_vars where
  "rm_vars vs σ ≡ (λv. if v ∈ vs then Var v else σ v)"

definition subst_elim where
  "subst_elim σ v ≡ ∀t. v ∉ fv (t · σ)"

definition subst_idem where
  "subst_idem s ≡ s ∘s s = s"

```



```

lemma subst_support_def: " $\vartheta$  supports  $\tau \longleftrightarrow \tau = \vartheta \circ_s \tau$ "
unfolding subst_compose_def by metis

lemma subst_supportD: " $\vartheta$  supports  $\delta \implies \vartheta \preceq_o \delta$ "
using subst_support_def by auto

lemma rm_vars_empty[simp]: "rm_vars {} s = s" "rm_vars (set []) s = s"
by simp_all

lemma rm_vars_singleton: "rm_vars {v} s = rm_var v s"
by auto

lemma subst_apply_terms_empty: "M ·set Var = M"
by simp

lemma subst_agreement: "(t · r = t · s)  $\longleftrightarrow$  ( $\forall v \in \text{fv } t. \text{Var } v \cdot r = \text{Var } v \cdot s$ )"
by (induct t) auto

lemma repl_invariance[dest?]: " $v \notin \text{fv } t \implies t \cdot s(v := u) = t \cdot s$ "
by (simp add: subst_agreement)

lemma subst_idx_map:
  assumes " $\forall i \in \text{set } I. i < \text{length } T$ "
  shows "(map (!! T) I) ·list  $\delta$  = map (!! (map ( $\lambda t. t \cdot \delta$ ) T)) I"
using assms by auto

lemma subst_idx_map':
  assumes " $\forall i \in \text{fv}_{\text{set}} (\text{set } K). i < \text{length } T$ "
  shows "(K ·list (!) T) ·list  $\delta$  = K ·list (!! (map ( $\lambda t. t \cdot \delta$ ) T))" (is "?A = ?B")
proof -
  have "T ! i ·  $\delta$  = (map ( $\lambda t. t \cdot \delta$ ) T) ! i"
  when "i < length T" for i
  using that by auto
  hence "T ! i ·  $\delta$  = (map ( $\lambda t. t \cdot \delta$ ) T) ! i"
  when "i ∈ fvset (set K)" for i
  using that assms by auto
  hence "k · (!) T ·  $\delta$  = k · (!) (map ( $\lambda t. t \cdot \delta$ ) T)"
  when "fv k ⊆ fvset (set K)" for k
  using that by (induction k) force+
  thus ?thesis by auto
qed

lemma subst_remove_var: " $v \notin \text{fv } s \implies v \notin \text{fv } (t \cdot \text{Var}(v := s))$ "
by (induct t) simp_all

lemma subst_set_map: " $x \in \text{set } X \implies x \cdot s \in \text{set } (\text{map } (\lambda x. x \cdot s) X)$ "
by simp

lemma subst_set_idx_map:
  assumes " $\forall i \in I. i < \text{length } T$ "
  shows "(!! T ' I ·set  $\delta$  = (!! (map ( $\lambda t. t \cdot \delta$ ) T) ' I" (is "?A = ?B")
proof
  have *: "T ! i ·  $\delta$  = (map ( $\lambda t. t \cdot \delta$ ) T) ! i"
  when "i < length T" for i
  using that by auto

  show "?A ⊆ ?B" using * assms by blast
  show "?B ⊆ ?A" using * assms by auto
qed

lemma subst_set_idx_map':
  assumes " $\forall i \in \text{fv}_{\text{set}} K. i < \text{length } T$ "

```

```

shows "K ·set (!) T ·set δ = K ·set (!) (map (λt. t · δ) T)" (is "?A = ?B")
proof
  have "T ! i · δ = (map (λt. t · δ) T) ! i"
    when "i < length T" for i
    using that by auto
  hence "T ! i · δ = (map (λt. t · δ) T) ! i"
    when "i ∈ fvset K" for i
    using that assms by auto
  hence *: "k · (!) T · δ = k · (!) (map (λt. t · δ) T)"
    when "fv k ⊆ fvset K" for k
    using that by (induction k) force+

  show "?A ⊆ ?B" using * by auto
  show "?B ⊆ ?A" using * by force
qed

lemma subst_term_list_obtain:
  assumes "∀i < length T. ∃s. P (T ! i) s ∧ S ! i = s · δ"
    and "length T = length S"
  shows "∃U. length T = length U ∧ (∀i < length T. P (T ! i) (U ! i)) ∧ S = map (λu. u · δ) U"
using assms
proof (induction T arbitrary: S)
  case (Cons t T S')
  then obtain s S where S': "S' = s#S" by (cases S') auto

  have "∀i < length T. ∃s. P (T ! i) s ∧ S ! i = s · δ" "length T = length S"
    using Cons.prem1 S' by force+
  then obtain U where U:
    "length T = length U" "∀i < length T. P (T ! i) (U ! i)" "S = map (λu. u · δ) U"
    using Cons.IH by mouna

  obtain u where u: "P t u" "s = u · δ"
    using Cons.prem2 S' by auto

  have 1: "length (t#T) = length (u#U)"
    using Cons.prem2 U(1) by fastforce

  have 2: "∀i < length (t#T). P ((t#T) ! i) ((u#U) ! i)"
    using u(1) U(2) by (simp add: nth_Cons')

  have 3: "S' = map (λu. u · δ) (u#U)"
    using U u S' by simp

  show ?case using 1 2 3 by blast
qed simp

lemma subst_mono: "t ⊆ u ⟹ t · s ⊆ u · s"
by (induct u) auto

lemma subst_mono_fv: "x ∈ fv t ⟹ s x ⊆ t · s"
by (induct t) auto

lemma subst_mono_neq:
  assumes "t ⊆ u"
  shows "t · s ⊆ u · s"
proof (cases u)
  case (Var v)
  hence False using ⟨t ⊆ u⟩ by simp
  thus ?thesis ..
next
  case (Fun f X)
  then obtain x where "x ∈ set X" "t ⊆ x" using ⟨t ⊆ u⟩ by auto
  hence "t · s ⊆ x · s" using subst_mono by metis

```

```

obtain Y where "Fun f X · s = Fun f Y" by auto
hence "x · s ∈ set Y" using ⟨x ∈ set X⟩ by auto
hence "x · s ⊆ Fun f X · s" using ⟨Fun f X · s = Fun f Y⟩ Fun_param_is_subterm by simp
hence "t · s ⊆ Fun f X · s" using ⟨t · s ⊆ x · s⟩ by (metis term.dual_order.trans term.eq_iff)
thus ?thesis using ⟨u = Fun f X⟩ ⟨t ⊆ u⟩ by metis
qed

```

```

lemma subst_no_occs[dest]: "¬Var v ⊆ t ⇒ t · Var(v := s) = t"
by (induct t) (simp_all add: map_idI)

```

```

lemma var_comp[simp]: "σ ∘s Var = σ" "Var ∘s σ = σ"
unfolding subst_compose_def by simp_all

```

```

lemma subst_comp_all: "M ·set (δ ∘s ϑ) = (M ·set δ) ·set ϑ"
using subst_subst_compose[of _ δ ϑ] by auto

```

```

lemma subst_all_mono: "M ⊆ M' ⇒ M ·set s ⊆ M' ·set s"
by auto

```

```

lemma subst_comp_set_image: "(δ ∘s ϑ) ' X = δ ' X ·set ϑ"
using subst_compose by fastforce

```

```

lemma subst_ground_ident[dest?]: "fv t = {} ⇒ t · s = t"
by (induct t, simp, metis subst_agreement empty_iff subst_apply_term_empty)

```

```

lemma subst_ground_ident_compose:
  "fv (σ x) = {} ⇒ (σ ∘s ϑ) x = σ x"
  "fv (t · σ) = {} ⇒ t · (σ ∘s ϑ) = t · σ"
using subst_subst_compose[of t σ ϑ]
by (simp_all add: subst_compose_def subst_ground_ident)

```

```

lemma subst_all_ground_ident[dest?]: "ground M ⇒ M ·set s = M"
proof -
  assume "ground M"
  hence "⋀t. t ∈ M ⇒ fv t = {}" by auto
  hence "⋀t. t ∈ M ⇒ t · s = t" by (metis subst_ground_ident)
  moreover have "⋀t. t ∈ M ⇒ t · s ∈ M ·set s" by (metis imageI)
  ultimately show "M ·set s = M" by (simp add: image_cong)
qed

```

```

lemma subst_eqI[intro]: "(⋀t. t · σ = t · ϑ) ⇒ σ = ϑ"
proof -
  assume "⋀t. t · σ = t · ϑ"
  hence "⋀v. Var v · σ = Var v · ϑ" by auto
  thus "σ = ϑ" by auto
qed

```

```

lemma subst_cong: "[σ = σ'; ϑ = ϑ'] ⇒ (σ ∘s ϑ) = (σ' ∘s ϑ')"
by auto

```

```

lemma subst_mgt_bot[simp]: "Var ≤o ϑ"
by simp

```

```

lemma subst_mgt_refl[simp]: "ϑ ≤o ϑ"
by (metis var_comp(1))

```

```

lemma subst_mgt_trans: "[ϑ ≤o δ; δ ≤o σ] ⇒ ϑ ≤o σ"
by (metis subst_compose_assoc)

```

```

lemma subst_mgt_comp: "ϑ ≤o ϑ ∘s δ"
by auto

```

```

lemma subst_mgt_comp': " $\vartheta \circ_s \delta \preceq_o \sigma \implies \vartheta \preceq_o \sigma$ "
by (metis subst_compose_assoc)

lemma var_self: " $(\lambda w. \text{if } w = v \text{ then Var } v \text{ else Var } w) = \text{Var}$ "
using subst_agreement by auto

lemma var_same[simp]: " $\text{Var}(v := t) = \text{Var} \longleftrightarrow t = \text{Var } v$ "
by (intro iffI, metis fun_upd_same, simp add: var_self)

lemma subst_eq_if_eq_vars: " $(\bigwedge v. (\text{Var } v) \cdot \vartheta = (\text{Var } v) \cdot \sigma) \implies \vartheta = \sigma$ "
by (auto simp add: subst_agreement)

lemma subst_all_empty[simp]: " $\{\} \cdot_{set} \vartheta = \{\}$ "
by simp

lemma subst_all_insert: " $(\text{insert } t \ M) \cdot_{set} \delta = \text{insert } (t \cdot \delta) \ (M \cdot_{set} \delta)$ "
by auto

lemma subst_apply_fv_subset: " $\text{fv } t \subseteq V \implies \text{fv } (t \cdot \delta) \subseteq \text{fv}_{set} (\delta \text{ ' } V)$ "
by (induct t) auto

lemma subst_apply_fv_empty:
  assumes "fv t = {"
  shows "fv (t · σ) = {"
using assms subst_apply_fv_subset[of t "{" σ]
by auto

lemma subst_compose_fv:
  assumes "fv (ϑ x) = {"
  shows "fv ((ϑ ∘s σ) x) = {"
using assms subst_apply_fv_empty
unfolding subst_compose_def by fast

lemma subst_compose_fv':
  fixes ϑ σ :: "('a, 'b) subst"
  assumes "y ∈ fv ((ϑ ∘s σ) x)"
  shows "∃ z. z ∈ fv (ϑ x)"
using assms subst_compose_fv
by fast

lemma subst_apply_fv_unfold: " $\text{fv } (t \cdot \delta) = \text{fv}_{set} (\delta \text{ ' } \text{fv } t)$ "
by (induct t) auto

lemma subst_apply_fv_unfold': " $\text{fv } (t \cdot \delta) = (\bigcup v \in \text{fv } t. \text{fv } (\delta \text{ ' } v))$ "
using subst_apply_fv_unfold by simp

lemma subst_apply_fv_union: " $\text{fv}_{set} (\delta \text{ ' } V) \cup \text{fv } (t \cdot \delta) = \text{fv}_{set} (\delta \text{ ' } (V \cup \text{fv } t))$ "
proof -
  have " $\text{fv}_{set} (\delta \text{ ' } (V \cup \text{fv } t)) = \text{fv}_{set} (\delta \text{ ' } V) \cup \text{fv}_{set} (\delta \text{ ' } \text{fv } t)$ " by auto
  thus ?thesis using subst_apply_fv_unfold by metis
qed

lemma subst_elimI[intro]: " $(\bigwedge t. v \notin \text{fv } (t \cdot \sigma)) \implies \text{subst\_elim } \sigma \ v$ "
by (auto simp add: subst_elim_def)

lemma subst_elimI'[intro]: " $(\bigwedge w. v \notin \text{fv } (\text{Var } w \cdot \vartheta)) \implies \text{subst\_elim } \vartheta \ v$ "
by (simp add: subst_elim_def subst_apply_fv_unfold')

lemma subst_elimD[dest]: " $\text{subst\_elim } \sigma \ v \implies v \notin \text{fv } (t \cdot \sigma)$ "
by (auto simp add: subst_elim_def)

lemma subst_elimD'[dest]: " $\text{subst\_elim } \sigma \ v \implies \sigma \ v \neq \text{Var } v$ "
by (metis subst_elim_def subst_apply_term.simps(1) term.set_intros(3))

```

```

lemma subst_elimD''[dest]: "subst_elim  $\sigma$  v  $\implies$  v  $\notin$  fv ( $\sigma$  w)"
by (metis subst_elim_def subst_apply_term.simps(1))

lemma subst_elim_rm_vars_dest[dest]:
  "subst_elim ( $\sigma :: ('a, 'b)$  subst) v  $\implies$  v  $\notin$  vs  $\implies$  subst_elim (rm_vars vs  $\sigma$ ) v"
proof -
  assume assms: "subst_elim  $\sigma$  v" "v  $\notin$  vs"
  obtain f::('a, 'b) subst  $\Rightarrow$  'b  $\Rightarrow$  'b" where
    " $\forall \sigma$  v. ( $\exists w$ . v  $\in$  fv (Var w  $\cdot$   $\sigma$ )) = (v  $\in$  fv (Var (f  $\sigma$  v)  $\cdot$   $\sigma$ ))"
  by moura
  hence *: " $\forall a$   $\sigma$ . a  $\in$  fv (Var (f  $\sigma$  a)  $\cdot$   $\sigma$ )  $\vee$  subst_elim  $\sigma$  a" by blast
  have "Var (f (rm_vars vs  $\sigma$ ) v)  $\cdot$   $\sigma \neq$  Var (f (rm_vars vs  $\sigma$ ) v)  $\cdot$  rm_vars vs  $\sigma$ "
     $\vee$  v  $\notin$  fv (Var (f (rm_vars vs  $\sigma$ ) v)  $\cdot$  rm_vars vs  $\sigma$ )"
  using assms(1) by fastforce
  moreover
  { assume "Var (f (rm_vars vs  $\sigma$ ) v)  $\cdot$   $\sigma \neq$  Var (f (rm_vars vs  $\sigma$ ) v)  $\cdot$  rm_vars vs  $\sigma$ "
    hence "rm_vars vs  $\sigma$  (f (rm_vars vs  $\sigma$ ) v)  $\neq$   $\sigma$  (f (rm_vars vs  $\sigma$ ) v)" by auto
    hence "f (rm_vars vs  $\sigma$ ) v  $\in$  vs" by meson
    hence ?thesis using * assms(2) by force
  }
  ultimately show ?thesis using * by blast
qed

lemma occs_subst_elim: " $\neg$ Var v  $\sqsubseteq$  t  $\implies$  subst_elim (Var(v := t)) v  $\vee$  (Var(v := t)) = Var"
proof (cases "Var v = t")
  assume "Var v  $\neq$  t" " $\neg$ Var v  $\sqsubseteq$  t"
  hence "v  $\notin$  fv t" by (simp add: vars_iff_subterm_or_eq)
  thus ?thesis by (auto simp add: subst_remove_var)
qed auto

lemma occs_subst_elim': " $\neg$ Var v  $\sqsubseteq$  t  $\implies$  subst_elim (Var(v := t)) v"
proof -
  assume " $\neg$ Var v  $\sqsubseteq$  t"
  hence "v  $\notin$  fv t" by (auto simp add: vars_iff_subterm_or_eq)
  thus "subst_elim (Var(v := t)) v" by (simp add: subst_elim_def subst_remove_var)
qed

lemma subst_elim_comp: "subst_elim  $\vartheta$  v  $\implies$  subst_elim ( $\delta \circ_s \vartheta$ ) v"
by (auto simp add: subst_elim_def)

lemma var_subst_idem: "subst_idem Var"
by (simp add: subst_idem_def)

lemma var_upd_subst_idem:
  assumes " $\neg$ Var v  $\sqsubseteq$  t" shows "subst_idem (Var(v := t))"
unfolding subst_idem_def
proof
  let ? $\vartheta$  = "Var(v := t)"
  from assms have t_ $\vartheta$ _id: "t  $\cdot$  ? $\vartheta$  = t" by blast
  fix s show "s  $\cdot$  (? $\vartheta$   $\circ_s$  ? $\vartheta$ ) = s  $\cdot$  ? $\vartheta$ "
    unfolding subst_compose_def
    by (induction s, metis t_ $\vartheta$ _id fun_upd_def subst_apply_term.simps(1), simp)
qed

```

### 2.3.2 Lemmata: Domain and Range of Substitutions

```

lemma range_vars_alt_def: "range_vars s  $\equiv$  fvset (subst_range s)"
unfolding range_vars_def by simp

lemma subst_dom_var_finite[simp]: "finite (subst_domain Var)" by simp

lemma subst_range_Var[simp]: "subst_range Var = {}" by simp

```

```

lemma range_vars_Var[simp]: "range_vars Var = {}" by fastforce

lemma finite_subst_img_if_finite_dom: "finite (subst_domain  $\sigma$ )  $\implies$  finite (range_vars  $\sigma$ )"
unfolding range_vars_alt_def by auto

lemma finite_subst_img_if_finite_dom': "finite (subst_domain  $\sigma$ )  $\implies$  finite (subst_range  $\sigma$ )"
by auto

lemma subst_img_alt_def: "subst_range s = {t.  $\exists v. s\ v = t \wedge t \neq \text{Var } v$ }"
by (auto simp add: subst_domain_def)

lemma subst_fv_img_alt_def: "range_vars s = ( $\bigcup t \in \{t. \exists v. s\ v = t \wedge t \neq \text{Var } v\}. \text{fv } t$ )"
unfolding range_vars_alt_def by (auto simp add: subst_domain_def)

lemma subst_domI[intro]: " $\sigma\ v \neq \text{Var } v \implies v \in \text{subst\_domain } \sigma$ "
by (simp add: subst_domain_def)

lemma subst_imgI[intro]: " $\sigma\ v \neq \text{Var } v \implies \sigma\ v \in \text{subst\_range } \sigma$ "
by (simp add: subst_domain_def)

lemma subst_fv_imgI[intro]: " $\sigma\ v \neq \text{Var } v \implies \text{fv } (\sigma\ v) \subseteq \text{range\_vars } \sigma$ "
unfolding range_vars_alt_def by auto

lemma subst_domain_subst_Fun_single[simp]:
  "subst_domain (Var(x := Fun f T)) = {x}" (is "?A = ?B")
unfolding subst_domain_def by simp

lemma subst_range_subst_Fun_single[simp]:
  "subst_range (Var(x := Fun f T)) = {Fun f T}" (is "?A = ?B")
by simp

lemma range_vars_subst_Fun_single[simp]:
  "range_vars (Var(x := Fun f T)) = fv (Fun f T)"
unfolding range_vars_alt_def by force

lemma var_renaming_is_Fun_iff:
  assumes "subst_range  $\delta \subseteq \text{range Var}$ "
  shows "is_Fun t = is_Fun (t  $\cdot$   $\delta$ )"
proof (cases t)
  case (Var x)
  hence " $\exists y. \delta\ x = \text{Var } y$ " using assms by auto
  thus ?thesis using Var by auto
qed simp

lemma subst_fv_dom_img_subset: "fv t  $\subseteq$  subst_domain  $\vartheta \implies \text{fv } (t \cdot \vartheta) \subseteq \text{range\_vars } \vartheta$ "
unfolding range_vars_alt_def by (induct t) auto

lemma subst_fv_dom_img_subset_set: "fvset M  $\subseteq$  subst_domain  $\vartheta \implies \text{fv}_{set} (M \cdot_{set} \vartheta) \subseteq \text{range\_vars } \vartheta$ "
proof -
  assume assms: "fvset M  $\subseteq$  subst_domain  $\vartheta$ "
  obtain f::"'a set  $\Rightarrow$  (('b, 'a) term  $\Rightarrow$  'a set)  $\Rightarrow$  (('b, 'a) terms  $\Rightarrow$  ('b, 'a) term)" where
    " $\forall x\ y\ z. (\exists v. v \in z \wedge \neg y\ v \subseteq x) \iff (f\ x\ y\ z \in z \wedge \neg y\ (f\ x\ y\ z) \subseteq x)$ "
  by moura
  hence *:
    " $\forall T\ g\ A. (\neg \bigcup (g\ 'T) \subseteq A \vee (\forall t. t \notin T \vee g\ t \subseteq A)) \wedge$ 
       $(\bigcup (g\ 'T) \subseteq A \vee f\ A\ g\ T \in T \wedge \neg g\ (f\ A\ g\ T) \subseteq A)$ "
  by (metis (no_types) SUP_le_iff)
  hence **: " $\forall t. t \notin M \vee \text{fv } t \subseteq \text{subst\_domain } \vartheta$ " by (metis (no_types) assms fvset.sims)
  have " $\forall t::('b, 'a) \text{ term}. \forall f\ T. t \notin f\ 'T \vee (\exists t':('b, 'a) \text{ term}. t = f\ t' \wedge t' \in T)$ " by blast
  hence "f (range_vars  $\vartheta$ ) fv (M  $\cdot_{set} \vartheta$ )  $\notin$  M  $\cdot_{set} \vartheta \vee$ 
    fv (f (range_vars  $\vartheta$ ) fv (M  $\cdot_{set} \vartheta$ ))  $\subseteq$  range_vars  $\vartheta$ "
  by (metis (full_types) ** subst_fv_dom_img_subset)

```

```

  thus ?thesis by (metis (no_types) * fv_set.simps)
qed

lemma subst_fv_dom_ground_if_ground_img:
  assumes "fv t  $\subseteq$  subst_domain s" "ground (subst_range s)"
  shows "fv (t  $\cdot$  s) = {}"
using subst_fv_dom_img_subset[OF assms(1)] assms(2) by force

lemma subst_fv_dom_ground_if_ground_img':
  assumes "fv t  $\subseteq$  subst_domain s" " $\bigwedge x. x \in \text{subst\_domain } s \implies \text{fv } (s \ x) = \{\}$ "
  shows "fv (t  $\cdot$  s) = {}"
using subst_fv_dom_ground_if_ground_img[OF assms(1)] assms(2) by auto

lemma subst_fv_unfold: "fv (t  $\cdot$  s) = (fv t - subst_domain s)  $\cup$  fv_set (s ' (fv t  $\cap$  subst_domain s))"
proof (induction t)
  case (Var v) thus ?case
  proof (cases "v  $\in$  subst_domain s")
    case True thus ?thesis by auto
    case False
    hence "fv (Var v  $\cdot$  s) = {v}" "fv (Var v)  $\cap$  subst_domain s = {}" by auto
    thus ?thesis by auto
  qed
next
  case Fun thus ?case by auto
qed

lemma subst_fv_unfold_ground_img: "range_vars s = {}  $\implies$  fv (t  $\cdot$  s) = fv t - subst_domain s"
using subst_fv_unfold[of t s] unfolding range_vars_alt_def by auto

lemma subst_img_update:
  "[[ $\sigma \ v = \text{Var } v$ ; t  $\neq \text{Var } v$ ]]  $\implies$  range_vars ( $\sigma(v := t)$ ) = range_vars  $\sigma \cup$  fv t"
proof -
  assume " $\sigma \ v = \text{Var } v$ " "t  $\neq \text{Var } v$ "
  hence "( $\bigcup s \in \{s. \exists w. (\sigma(v := t)) \ w = s \wedge s \neq \text{Var } w\}. \text{fv } s$ ) = fv t  $\cup$  range_vars  $\sigma$ "
  unfolding range_vars_alt_def by (auto simp add: subst_domain_def)
  thus "range_vars ( $\sigma(v := t)$ ) = range_vars  $\sigma \cup$  fv t"
  by (metis Un_commute subst_fv_img_alt_def)
qed

lemma subst_dom_update1: "v  $\notin$  subst_domain  $\sigma \implies$  subst_domain ( $\sigma(v := \text{Var } v)$ ) = subst_domain  $\sigma$ "
by (auto simp add: subst_domain_def)

lemma subst_dom_update2: "t  $\neq \text{Var } v \implies$  subst_domain ( $\sigma(v := t)$ ) = insert v (subst_domain  $\sigma$ )"
by (auto simp add: subst_domain_def)

lemma subst_dom_update3: "t = Var v  $\implies$  subst_domain ( $\sigma(v := t)$ ) = subst_domain  $\sigma - \{v\}$ "
by (auto simp add: subst_domain_def)

lemma var_not_in_subst_dom[elim]: "v  $\notin$  subst_domain s  $\implies$  s v = Var v"
by (simp add: subst_domain_def)

lemma subst_dom_vars_in_subst[elim]: "v  $\in$  subst_domain s  $\implies$  s v  $\neq$  Var v"
by (simp add: subst_domain_def)

lemma subst_not_dom_fixed: "[[v  $\in$  fv t; v  $\notin$  subst_domain s]]  $\implies$  v  $\in$  fv (t  $\cdot$  s)" by (induct t) auto

lemma subst_not_img_fixed: "[[v  $\in$  fv (t  $\cdot$  s); v  $\notin$  range_vars s]]  $\implies$  v  $\in$  fv t"
unfolding range_vars_alt_def by (induct t) force+

lemma ground_range_vars[intro]: "ground (subst_range s)  $\implies$  range_vars s = {}"
unfolding range_vars_alt_def by metis

```

```

lemma ground_subst_no_var[intro]: "ground (subst_range s)  $\implies$  x  $\notin$  range_vars s"
using ground_range_vars[of s] by blast

lemma ground_img_obtain_fun:
  assumes "ground (subst_range s)" "x  $\in$  subst_domain s"
  obtains f T where "s x = Fun f T" "Fun f T  $\in$  subst_range s" "fv (Fun f T) = {}"
proof -
  from assms(2) obtain t where t: "s x = t" "t  $\in$  subst_range s" by moura
  hence "fv t = {}" using assms(1) by auto
  thus ?thesis using t that by (cases t) simp_all
qed

lemma ground_term_subst_domain_fv_subset:
  "fv (t  $\cdot$   $\delta$ ) = {}  $\implies$  fv t  $\subseteq$  subst_domain  $\delta$ "
by (induct t) auto

lemma ground_subst_range_empty_fv:
  "ground (subst_range  $\vartheta$ )  $\implies$  x  $\in$  subst_domain  $\vartheta \implies$  fv ( $\vartheta$  x) = {}"
by simp

lemma subst_Var_notin_img: "x  $\notin$  range_vars s  $\implies$  t  $\cdot$  s = Var x  $\implies$  t = Var x"
using subst_not_img_fixed[of x t s] by (induct t) auto

lemma fv_in_subst_img: "[s v = t; t  $\neq$  Var v]  $\implies$  fv t  $\subseteq$  range_vars s"
unfolding range_vars_alt_def by auto

lemma empty_dom_iff_empty_subst: "subst_domain  $\vartheta$  = {}  $\longleftrightarrow$   $\vartheta$  = Var" by auto

lemma subst_dom_cong: "( $\bigwedge$  v t.  $\vartheta$  v = t  $\implies$   $\delta$  v = t)  $\implies$  subst_domain  $\vartheta \subseteq$  subst_domain  $\delta$ "
by (auto simp add: subst_domain_def)

lemma subst_img_cong: "( $\bigwedge$  v t.  $\vartheta$  v = t  $\implies$   $\delta$  v = t)  $\implies$  range_vars  $\vartheta \subseteq$  range_vars  $\delta$ "
unfolding range_vars_alt_def by (auto simp add: subst_domain_def)

lemma subst_dom_elim: "subst_domain s  $\cap$  range_vars s = {}  $\implies$  fv (t  $\cdot$  s)  $\cap$  subst_domain s = {}"
proof (induction t)
  case (Var v) thus ?case
    using fv_in_subst_img[of s]
    by (cases "s v = Var v") (auto simp add: subst_domain_def)
next
  case Fun thus ?case by auto
qed

lemma subst_dom_insert_finite: "finite (subst_domain s) = finite (subst_domain (s(v := t)))"
proof
  assume "finite (subst_domain s)"
  have "subst_domain (s(v := t))  $\subseteq$  insert v (subst_domain s)" by (auto simp add: subst_domain_def)
  thus "finite (subst_domain (s(v := t)))"
    by (meson (finite (subst_domain s)) finite_insert rev_finite_subset)
next
  assume *: "finite (subst_domain (s(v := t)))"
  hence "finite (insert v (subst_domain s))"
  proof (cases "t = Var v")
    case True
    hence "finite (subst_domain s - {v})" by (metis * subst_dom_update3)
    thus ?thesis by simp
  qed (metis * subst_dom_update2[of t v s])
  thus "finite (subst_domain s)" by simp
qed

lemma trm_subst_disj: "t  $\cdot$   $\vartheta$  = t  $\implies$  fv t  $\cap$  subst_domain  $\vartheta$  = {}"
proof (induction t)
  case (Fun f X)

```



```

hence "map (λx. x · ∅) X = X" by simp
hence "λx. x ∈ set X ⇒ x · ∅ = x" using map_eq_conv by fastforce
thus ?case using Fun.IH by auto
qed (simp add: subst_domain_def)

lemma trm_subst_ident[intro]: "fv t ∩ subst_domain ∅ = {} ⇒ t · ∅ = t"
proof -
  assume "fv t ∩ subst_domain ∅ = {}"
  hence "∀v ∈ fv t. ∀w ∈ subst_domain ∅. v ≠ w" by auto
  thus ?thesis
    by (metis subst_agreement subst_apply_term.simps(1) subst_apply_term_empty subst_domI)
qed

lemma trm_subst_ident'[intro]: "v ∉ subst_domain ∅ ⇒ (Var v) · ∅ = Var v"
using trm_subst_ident by (simp add: subst_domain_def)

lemma trm_subst_ident''[intro]: "(λx. x ∈ fv t ⇒ ∅ x = Var x) ⇒ t · ∅ = t"
proof -
  assume "λx. x ∈ fv t ⇒ ∅ x = Var x"
  hence "fv t ∩ subst_domain ∅ = {}" by (auto simp add: subst_domain_def)
  thus ?thesis using trm_subst_ident by auto
qed

lemma set_subst_ident: "fv_set M ∩ subst_domain ∅ = {} ⇒ M ·_set ∅ = M"
proof -
  assume "fv_set M ∩ subst_domain ∅ = {}"
  hence "∀t ∈ M. t · ∅ = t" by auto
  thus ?thesis by force
qed

lemma trm_subst_ident_subterms[intro]:
  "fv t ∩ subst_domain ∅ = {} ⇒ subterms t ·_set ∅ = subterms t"
using set_subst_ident[of "subterms t" ∅] fv_subterms[of t] by simp

lemma trm_subst_ident_subterms'[intro]:
  "v ∉ fv t ⇒ subterms t ·_set Var(v := s) = subterms t"
using trm_subst_ident_subterms[of t "Var(v := s)"]
by (meson subst_no_occs trm_subst_disj vars_iff_subtermeq)

lemma const_mem_subst_cases:
  assumes "Fun c [] ∈ M ·_set ∅"
  shows "Fun c [] ∈ M ∨ Fun c [] ∈ ∅ ' fv_set M"
proof -
  obtain m where m: "m ∈ M" "m · ∅ = Fun c []" using assms by auto
  thus ?thesis by (cases m) force+
qed

lemma const_mem_subst_cases':
  assumes "Fun c [] ∈ M ·_set ∅"
  shows "Fun c [] ∈ M ∨ Fun c [] ∈ subst_range ∅"
using const_mem_subst_cases[OF assms] by force

lemma fv_subterms_substI[intro]: "y ∈ fv t ⇒ ∅ y ∈ subterms t ·_set ∅"
using image_iff vars_iff_subtermeq by fastforce

lemma fv_subterms_subst_eq[simp]: "fv_set (subterms (t · ∅)) = fv_set (subterms t ·_set ∅)"
using fv_subterms by (induct t) force+

lemma fv_subterms_set_subst: "fv_set (subterms_set M ·_set ∅) = fv_set (subterms_set (M ·_set ∅))"
using fv_subterms_subst_eq[of _ ∅] by auto

lemma fv_subterms_set_subst': "fv_set (subterms_set M ·_set ∅) = fv_set (M ·_set ∅)"
using fv_subterms_set[of "M ·_set ∅"] fv_subterms_set_subst[of ∅ M] by simp

```

```

lemma fv_subst_subset: "x ∈ fv t ⇒ fv (∅ x) ⊆ fv (t · ∅)"
by (metis fv_subset image_eqI subst_apply_fv_unfold)

lemma fv_subst_subset': "fv s ⊆ fv t ⇒ fv (s · ∅) ⊆ fv (t · ∅)"
using fv_subst_subset by (induct s) force+

lemma fv_subst_obtain_var:
  fixes δ::"('a,'b) subst"
  assumes "x ∈ fv (t · δ)"
  shows "∃y ∈ fv t. x ∈ fv (δ y)"
using assms by (induct t) force+

lemma set_subst_all_ident: "fvset (M ·set ∅) ∩ subst_domain δ = {} ⇒ M ·set (∅ ∘s δ) = M ·set ∅"
by (metis set_subst_ident subst_comp_all)

lemma subterms_subst:
  "subterms (t · d) = (subterms t ·set d) ∪ subtermsset (d ' (fv t ∩ subst_domain d))"
by (induct t) (auto simp add: subst_domain_def)

lemma subterms_subst':
  fixes ∅::"('a,'b) subst"
  assumes "∀x ∈ fv t. (∃f. ∅ x = Fun f []) ∨ (∃y. ∅ x = Var y)"
  shows "subterms (t · ∅) = subterms t ·set ∅"
using assms
proof (induction t)
  case (Var x) thus ?case
  proof (cases "x ∈ subst_domain ∅")
    case True
    hence "(∃f. ∅ x = Fun f []) ∨ (∃y. ∅ x = Var y)" using Var by simp
    hence "subterms (∅ x) = {∅ x}" by auto
    thus ?thesis by simp
  qed auto
qed auto

lemma subterms_subst'':
  fixes ∅::"('a,'b) subst"
  assumes "∀x ∈ fvset M. (∃f. ∅ x = Fun f []) ∨ (∃y. ∅ x = Var y)"
  shows "subtermsset (M ·set ∅) = subtermsset M ·set ∅"
using subterms_subst'[of _ ∅] assms by auto

lemma subterms_subst_subterm:
  fixes ∅::"('a,'b) subst"
  assumes "∀x ∈ fv a. (∃f. ∅ x = Fun f []) ∨ (∃y. ∅ x = Var y)"
  and "b ∈ subterms (a · ∅)"
  shows "∃c ∈ subterms a. c · ∅ = b"
using subterms_subst'[OF assms(1)] assms(2) by auto

lemma subterms_subst_subset: "subterms t ·set σ ⊆ subterms (t · σ)"
by (induct t) auto

lemma subterms_subst_subset': "subtermsset M ·set σ ⊆ subtermsset (M ·set σ)"
using subterms_subst_subset by fast

lemma subtermsset_subst:
  fixes ∅::"('a,'b) subst"
  assumes "t ∈ subtermsset (M ·set ∅)"
  shows "t ∈ subtermsset M ·set ∅ ∨ (∃x ∈ fvset M. t ∈ subterms (∅ x))"
using assms subterms_subst[of _ ∅] by auto

lemma rm_vars_dom: "subst_domain (rm_vars V s) = subst_domain s - V"
by (auto simp add: subst_domain_def)

```

```

lemma rm_vars_dom_subset: "subst_domain (rm_vars V s)  $\subseteq$  subst_domain s"
by (auto simp add: subst_domain_def)

lemma rm_vars_dom_eq':
  "subst_domain (rm_vars (UNIV - V) s) = subst_domain s  $\cap$  V"
using rm_vars_dom[of "UNIV - V" s] by blast

lemma rm_vars_img: "subst_range (rm_vars V s) = s ` subst_domain (rm_vars V s)"
by (auto simp add: subst_domain_def)

lemma rm_vars_img_subset: "subst_range (rm_vars V s)  $\subseteq$  subst_range s"
by (auto simp add: subst_domain_def)

lemma rm_vars_img_fv_subset: "range_vars (rm_vars V s)  $\subseteq$  range_vars s"
unfolding range_vars_alt_def by (auto simp add: subst_domain_def)

lemma rm_vars_fv_obtain:
  assumes "x  $\in$  fv (t  $\cdot$  rm_vars X  $\vartheta$ ) - X"
  shows " $\exists y \in$  fv t - X. x  $\in$  fv (rm_vars X  $\vartheta$  y)"
using assms by (induct t) (fastforce, force)

lemma rm_vars_apply: "v  $\in$  subst_domain (rm_vars V s)  $\implies$  (rm_vars V s) v = s v"
by (auto simp add: subst_domain_def)

lemma rm_vars_apply': "subst_domain  $\delta \cap$  vs = {}  $\implies$  rm_vars vs  $\delta$  =  $\delta$ "
by force

lemma rm_vars_ident: "fv t  $\cap$  vs = {}  $\implies$  t  $\cdot$  (rm_vars vs  $\vartheta$ ) = t  $\cdot$   $\vartheta$ "
by (induct t) auto

lemma rm_vars_fv_subset: "fv (t  $\cdot$  rm_vars X  $\vartheta$ )  $\subseteq$  fv t  $\cup$  fv (t  $\cdot$   $\vartheta$ )"
by (induct t) auto

lemma rm_vars_fv_disj:
  assumes "fv t  $\cap$  X = {}" "fv (t  $\cdot$   $\vartheta$ )  $\cap$  X = {}"
  shows "fv (t  $\cdot$  rm_vars X  $\vartheta$ )  $\cap$  X = {}"
using rm_vars_ident[OF assms(1)] assms(2) by auto

lemma rm_vars_ground_supports:
  assumes "ground (subst_range  $\vartheta$ )"
  shows "rm_vars X  $\vartheta$  supports  $\vartheta$ "
proof
  fix x
  have *: "ground (subst_range (rm_vars X  $\vartheta$ ))"
  using rm_vars_img_subset[of X  $\vartheta$ ] assms
  by (auto simp add: subst_domain_def)
  show "rm_vars X  $\vartheta$  x  $\cdot$   $\vartheta$  =  $\vartheta$  x"
  proof (cases "x  $\in$  subst_domain (rm_vars X  $\vartheta$ )")
    case True
    hence "fv (rm_vars X  $\vartheta$  x) = {}" using * by auto
    thus ?thesis using True by auto
  qed (simp add: subst_domain_def)
qed

lemma rm_vars_split:
  assumes "ground (subst_range  $\vartheta$ )"
  shows " $\vartheta$  = rm_vars X  $\vartheta$   $\circ_s$  rm_vars (subst_domain  $\vartheta$  - X)  $\vartheta$ "
proof -
  let ?s1 = "rm_vars X  $\vartheta$ "
  let ?s2 = "rm_vars (subst_domain  $\vartheta$  - X)  $\vartheta$ "

  have doms: "subst_domain ?s1  $\subseteq$  subst_domain  $\vartheta$ " "subst_domain ?s2  $\subseteq$  subst_domain  $\vartheta$ "
  by (auto simp add: subst_domain_def)

```

```

{ fix x assume "x ∉ subst_domain ∅"
  hence "∅ x = Var x" "?s1 x = Var x" "?s2 x = Var x" using doms by auto
  hence "∅ x = (?s1 ∘s ?s2) x" by (simp add: subst_compose_def)
} moreover {
  fix x assume "x ∈ subst_domain ∅" "x ∈ X"
  hence "?s1 x = Var x" "?s2 x = ∅ x" using doms by auto
  hence "∅ x = (?s1 ∘s ?s2) x" by (simp add: subst_compose_def)
} moreover {
  fix x assume "x ∈ subst_domain ∅" "x ∉ X"
  hence "?s1 x = ∅ x" "fv (∅ x) = {}" using assms doms by auto
  hence "∅ x = (?s1 ∘s ?s2) x" by (simp add: subst_compose subst_ground_ident)
} ultimately show ?thesis by blast
qed

lemma rm_vars_fv_img_disj:
  assumes "fv t ∩ X = {}" "X ∩ range_vars ∅ = {}"
  shows "fv (t · rm_vars X ∅) ∩ X = {}"
using assms
proof (induction t)
  case (Var x)
  hence *: "(rm_vars X ∅) x = ∅ x" by auto
  show ?case
  proof (cases "x ∈ subst_domain ∅")
    case True
    hence "∅ x ∈ subst_range ∅" by auto
    hence "fv (∅ x) ∩ X = {}" using Var.prem1(2) unfolding range_vars_alt_def by fastforce
    thus ?thesis using * by auto
  next
    case False thus ?thesis using Var.prem1(1) by auto
  qed
next
  case Fun thus ?case by auto
qed

lemma subst_apply_dom_ident: "t · ∅ = t ⟹ subst_domain δ ⊆ subst_domain ∅ ⟹ t · δ = t"
proof (induction t)
  case (Fun f T) thus ?case by (induct T) auto
qed (auto simp add: subst_domain_def)

lemma rm_vars_subst_apply_ident:
  assumes "t · ∅ = t"
  shows "t · (rm_vars vs ∅) = t"
using rm_vars_dom[of vs ∅] subst_apply_dom_ident[OF assms, of "rm_vars vs ∅"] by auto

lemma rm_vars_subst_eq:
  "t · δ = t · rm_vars (subst_domain δ - subst_domain δ ∩ fv t) δ"
by (auto intro: term_subst_eq)

lemma rm_vars_subst_eq':
  "t · δ = t · rm_vars (UNIV - fv t) δ"
by (auto intro: term_subst_eq)

lemma rm_vars_comp:
  assumes "range_vars δ ∩ vs = {}"
  shows "t · rm_vars vs (δ ∘s ∅) = t · (rm_vars vs δ ∘s rm_vars vs ∅)"
using assms
proof (induction t)
  case (Var x) thus ?case
  proof (cases "x ∈ vs")
    case True thus ?thesis using Var by auto
  next
    case False

```

```

have "subst_domain (rm_vars vs  $\vartheta$ )  $\cap$  vs = {}" by (auto simp add: subst_domain_def)
moreover have "fv ( $\delta$  x)  $\cap$  vs = {}"
  using Var False unfolding range_vars_alt_def by force
ultimately have " $\delta$  x  $\cdot$  (rm_vars vs  $\vartheta$ ) =  $\delta$  x  $\cdot$   $\vartheta$ "
  using rm_vars_ident by (simp add: subst_domain_def)
moreover have "(rm_vars vs ( $\delta \circ_s \vartheta$ )) x = ( $\delta \circ_s \vartheta$ ) x" by (metis False)
ultimately show ?thesis using subst_compose by auto
qed
next
case Fun thus ?case by auto
qed

lemma rm_vars_fv_set_subst:
  assumes "x  $\in$  fv_set (rm_vars X  $\vartheta$  ' Y)"
  shows "x  $\in$  fv_set ( $\vartheta$  ' Y)  $\vee$  x  $\in$  X"
using assms by auto

lemma disj_dom_img_var_notin:
  assumes "subst_domain  $\vartheta \cap$  range_vars  $\vartheta$  = {}" " $\vartheta$  v = t" "t  $\neq$  Var v"
  shows "v  $\notin$  fv t" " $\forall v \in$  fv (t  $\cdot$   $\vartheta$ ). v  $\notin$  subst_domain  $\vartheta$ "
proof -
  have "v  $\in$  subst_domain  $\vartheta$ " "fv t  $\subseteq$  range_vars  $\vartheta$ "
    using fv_in_subst_img[of  $\vartheta$  v t, OF assms(2)] assms(2,3)
    by (auto simp add: subst_domain_def)
  thus "v  $\notin$  fv t" using assms(1) by auto

  have *: "fv t  $\cap$  subst_domain  $\vartheta$  = {}"
    using assms(1) <fv t  $\subseteq$  range_vars  $\vartheta$ >
    by auto
  hence "t  $\cdot$   $\vartheta$  = t" by blast
  thus " $\forall v \in$  fv (t  $\cdot$   $\vartheta$ ). v  $\notin$  subst_domain  $\vartheta$ " using * by auto
qed

lemma subst_sends_dom_to_img: "v  $\in$  subst_domain  $\vartheta \implies$  fv (Var v  $\cdot$   $\vartheta$ )  $\subseteq$  range_vars  $\vartheta$ "
unfolding range_vars_alt_def by auto

lemma subst_sends_fv_to_img: "fv (t  $\cdot$  s)  $\subseteq$  fv t  $\cup$  range_vars s"
proof (induction t)
  case (Var v) thus ?case
  proof (cases "Var v  $\cdot$  s = Var v")
    case True thus ?thesis by simp
  next
    case False
    hence "v  $\in$  subst_domain s" by (meson trm_subst_ident')
    hence "fv (Var v  $\cdot$  s)  $\subseteq$  range_vars s"
      using subst_sends_dom_to_img by simp
    thus ?thesis by auto
  qed
next
case Fun thus ?case by auto
qed

lemma ident_comp_subst_trm_if_disj:
  assumes "subst_domain  $\sigma \cap$  range_vars  $\vartheta$  = {}" "v  $\in$  subst_domain  $\vartheta$ "
  shows "( $\vartheta \circ_s \sigma$ ) v =  $\vartheta$  v"
proof -
  from assms have "subst_domain  $\sigma \cap$  fv ( $\vartheta$  v) = {}"
    using fv_in_subst_img unfolding range_vars_alt_def by auto
  thus "( $\vartheta \circ_s \sigma$ ) v =  $\vartheta$  v" unfolding subst_compose_def by blast
qed

lemma ident_comp_subst_trm_if_disj': "fv ( $\vartheta$  v)  $\cap$  subst_domain  $\sigma$  = {}  $\implies$  ( $\vartheta \circ_s \sigma$ ) v =  $\vartheta$  v"
unfolding subst_compose_def by blast

```

```

lemma subst_idemI[intro]: "subst_domain  $\sigma \cap \text{range\_vars } \sigma = \{\}$   $\implies \text{subst\_idem } \sigma$ "
using ident_comp_subst_trm_if_disj[of  $\sigma$   $\sigma$ ]
  var_not_in_subst_dom[of  $\_$   $\sigma$ ]
  subst_eq_if_eq_vars[of  $\sigma$ ]
by (metis subst_idem_def subst_compose_def var_comp(2))

lemma subst_idemI'[intro]: "ground (subst_range  $\sigma$ )  $\implies \text{subst\_idem } \sigma$ "
proof (intro subst_idemI)
  assume "ground (subst_range  $\sigma$ )"
  hence "range_vars  $\sigma = \{\}$ " by (metis ground_range_vars)
  thus "subst_domain  $\sigma \cap \text{range\_vars } \sigma = \{\}$ " by blast
qed

lemma subst_idemE: "subst_idem  $\sigma \implies \text{subst\_domain } \sigma \cap \text{range\_vars } \sigma = \{\}$ "
proof -
  assume "subst_idem  $\sigma$ "
  hence " $\bigwedge v. \text{fv } (\sigma v) \cap \text{subst\_domain } \sigma = \{\}$ "
  unfolding subst_idem_def subst_compose_def by (metis trm_subst_disj)
  thus ?thesis
  unfolding range_vars_alt_def by auto
qed

lemma subst_idem_rm_vars: "subst_idem  $\vartheta \implies \text{subst\_idem } (\text{rm\_vars } X \vartheta)$ "
proof -
  assume "subst_idem  $\vartheta$ "
  hence "subst_domain  $\vartheta \cap \text{range\_vars } \vartheta = \{\}$ " by (metis subst_idemE)
  moreover have
    "subst_domain (rm_vars X  $\vartheta$ )  $\subseteq \text{subst\_domain } \vartheta$ "
    "range_vars (rm_vars X  $\vartheta$ )  $\subseteq \text{range\_vars } \vartheta$ "
  unfolding range_vars_alt_def by (auto simp add: subst_domain_def)
  ultimately show ?thesis by blast
qed

lemma subst_fv_bounded_if_img_bounded: "range_vars  $\vartheta \subseteq \text{fv } t \cup V \implies \text{fv } (t \cdot \vartheta) \subseteq \text{fv } t \cup V$ "
proof (induction t)
  case (Var v) thus ?case unfolding range_vars_alt_def by (cases " $\vartheta v = \text{Var } v$ ") auto
qed (metis (no_types, lifting) Un_assoc Un_commute subst_sends_fv_to_img sup.absorb_iff2)

lemma subst_fv_bound_singleton: "fv (t · Var(v := t'))  $\subseteq \text{fv } t \cup \text{fv } t'$ "
using subst_fv_bounded_if_img_bounded[of "Var(v := t')" t "fv t'"]
unfolding range_vars_alt_def by (auto simp add: subst_domain_def)

lemma subst_fv_bounded_if_img_bounded':
  assumes "range_vars  $\vartheta \subseteq \text{fv}_{\text{set}} M$ "
  shows "fvset (M ·set  $\vartheta$ )  $\subseteq \text{fv}_{\text{set}} M$ "
proof
  fix v assume *: "v  $\in \text{fv}_{\text{set}} (M \cdot_{\text{set}} \vartheta)$ "

  obtain t where t: "t  $\in M$ " "t ·  $\vartheta \in M \cdot_{\text{set}} \vartheta$ " "v  $\in \text{fv } (t \cdot \vartheta)$ "
  proof -
    assume **: " $\bigwedge t. [\![t \in M; t \cdot \vartheta \in M \cdot_{\text{set}} \vartheta; v \in \text{fv } (t \cdot \vartheta)]\!] \implies \text{thesis}$ "
    have "v  $\in \bigcup (\text{fv } ' ((\lambda t. t \cdot \vartheta) ' M))$ " using * by (metis fvset.sims)
    hence " $\exists t. t \in M \wedge v \in \text{fv } (t \cdot \vartheta)$ " by blast
    thus ?thesis using ** imageI by blast
  qed

  from (t  $\in M$ ) obtain M' where "t  $\notin M'$ " "M = insert t M'" by (meson Set.set_insert)
  hence "fvset M = fv t  $\cup \text{fv}_{\text{set}} M'$ " by simp
  hence "fv (t ·  $\vartheta$ )  $\subseteq \text{fv}_{\text{set}} M$ " using subst_fv_bounded_if_img_bounded assms by simp
  thus "v  $\in \text{fv}_{\text{set}} M$ " using assms (v  $\in \text{fv } (t \cdot \vartheta)$ ) by auto
qed

```

```

lemma ground_img_if_ground_subst: "( $\bigwedge v t. s v = t \implies fv\ t = \{\}$ )  $\implies range\_vars\ s = \{\}$ "
unfolding range_vars_alt_def by auto

lemma ground_subst_fv_subset: "ground (subst_range  $\vartheta$ )  $\implies fv\ (t \cdot \vartheta) \subseteq fv\ t$ "
using subst_fv_bounded_if_img_bounded[of  $\vartheta$ ]
unfolding range_vars_alt_def by force

lemma ground_subst_fv_subset': "ground (subst_range  $\vartheta$ )  $\implies fv_{set}\ (M \cdot_{set}\ \vartheta) \subseteq fv_{set}\ M$ "
using subst_fv_bounded_if_img_bounded'[of  $\vartheta\ M$ ]
unfolding range_vars_alt_def by auto

lemma subst_to_var_is_var[elim]: " $t \cdot s = Var\ v \implies \exists w. t = Var\ w$ "
using subst_apply_term.elims by blast

lemma subst_dom_comp_inI:
  assumes "y  $\notin$  subst_domain  $\sigma$ "
  and "y  $\in$  subst_domain  $\delta$ "
  shows "y  $\in$  subst_domain ( $\sigma \circ_s \delta$ )"
using assms subst_domain_subst_compose[of  $\sigma\ \delta$ ] by blast

lemma subst_comp_notin_dom_eq:
  " $x \notin$  subst_domain  $\vartheta1 \implies (\vartheta1 \circ_s \vartheta2)\ x = \vartheta2\ x$ "
unfolding subst_compose_def by fastforce

lemma subst_dom_comp_eq:
  assumes "subst_domain  $\vartheta \cap range\_vars\ \sigma = \{\}$ "
  shows "subst_domain ( $\vartheta \circ_s \sigma$ ) = subst_domain  $\vartheta \cup$  subst_domain  $\sigma$ "
proof (rule ccontr)
  assume "subst_domain ( $\vartheta \circ_s \sigma$ )  $\neq$  subst_domain  $\vartheta \cup$  subst_domain  $\sigma$ "
  hence "subst_domain ( $\vartheta \circ_s \sigma$ )  $\subset$  subst_domain  $\vartheta \cup$  subst_domain  $\sigma$ "
  using subst_domain_compose[of  $\vartheta\ \sigma$ ] by (simp add: subst_domain_def)
  then obtain v where "v  $\notin$  subst_domain ( $\vartheta \circ_s \sigma$ )" "v  $\in$  subst_domain  $\vartheta \cup$  subst_domain  $\sigma$ " by auto
  hence v_in_some_subst: " $\vartheta\ v \neq Var\ v \vee \sigma\ v \neq Var\ v$ " and " $\vartheta\ v \cdot \sigma = Var\ v$ "
  unfolding subst_compose_def by (auto simp add: subst_domain_def)
  then obtain w where " $\vartheta\ v = Var\ w$ " using subst_to_var_is_var by fastforce
  show False
  proof (cases "v = w")
    case True
    hence " $\vartheta\ v = Var\ v$ " using " $\vartheta\ v = Var\ w$ " by simp
    hence " $\sigma\ v \neq Var\ v$ " using v_in_some_subst by simp
    thus False using " $\vartheta\ v = Var\ v$ " " $\vartheta\ v \cdot \sigma = Var\ v$ " by simp
  next
    case False
    hence "v  $\in$  subst_domain  $\vartheta$ " using v_in_some_subst " $\vartheta\ v \cdot \sigma = Var\ v$ " by auto
    hence "v  $\notin$  range_vars  $\sigma$ " using assms by auto
    moreover have " $\sigma\ w = Var\ v$ " using " $\vartheta\ v \cdot \sigma = Var\ v$ " " $\vartheta\ v = Var\ w$ " by simp
    hence "v  $\in$  range_vars  $\sigma$ " using "v  $\neq w$ " subst_fv_imgI[of  $\sigma\ w$ ] by simp
    ultimately show False ..
  qed
qed
qed

lemma subst_img_comp_subset[simp]:
  "range_vars ( $\vartheta1 \circ_s \vartheta2$ )  $\subseteq$  range_vars  $\vartheta1 \cup$  range_vars  $\vartheta2$ "
proof
  let ?img = "range_vars"
  fix x assume "x  $\in$  ?img ( $\vartheta1 \circ_s \vartheta2$ )"
  then obtain v t where vt: "x  $\in$  fv t" "t = ( $\vartheta1 \circ_s \vartheta2$ ) v" "t  $\neq$  Var v"
  unfolding range_vars_alt_def subst_compose_def by (auto simp add: subst_domain_def)

  { assume "x  $\notin$  ?img  $\vartheta1$ " hence "x  $\in$  ?img  $\vartheta2$ "
    by (metis (no_types, hide_lams) fv_in_subst_img Un_iff subst_compose_def
      vt subsetCE subst_apply_term.simps(1) subst_sends_fv_to_img)
  }

```

thus "x ∈ ?img  $\vartheta_1 \cup \text{?img } \vartheta_2$ " by auto  
qed

lemma subst\_img\_comp\_subset':  
 assumes "t ∈ subst\_range ( $\vartheta_1 \circ_s \vartheta_2$ )"  
 shows "t ∈ subst\_range  $\vartheta_2 \vee (\exists t' \in \text{subst\_range } \vartheta_1. t = t' \cdot \vartheta_2)$ "  
proof -  
 obtain x where x: "x ∈ subst\_domain ( $\vartheta_1 \circ_s \vartheta_2$ )" "( $\vartheta_1 \circ_s \vartheta_2$ ) x = t" "t ≠ Var x"  
 using assms by (auto simp add: subst\_domain\_def)  
 { assume "x ∉ subst\_domain  $\vartheta_1$ "  
 hence "( $\vartheta_1 \circ_s \vartheta_2$ ) x =  $\vartheta_2$  x" unfolding subst\_compose\_def by auto  
 hence ?thesis using x by auto  
 } moreover {  
 assume "x ∈ subst\_domain  $\vartheta_1$ " hence ?thesis using subst\_compose x(2) by fastforce  
 } ultimately show ?thesis by metis  
qed

lemma subst\_img\_comp\_subset'':  
 "subterms<sub>set</sub> (subst\_range ( $\vartheta_1 \circ_s \vartheta_2$ )) ⊆  
 subterms<sub>set</sub> (subst\_range  $\vartheta_2$ ) ∪ ((subterms<sub>set</sub> (subst\_range  $\vartheta_1$ )) ·<sub>set</sub>  $\vartheta_2$ )"  
proof  
 fix t assume "t ∈ subterms<sub>set</sub> (subst\_range ( $\vartheta_1 \circ_s \vartheta_2$ ))"  
 then obtain x where x: "x ∈ subst\_domain ( $\vartheta_1 \circ_s \vartheta_2$ )" "t ∈ subterms (( $\vartheta_1 \circ_s \vartheta_2$ ) x)"  
 by auto  
 show "t ∈ subterms<sub>set</sub> (subst\_range  $\vartheta_2$ ) ∪ (subterms<sub>set</sub> (subst\_range  $\vartheta_1$ ) ·<sub>set</sub>  $\vartheta_2$ )"  
proof (cases "x ∈ subst\_domain  $\vartheta_1$ ")  
 case True thus ?thesis  
 using subst\_compose[of  $\vartheta_1 \vartheta_2$ ] x(2) subterms\_subst  
 by fastforce  
 next  
 case False  
 hence "( $\vartheta_1 \circ_s \vartheta_2$ ) x =  $\vartheta_2$  x" unfolding subst\_compose\_def by auto  
 thus ?thesis using x by (auto simp add: subst\_domain\_def)  
qed  
qed

lemma subst\_img\_comp\_subset''':  
 "subterms<sub>set</sub> (subst\_range ( $\vartheta_1 \circ_s \vartheta_2$ )) - range Var ⊆  
 subterms<sub>set</sub> (subst\_range  $\vartheta_2$ ) - range Var ∪ ((subterms<sub>set</sub> (subst\_range  $\vartheta_1$ ) - range Var) ·<sub>set</sub>  $\vartheta_2$ )"  
proof  
 fix t assume t: "t ∈ subterms<sub>set</sub> (subst\_range ( $\vartheta_1 \circ_s \vartheta_2$ )) - range Var"  
 then obtain f T where fT: "t = Fun f T" by (cases t) simp\_all  
 then obtain x where x: "x ∈ subst\_domain ( $\vartheta_1 \circ_s \vartheta_2$ )" "Fun f T ∈ subterms (( $\vartheta_1 \circ_s \vartheta_2$ ) x)"  
 using t by auto  
 have "Fun f T ∈ subterms<sub>set</sub> (subst\_range  $\vartheta_2$ ) ∪ (subterms<sub>set</sub> (subst\_range  $\vartheta_1$ ) - range Var ·<sub>set</sub>  $\vartheta_2$ )"  
proof (cases "x ∈ subst\_domain  $\vartheta_1$ ")  
 case True  
 hence "Fun f T ∈ (subterms<sub>set</sub> (subst\_range  $\vartheta_2$ )) ∪ (subterms ( $\vartheta_1$  x) ·<sub>set</sub>  $\vartheta_2$ )"  
 using x(2) subterms\_subst[of " $\vartheta_1$  x"  $\vartheta_2$ ]  
 unfolding subst\_compose[of  $\vartheta_1 \vartheta_2$  x] by auto  
 moreover have ?thesis when \*: "Fun f T ∈ subterms ( $\vartheta_1$  x) ·<sub>set</sub>  $\vartheta_2$ "  
proof -  
 obtain s where s: "s ∈ subterms ( $\vartheta_1$  x)" "Fun f T = s ·  $\vartheta_2$ " using \* by mouna  
 show ?thesis  
 proof (cases s)  
 case (Var y)  
 hence "Fun f T ∈ subst\_range  $\vartheta_2$ " using s by force  
 thus ?thesis by blast  
 next  
 case (Fun g S)  
 hence "Fun f T ∈ (subterms ( $\vartheta_1$  x) - range Var) ·<sub>set</sub>  $\vartheta_2$ " using s by blast  
 thus ?thesis using True by auto  
 qed  
qed



```

qed
ultimately show ?thesis by blast
next
case False
hence " $(\vartheta_1 \circ_s \vartheta_2) x = \vartheta_2 x$ " unfolding subst_compose_def by auto
thus ?thesis using x by (auto simp add: subst_domain_def)
qed
thus "t  $\in$  subtermsset (subst_range  $\vartheta_2$ ) - range Var  $\cup$ 
      (subtermsset (subst_range  $\vartheta_1$ ) - range Var  $\cdot_{set} \vartheta_2$ )"
  using fT by auto
qed

lemma subst_img_comp_subset_const:
  assumes "Fun c []  $\in$  subst_range ( $\vartheta_1 \circ_s \vartheta_2$ )"
  shows "Fun c []  $\in$  subst_range  $\vartheta_2 \vee$  Fun c []  $\in$  subst_range  $\vartheta_1 \vee$ 
        ( $\exists x. \text{Var } x \in \text{subst\_range } \vartheta_1 \wedge \vartheta_2 x = \text{Fun } c []$ )"
proof (cases "Fun c []  $\in$  subst_range  $\vartheta_2$ ")
case False
then obtain t where t: "t  $\in$  subst_range  $\vartheta_1$ " "Fun c [] = t  $\cdot$   $\vartheta_2$ "
  using subst_img_comp_subset'[OF assms] by auto
thus ?thesis by (cases t) auto
qed (simp add: subst_img_comp_subset'[OF assms])

lemma subst_img_comp_subset_const':
  fixes  $\delta \tau :: ('f, 'v) \text{subst}$ 
  assumes " $(\delta \circ_s \tau) x = \text{Fun } c []$ "
  shows " $\delta x = \text{Fun } c [] \vee (\exists z. \delta x = \text{Var } z \wedge \tau z = \text{Fun } c [])$ "
proof (cases " $\delta x = \text{Fun } c []$ ")
case False
then obtain t where " $\delta x = t$ " " $t \cdot \tau = \text{Fun } c []$ " using assms unfolding subst_compose_def by auto
thus ?thesis by (cases t) auto
qed simp

lemma subst_img_comp_subset_ground:
  assumes "ground (subst_range  $\vartheta_1$ )"
  shows "subst_range ( $\vartheta_1 \circ_s \vartheta_2$ )  $\subseteq$  subst_range  $\vartheta_1 \cup$  subst_range  $\vartheta_2$ "
proof
fix t assume t: "t  $\in$  subst_range ( $\vartheta_1 \circ_s \vartheta_2$ )"
then obtain x where x: "x  $\in$  subst_domain ( $\vartheta_1 \circ_s \vartheta_2$ )" "t = ( $\vartheta_1 \circ_s \vartheta_2$ ) x" by auto

show "t  $\in$  subst_range  $\vartheta_1 \cup$  subst_range  $\vartheta_2$ "
proof (cases "x  $\in$  subst_domain  $\vartheta_1$ ")
case True
hence "fv ( $\vartheta_1 x$ ) = {}" using assms ground_subst_range_empty_fv by fast
hence "t =  $\vartheta_1 x$ " using x(2) unfolding subst_compose_def by blast
thus ?thesis using True by simp
next
case False
hence "t =  $\vartheta_2 x$ " "x  $\in$  subst_domain  $\vartheta_2$ "
  using x subst_domain_compose[of  $\vartheta_1 \vartheta_2$ ]
  by (metis subst_comp_notin_dom_eq, blast)
thus ?thesis using x by simp
qed
qed

lemma subst_fv_dom_img_single:
  assumes "v  $\notin$  fv t" " $\sigma v = t$ " " $\bigwedge w. v \neq w \implies \sigma w = \text{Var } w$ "
  shows "subst_domain  $\sigma = \{v\}$ " "range_vars  $\sigma = \text{fv } t$ "
proof -
show "subst_domain  $\sigma = \{v\}$ " using assms by (fastforce simp add: subst_domain_def)
have "fv t  $\subseteq$  range_vars  $\sigma$ " by (metis fv_in_subst_img assms(1,2) vars_iff_subterm_or_eq)
moreover have " $\bigwedge v. \sigma v \neq \text{Var } v \implies \sigma v = t$ " using assms by fastforce
ultimately show "range_vars  $\sigma = \text{fv } t$ "

```

```

unfolding range_vars_alt_def
by (auto simp add: subst_domain_def)
qed

lemma subst_comp_upd1:
  " $\vartheta(v := t) \circ_s \sigma = (\vartheta \circ_s \sigma)(v := t \cdot \sigma)$ "
unfolding subst_compose_def by auto

lemma subst_comp_upd2:
  assumes " $v \notin \text{subst\_domain } s$ " " $v \notin \text{range\_vars } s$ "
  shows " $s(v := t) = s \circ_s (\text{Var}(v := t))$ "
unfolding subst_compose_def
proof -
  { fix w
    have " $s(v := t) \cdot w = s \cdot w \cdot \text{Var}(v := t)$ "
    proof (cases " $w = v$ ")
      case True
      hence " $s \cdot w = \text{Var } w$ " using  $\langle v \notin \text{subst\_domain } s \rangle$  by (simp add: subst_domain_def)
      thus ?thesis using  $\langle w = v \rangle$  by simp
    next
      case False
      hence " $s(v := t) \cdot w = s \cdot w$ " by simp
      moreover have " $s \cdot w \cdot \text{Var}(v := t) = s \cdot w$ " using  $\langle w \neq v \rangle \langle v \notin \text{range\_vars } s \rangle$ 
        by (metis fv_in_subst_img fun_upd_apply insert_absorb insert_subset
          repl_invariance subst_apply_term.simps(1) subst_apply_term_empty)
      ultimately show ?thesis ..
    }
  }
  thus " $s(v := t) = (\lambda w. s \cdot w \cdot \text{Var}(v := t))$ " by auto
qed

lemma ground_subst_dom_iff_img:
  " $\text{ground } (\text{subst\_range } \sigma) \implies x \in \text{subst\_domain } \sigma \longleftrightarrow \sigma x \in \text{subst\_range } \sigma$ "
by (auto simp add: subst_domain_def)

lemma finite_dom_subst_exists:
  " $\text{finite } S \implies \exists \sigma :: ('f, 'v) \text{subst}. \text{subst\_domain } \sigma = S$ "
proof (induction S rule: finite.induct)
  case (insertI A a)
  then obtain  $\sigma :: ('f, 'v) \text{subst}$  where " $\text{subst\_domain } \sigma = A$ " by blast
  fix f :: 'f
  have " $\text{subst\_domain } (\sigma(a := \text{Fun } f [])) = \text{insert } a \ A$ "
    using  $\langle \text{subst\_domain } \sigma = A \rangle$ 
    by (auto simp add: subst_domain_def)
  thus ?case by metis
qed (auto simp add: subst_domain_def)

lemma subst_inj_is_bij_betw_dom_img_if_ground_img:
  assumes " $\text{ground } (\text{subst\_range } \sigma)$ "
  shows " $\text{inj } \sigma \longleftrightarrow \text{bij\_betw } \sigma (\text{subst\_domain } \sigma) (\text{subst\_range } \sigma)$ " (is " $?A \longleftrightarrow ?B$ ")
proof
  show " $?A \implies ?B$ " by (metis bij_betw_def injD inj_onI subst_range.simps)
next
  assume ?B
  hence " $\text{inj\_on } \sigma (\text{subst\_domain } \sigma)$ " unfolding bij_betw_def by auto
  moreover have " $\bigwedge x. x \in \text{UNIV} - \text{subst\_domain } \sigma \implies \sigma x = \text{Var } x$ " by auto
  hence " $\text{inj\_on } \sigma (\text{UNIV} - \text{subst\_domain } \sigma)$ "
    using inj_onI[of " $\text{UNIV} - \text{subst\_domain } \sigma$ "]
    by (metis term.inject(1))
  moreover have " $\bigwedge x y. x \in \text{subst\_domain } \sigma \implies y \notin \text{subst\_domain } \sigma \implies \sigma x \neq \sigma y$ "
    using assms by (auto simp add: subst_domain_def)
  ultimately show ?A by (metis injI inj_onD subst_domI term.inject(1))
qed

```

```

lemma bij_finite_ground_subst_exists:
  assumes "finite (S::'v set)" "infinite (U::('f,'v) term set)" "ground U"
  shows "∃σ::('f,'v) subst. subst_domain σ = S
        ∧ bij_betw σ (subst_domain σ) (subst_range σ)
        ∧ subst_range σ ⊆ U"

proof -
  obtain T' where "T' ⊆ U" "card T' = card S" "finite T'"
    by (meson assms(2) finite_Diff2 infinite_arbitrarily_large)
  then obtain f::"'v ⇒ ('f,'v) term" where f_bij: "bij_betw f S T'"
    using finite_same_card_bij[OF assms(1)] by metis
  hence *: "∧v. v ∈ S ⇒ f v ≠ Var v"
    using ⟨ground U⟩ ⟨T' ⊆ U⟩ bij_betwE
    by fastforce

  let ?σ = "λv. if v ∈ S then f v else Var v"
  have "subst_domain ?σ = S"
  proof
    show "subst_domain ?σ ⊆ S" by (auto simp add: subst_domain_def)

    { fix v assume "v ∈ S" "v ∉ subst_domain ?σ"
      hence "f v = Var v" by (simp add: subst_domain_def)
      hence False using *[OF ⟨v ∈ S⟩] by metis
    }
    thus "S ⊆ subst_domain ?σ" by blast
  qed
  hence "∧v w. [v ∈ subst_domain ?σ; w ∉ subst_domain ?σ] ⇒ ?σ w ≠ ?σ v"
    using ⟨ground U⟩ bij_betwE[OF f_bij] set_rev_mp[OF _ ⟨T' ⊆ U⟩]
    by (metis (no_types, lifting) UN_iff empty_iff vars_iff_subterm_or_eq fv_set.simps)
  hence "inj_on ?σ (subst_domain ?σ)"
    using f_bij ⟨subst_domain ?σ = S⟩
    unfolding bij_betw_def inj_on_def
    by metis
  hence "bij_betw ?σ (subst_domain ?σ) (subst_range ?σ)"
    using inj_on_imp_bij_betw[OF ?σ] by simp
  moreover have "subst_range ?σ = T'"
    using ⟨bij_betw f S T'⟩ ⟨subst_domain ?σ = S⟩
    unfolding bij_betw_def by auto
  hence "subst_range ?σ ⊆ U" using ⟨T' ⊆ U⟩ by auto
  ultimately show ?thesis using ⟨subst_domain ?σ = S⟩ by (metis (lifting))
qed

lemma bij_finite_const_subst_exists:
  assumes "finite (S::'v set)" "finite (T::'f set)" "infinite (U::'f set)"
  shows "∃σ::('f,'v) subst. subst_domain σ = S
        ∧ bij_betw σ (subst_domain σ) (subst_range σ)
        ∧ subst_range σ ⊆ (λc. Fun c []) ' (U - T)"

proof -
  obtain T' where "T' ⊆ U - T" "card T' = card S" "finite T'"
    by (meson assms(2,3) finite_Diff2 infinite_arbitrarily_large)
  then obtain f::"'v ⇒ 'f" where f_bij: "bij_betw f S T'"
    using finite_same_card_bij[OF assms(1)] by metis

  let ?σ = "λv. if v ∈ S then Fun (f v) [] else Var v"
  have "subst_domain ?σ = S" by (simp add: subst_domain_def)
  moreover have "∧v w. [v ∈ subst_domain ?σ; w ∉ subst_domain ?σ] ⇒ ?σ w ≠ ?σ v" by auto
  hence "inj_on ?σ (subst_domain ?σ)"
    using f_bij unfolding bij_betw_def inj_on_def
    by (metis ⟨subst_domain ?σ = S⟩ term.inject(2))
  hence "bij_betw ?σ (subst_domain ?σ) (subst_range ?σ)"
    using inj_on_imp_bij_betw[OF ?σ] by simp
  moreover have "subst_range ?σ = ((λc. Fun c []) ' T)"
    using ⟨bij_betw f S T'⟩ unfolding bij_betw_def inj_on_def by (auto simp add: subst_domain_def)

```

hence "subst\_range ? $\sigma \subseteq ((\lambda c. \text{Fun } c []) \text{ ' } (U - T))"$  using  $\langle T' \subseteq U - T \rangle$  by auto  
ultimately show ?thesis by (metis (lifting))  
qed

lemma bij\_finite\_const\_subst\_exists':  
assumes "finite (S::'v set)" "finite (T::('f,'v) terms)" "infinite (U::'f set)"  
shows " $\exists \sigma::('f,'v) \text{ subst. } \text{subst\_domain } \sigma = S$   
           $\wedge \text{bij\_betw } \sigma (\text{subst\_domain } \sigma) (\text{subst\_range } \sigma)$   
           $\wedge \text{subst\_range } \sigma \subseteq ((\lambda c. \text{Fun } c []) \text{ ' } U) - T"$   
proof -  
have "finite ( $\bigcup (\text{funs\_term ' } T)$ )" using assms(2) by auto  
then obtain  $\sigma$  where  $\sigma$ :  
  "subst\_domain  $\sigma = S$ " "bij\_betw  $\sigma (\text{subst\_domain } \sigma) (\text{subst\_range } \sigma)"$   
  "subst\_range  $\sigma \subseteq ((\lambda c. \text{Fun } c []) \text{ ' } (U - (\bigcup (\text{funs\_term ' } T))))"$   
  using bij\_finite\_const\_subst\_exists[OF assms(1) \_ assms(3)] by blast  
moreover have " $(\lambda c. \text{Fun } c []) \text{ ' } (U - (\bigcup (\text{funs\_term ' } T))) \subseteq ((\lambda c. \text{Fun } c []) \text{ ' } U) - T"$  by auto  
ultimately show ?thesis by blast  
qed

lemma bij\_betw\_iterI:  
assumes "bij\_betw f A B" "bij\_betw g C D" "A  $\cap$  C = {}" "B  $\cap$  D = {}"  
shows "bij\_betw ( $\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } g \ x$ ) (A  $\cup$  C) (B  $\cup$  D)"  
proof -  
have "bij\_betw ( $\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } g \ x$ ) A B"  
  by (metis bij\_betw\_cong[of A f " $\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } g \ x$ " B] assms(1))  
moreover have "bij\_betw ( $\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } g \ x$ ) C D"  
  using bij\_betw\_cong[of C g " $\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } g \ x$ " D] assms(2,3) by force  
ultimately show ?thesis using bij\_betw\_combine[OF \_ \_ assms(4)] by metis  
qed

lemma subst\_comp\_split:  
assumes "subst\_domain  $\vartheta \cap \text{range\_vars } \vartheta = \{\}$ "  
shows " $\vartheta = (\text{rm\_vars } (\text{subst\_domain } \vartheta - V) \ \vartheta) \circ_s (\text{rm\_vars } V \ \vartheta)$ " (is ?P)  
and " $\vartheta = (\text{rm\_vars } V \ \vartheta) \circ_s (\text{rm\_vars } (\text{subst\_domain } \vartheta - V) \ \vartheta)$ " (is ?Q)  
proof -  
let ?rm1 = "rm\_vars (subst\_domain  $\vartheta - V$ )  $\vartheta$ " and ?rm2 = "rm\_vars V  $\vartheta$ "  
have "subst\_domain ?rm2  $\cap$  range\_vars ?rm1 = {}"  
  "subst\_domain ?rm1  $\cap$  range\_vars ?rm2 = {}"  
  using assms unfolding range\_vars\_alt\_def by (force simp add: subst\_domain\_def)+  
hence \*: " $\bigwedge v. v \in \text{subst\_domain } ?rm1 \implies (?rm1 \circ_s ?rm2) \ v = \vartheta \ v$ "  
  " $\bigwedge v. v \in \text{subst\_domain } ?rm2 \implies (?rm2 \circ_s ?rm1) \ v = \vartheta \ v$ "  
  using ident\_comp\_subst\_trm\_if\_disj[of ?rm2 ?rm1]  
  ident\_comp\_subst\_trm\_if\_disj[of ?rm1 ?rm2]  
  by (auto simp add: subst\_domain\_def)  
hence " $\bigwedge v. v \notin \text{subst\_domain } ?rm1 \implies (?rm1 \circ_s ?rm2) \ v = \vartheta \ v$ "  
  " $\bigwedge v. v \notin \text{subst\_domain } ?rm2 \implies (?rm2 \circ_s ?rm1) \ v = \vartheta \ v$ "  
  unfolding subst\_compose\_def by (auto simp add: subst\_domain\_def)  
hence " $\bigwedge v. (?rm1 \circ_s ?rm2) \ v = \vartheta \ v$ " " $\bigwedge v. (?rm2 \circ_s ?rm1) \ v = \vartheta \ v$ " using \* by blast+  
thus ?P ?Q by auto  
qed

lemma subst\_comp\_eq\_if\_disjoint\_vars:  
assumes "(subst\_domain  $\delta \cup \text{range\_vars } \delta) \cap (\text{subst\_domain } \gamma \cup \text{range\_vars } \gamma) = \{\}$ "  
shows " $\gamma \circ_s \delta = \delta \circ_s \gamma$ "  
proof -  
{ fix x assume "x  $\in$  subst\_domain  $\gamma$ "  
  hence " $(\gamma \circ_s \delta) \ x = \gamma \ x$ " " $(\delta \circ_s \gamma) \ x = \gamma \ x$ "  
    using assms unfolding range\_vars\_alt\_def by (force simp add: subst\_compose)+  
  hence " $(\gamma \circ_s \delta) \ x = (\delta \circ_s \gamma) \ x$ " by metis  
}  
moreover  
{ fix x assume "x  $\in$  subst\_domain  $\delta$ "  
  hence " $(\gamma \circ_s \delta) \ x = \delta \ x$ " " $(\delta \circ_s \gamma) \ x = \delta \ x$ "  
    using assms  
  }

```

    unfolding range_vars_alt_def by (auto simp add: subst_compose subst_domain_def)
    hence "( $\gamma \circ_s \delta$ ) x = ( $\delta \circ_s \gamma$ ) x" by metis
  } moreover
  { fix x assume "x  $\notin$  subst_domain  $\gamma$ " "x  $\notin$  subst_domain  $\delta$ "
    hence "( $\gamma \circ_s \delta$ ) x = ( $\delta \circ_s \gamma$ ) x" by (simp add: subst_compose subst_domain_def)
  } ultimately show ?thesis by auto
qed

lemma subst_eq_if_disjoint_vars_ground:
  fixes  $\xi \delta :: ('f, 'v)$  subst
  assumes "subst_domain  $\delta \cap$  subst_domain  $\xi = \{\}$ " "ground (subst_range  $\xi$ )" "ground (subst_range  $\delta$ )"
  shows "t .  $\delta$  .  $\xi =$  t .  $\xi$  .  $\delta$ "
by (metis assms subst_comp_eq_if_disjoint_vars range_vars_alt_def
    subst_subst_compose sup_bot.right_neutral)

lemma subst_img_bound: "subst_domain  $\delta \cup$  range_vars  $\delta \subseteq$  fv t  $\implies$  range_vars  $\delta \subseteq$  fv (t .  $\delta$ )"
proof -
  assume "subst_domain  $\delta \cup$  range_vars  $\delta \subseteq$  fv t"
  hence "subst_domain  $\delta \subseteq$  fv t" by blast
  thus ?thesis
    by (metis (no_types) range_vars_alt_def le_iff_sup subst_apply_fv_unfold
        subst_apply_fv_union subst_range.simps)
qed

lemma subst_all_fv_subset: "fv t  $\subseteq$  fvset M  $\implies$  fv (t .  $\vartheta$ )  $\subseteq$  fvset (M .set  $\vartheta$ )"
proof -
  assume *: "fv t  $\subseteq$  fvset M"
  { fix v assume "v  $\in$  fv t"
    hence "v  $\in$  fvset M" using * by auto
    then obtain t' where "t'  $\in$  M" "v  $\in$  fv t'" by auto
    hence "fv ( $\vartheta$  v)  $\subseteq$  fv (t' .  $\vartheta$ )"
      by (metis subst_apply_term.simps(1) subst_apply_fv_subset subst_apply_fv_unfold
          subtermeq_vars_subset vars_iff_subtermeq)
    hence "fv ( $\vartheta$  v)  $\subseteq$  fvset (M .set  $\vartheta$ )" using <t'  $\in$  M> by auto
  }
  thus ?thesis using subst_apply_fv_unfold[of t  $\vartheta$ ] by auto
qed

lemma subst_support_if_mgt_subst_idem:
  assumes " $\vartheta \preceq_o \delta$ " "subst_idem  $\vartheta$ "
  shows " $\vartheta$  supports  $\delta$ "
proof -
  from < $\vartheta \preceq_o \delta$ > obtain  $\sigma$  where " $\delta = \vartheta \circ_s \sigma$ " by blast
  hence " $\bigwedge v. \vartheta v \cdot \delta = \text{Var } v \cdot (\vartheta \circ_s \vartheta \circ_s \sigma)$ " by simp
  hence " $\bigwedge v. \vartheta v \cdot \delta = \text{Var } v \cdot (\vartheta \circ_s \sigma)$ " using <subst_idem  $\vartheta$ > unfolding subst_idem_def by simp
  hence " $\bigwedge v. \vartheta v \cdot \delta = \text{Var } v \cdot \delta$ " using  $\sigma$  by simp
  thus " $\vartheta$  supports  $\delta$ " by simp
qed

lemma subst_support_iff_mgt_if_subst_idem:
  assumes "subst_idem  $\vartheta$ "
  shows " $\vartheta \preceq_o \delta \iff \vartheta$  supports  $\delta$ "
proof
  show " $\vartheta \preceq_o \delta \implies \vartheta$  supports  $\delta$ " by (fact subst_support_if_mgt_subst_idem[OF _ <subst_idem  $\vartheta$ >])
  show " $\vartheta$  supports  $\delta \implies \vartheta \preceq_o \delta$ " by (fact subst_supportD)
qed

lemma subst_support_comp:
  fixes  $\vartheta \delta \mathcal{I} :: ('a, 'b)$  subst
  assumes " $\vartheta$  supports  $\mathcal{I}$ " " $\delta$  supports  $\mathcal{I}$ "
  shows "( $\vartheta \circ_s \delta$ ) supports  $\mathcal{I}$ "
by (metis (no_types) assms subst_agreement subst_apply_term.simps(1) subst_subst_compose)

```

```

lemma subst_support_comp':
  fixes  $\vartheta \ \delta \ \sigma :: ('a, 'b) \text{subst}$ 
  assumes " $\vartheta$  supports  $\delta$ "
  shows " $\vartheta$  supports  $(\delta \circ_s \sigma)$ " " $\sigma$  supports  $\delta \implies \vartheta$  supports  $(\sigma \circ_s \delta)$ "
using assms unfolding subst_support_def by (metis subst_compose_assoc, metis)

lemma subst_support_comp_split:
  fixes  $\vartheta \ \delta \ \mathcal{I} :: ('a, 'b) \text{subst}$ 
  assumes " $(\vartheta \circ_s \delta)$  supports  $\mathcal{I}$ "
  shows "subst_domain  $\vartheta \cap \text{range\_vars } \vartheta = \{\}$   $\implies \vartheta$  supports  $\mathcal{I}$ "
  and "subst_domain  $\vartheta \cap \text{subst\_domain } \delta = \{\}$   $\implies \delta$  supports  $\mathcal{I}$ "
proof -
  assume "subst_domain  $\vartheta \cap \text{range\_vars } \vartheta = \{\}$ "
  hence "subst_idem  $\vartheta$ " by (metis subst_idemI)
  have " $\vartheta \preceq \mathcal{I}$ " using assms subst_compose_assoc[of  $\vartheta \ \delta \ \mathcal{I}$ ] unfolding subst_compose_def by metis
  show " $\vartheta$  supports  $\mathcal{I}$ " using subst_support_if_mgt_subst_idem[OF  $\vartheta \preceq \mathcal{I}$ ] (subst_idem  $\vartheta$ ) by auto
next
  assume "subst_domain  $\vartheta \cap \text{subst\_domain } \delta = \{\}$ "
  moreover have " $\forall v \in \text{subst\_domain } (\vartheta \circ_s \delta). (\vartheta \circ_s \delta) v \cdot \mathcal{I} = \mathcal{I} v$ " using assms by metis
  ultimately have " $\forall v \in \text{subst\_domain } \delta. \delta v \cdot \mathcal{I} = \mathcal{I} v$ "
  using var_not_in_subst_dom unfolding subst_compose_def
  by (metis IntI empty_iff subst_apply_term.simps(1))
  thus " $\delta$  supports  $\mathcal{I}$ " by force
qed

lemma subst_idem_support: "subst_idem  $\vartheta \implies \vartheta$  supports  $\vartheta \circ_s \delta$ "
unfolding subst_idem_def by (metis subst_support_def subst_compose_assoc)

lemma subst_idem_iff_self_support: "subst_idem  $\vartheta \longleftrightarrow \vartheta$  supports  $\vartheta$ "
using subst_support_def[of  $\vartheta \ \vartheta$ ] unfolding subst_idem_def by auto

lemma subterm_subst_neq: " $t \sqsubset t' \implies t \cdot s \neq t' \cdot s$ "
by (metis subst_mono_neq)

lemma fv_Fun_subst_neq: " $x \in \text{fv } (\text{Fun } f \ T) \implies \sigma \ x \neq \text{Fun } f \ T \cdot \sigma$ "
using subterm_subst_neq[of "Var x" "Fun f T"] vars_iff_subterm_or_eq[of x "Fun f T"] by auto

lemma subterm_subst_unfold:
  assumes " $t \sqsubseteq s \cdot \vartheta$ "
  shows " $(\exists s'. s' \sqsubseteq s \wedge t = s' \cdot \vartheta) \vee (\exists x \in \text{fv } s. t \sqsubset \vartheta \ x)$ "
using assms
proof (induction s)
  case (Fun f T) thus ?case
  proof (cases "t = Fun f T  $\cdot \vartheta$ ")
    case True thus ?thesis using Fun by auto
  next
    case False
    then obtain  $s'$  where  $s': s' \in \text{set } T$  " $t \sqsubseteq s' \cdot \vartheta$ " using Fun by auto
    hence " $(\exists s''. s'' \sqsubseteq s' \wedge t = s'' \cdot \vartheta) \vee (\exists x \in \text{fv } s'. t \sqsubset \vartheta \ x)$ " by (metis Fun.IH)
    thus ?thesis using  $s'(1)$  by auto
  qed
qed
qed simp

lemma subterm_subst_img_subterm:
  assumes " $t \sqsubseteq s \cdot \vartheta$ " " $\bigwedge s'. s' \sqsubseteq s \implies t \neq s' \cdot \vartheta$ "
  shows " $\exists w \in \text{fv } s. t \sqsubset \vartheta \ w$ "
using subterm_subst_unfold[OF assms(1)] assms(2) by force

lemma subterm_subst_not_img_subterm:
  assumes " $t \sqsubseteq s \cdot \mathcal{I}$ " " $\neg(\exists w \in \text{fv } s. t \sqsubseteq \mathcal{I} \ w)$ "
  shows " $\exists f \ T. \text{Fun } f \ T \sqsubseteq s \wedge t = \text{Fun } f \ T \cdot \mathcal{I}$ "
proof (rule ccontr)
  assume " $\neg(\exists f \ T. \text{Fun } f \ T \sqsubseteq s \wedge t = \text{Fun } f \ T \cdot \mathcal{I})$ "

```

```

hence "⋀ f T. Fun f T ⊆ s ⇒ t ≠ Fun f T · I" by simp
moreover have "⋀ x. Var x ⊆ s ⇒ t ≠ Var x · I"
  using assms(2) vars_iff_subtermeq by force
ultimately have "⋀ s'. s' ⊆ s ⇒ t ≠ s' · I" by (metis "term.exhaust")
thus False using assms subterm_subst_img_subterm by blast
qed

lemma subst_apply_img_var:
  assumes "v ∈ fv (t · δ)" "v ∉ fv t"
  obtains w where "w ∈ fv t" "v ∈ fv (δ w)"
using assms by (induct t) auto

lemma subst_apply_img_var':
  assumes "x ∈ fv (t · δ)" "x ∉ fv t"
  shows "∃ y ∈ fv t. x ∈ fv (δ y)"
by (metis assms subst_apply_img_var)

lemma nth_map_subst:
  fixes ϑ::('f,'v) subst and T::('f,'v) term list and i::nat
  shows "i < length T ⇒ (map (λt. t · ϑ) T) ! i = (T ! i) · ϑ"
by (fact nth_map)

lemma subst_subterm:
  assumes "Fun f T ⊆ t · ϑ"
  shows "(∃ S. Fun f S ⊆ t ∧ Fun f S · ϑ = Fun f T) ∨
    (∃ s ∈ subst_range ϑ. Fun f T ⊆ s)"
using assms subterm_subst_not_img_subterm by (cases "∃ s ∈ subst_range ϑ. Fun f T ⊆ s") fastforce+

lemma subst_subterm':
  assumes "Fun f T ⊆ t · ϑ"
  shows "∃ S. length S = length T ∧ (Fun f S ⊆ t ∨ (∃ s ∈ subst_range ϑ. Fun f S ⊆ s))"
using subst_subterm[OF assms] by auto

lemma subst_subterm'':
  assumes "s ∈ subterms (t · ϑ)"
  shows "(∃ u ∈ subterms t. s = u · ϑ) ∨ s ∈ subterms_set (subst_range ϑ)"
proof (cases s)
  case (Var x)
  thus ?thesis
    using assms subterm_subst_not_img_subterm vars_iff_subtermeq
    by (cases "s = t · ϑ") fastforce+
next
  case (Fun f T)
  thus ?thesis
    using subst_subterm[of f T t ϑ] assms
    by fastforce
qed

```

### 2.3.3 More Small Lemmata

```

lemma funs_term_subst: "funs_term (t · ϑ) = funs_term t ∪ (⋃ x ∈ fv t. funs_term (ϑ x))"
by (induct t) auto

lemma fv_set_subst_img_eq:
  assumes "X ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "fv_set (δ ' (Y - X)) = fv_set (δ ' Y) - X"
using assms unfolding range_vars_alt_def by force

lemma subst_Fun_index_eq:
  assumes "i < length T" "Fun f T · δ = Fun g T' · δ"
  shows "T ! i · δ = T' ! i · δ"
proof -
  have "map (λx. x · δ) T = map (λx. x · δ) T'" using assms by simp

```

```

  thus ?thesis by (metis assms(1) length_map nth_map)
qed

```

```

lemma fv_exists_if_unifiable_and_neq:
  fixes t t':: "('a,'b) term" and  $\delta \vartheta:: "('a,'b) \text{subst}$ "
  assumes "t  $\neq$  t'" "t  $\cdot \vartheta = t' \cdot \vartheta$ "
  shows "fv t  $\cup$  fv t'  $\neq \{\}$ "
proof
  assume "fv t  $\cup$  fv t' =  $\{\}$ "
  hence "fv t =  $\{\}$ " "fv t' =  $\{\}$ " by auto
  hence "t  $\cdot \vartheta = t$ " "t'  $\cdot \vartheta = t'$ " by auto
  hence "t = t'" using assms(2) by metis
  thus False using assms(1) by auto
qed

```

```

lemma const_subterm_subst: "Fun c []  $\sqsubseteq$  t  $\implies$  Fun c []  $\sqsubseteq$  t  $\cdot \sigma$ "
by (induct t) auto

```

```

lemma const_subterm_subst_var_obtain:
  assumes "Fun c []  $\sqsubseteq$  t  $\cdot \sigma$ " " $\neg$  Fun c []  $\sqsubseteq$  t"
  obtains x where "x  $\in$  fv t" "Fun c []  $\sqsubseteq$   $\sigma$  x"
using assms by (induct t) auto

```

```

lemma const_subterm_subst_cases:
  assumes "Fun c []  $\sqsubseteq$  t  $\cdot \sigma$ "
  shows "Fun c []  $\sqsubseteq$  t  $\vee (\exists x \in \text{fv t}. x \in \text{subst\_domain } \sigma \wedge \text{Fun c []} \sqsubseteq \sigma x)"
proof (cases "Fun c []  $\sqsubseteq$  t")
  case False
  then obtain x where "x  $\in$  fv t" "Fun c []  $\sqsubseteq$   $\sigma$  x"
    using const_subterm_subst_var_obtain[OF assms] by moura
  thus ?thesis by (cases "x  $\in$  subst_domain  $\sigma$ ") auto
qed simp$ 
```

```

lemma fv_pairs_subst_fv_subset:
  assumes "x  $\in$  fv_pairs F"
  shows "fv ( $\vartheta$  x)  $\subseteq$  fv_pairs (F  $\cdot_{\text{pairs}}$   $\vartheta$ )"
using assms
proof (induction F)
  case (Cons f F)
  then obtain t t' where "f = (t,t')" by (metis surj_pair)
  show ?case
  proof (cases "x  $\in$  fv_pairs F")
    case True thus ?thesis
      using Cons.IH
      unfolding subst_apply_pairs_def
      by auto
    next
    case False
    hence "x  $\in$  fv t  $\cup$  fv t'" using Cons.prem f by simp
    hence "fv ( $\vartheta$  x)  $\subseteq$  fv (t  $\cdot \vartheta$ )  $\cup$  fv (t'  $\cdot \vartheta$ )" using fv_subst_subset[of x] by force
    thus ?thesis using f unfolding subst_apply_pairs_def by auto
  qed
qed simp

```

```

lemma fv_pairs_step_subst: "fv_set ( $\delta$  ' fv_pairs F) = fv_pairs (F  $\cdot_{\text{pairs}}$   $\delta$ )"
proof (induction F)
  case (Cons f F)
  obtain t t' where "f = (t,t'" by moura
  thus ?case
    using Cons
    by (simp add: subst_apply_pairs_def subst_apply_fv_unfold)
qed (simp_all add: subst_apply_pairs_def)

```



```

lemma fv_pairs_subst_obtain_var:
  fixes  $\delta :: ('a, 'b) \text{ subst}$ 
  assumes " $x \in \text{fv\_pairs } (F \cdot_{\text{pairs}} \delta)$ "
  shows " $\exists y \in \text{fv\_pairs } F. x \in \text{fv } (\delta y)$ "
  using assms
proof (induction F)
  case (Cons f F)
  then obtain t s where  $f = (t, s)$  by (metis surj_pair)

  from Cons.IH show ?case
proof (cases " $x \in \text{fv\_pairs } (F \cdot_{\text{pairs}} \delta)$ ")
  case False
  hence " $x \in \text{fv } (t \cdot \delta) \vee x \in \text{fv } (s \cdot \delta)$ "
  using f Cons.prem by
  by (simp add: subst_apply_pairs_def)
  hence " $(\exists y \in \text{fv } t. x \in \text{fv } (\delta y)) \vee (\exists y \in \text{fv } s. x \in \text{fv } (\delta y))$ " by (metis fv_subst_obtain_var)
  thus ?thesis using f by (auto simp add: subst_apply_pairs_def)
qed (auto simp add: Cons.IH)
qed (simp add: subst_apply_pairs_def)

lemma pair_subst_ident[intro]: " $(\text{fv } t \cup \text{fv } t') \cap \text{subst\_domain } \vartheta = \{\} \implies (t, t') \cdot_p \vartheta = (t, t')$ "
by auto

lemma pairs_substI[intro]:
  assumes " $\text{subst\_domain } \vartheta \cap (\bigcup (s, t) \in M. \text{fv } s \cup \text{fv } t) = \{\}$ "
  shows " $M \cdot_{\text{pset}} \vartheta = M$ "
proof -
  { fix m assume M: " $m \in M$ "
    then obtain s t where  $m = (s, t)$  by (metis surj_pair)
    hence " $(\text{fv } s \cup \text{fv } t) \cap \text{subst\_domain } \vartheta = \{\}$ " using assms M by auto
    hence " $m \cdot_p \vartheta = m$ " using m by auto
  } thus ?thesis by (simp add: image_cong)
qed

lemma fv_pairs_subst: " $\text{fv\_pairs } (F \cdot_{\text{pairs}} \vartheta) = \text{fv\_set } (\vartheta \cdot (\text{fv\_pairs } F))$ "
proof (induction F)
  case (Cons g G)
  obtain t t' where  $g = (t, t')$  by (metis surj_pair)
  thus ?case
  using Cons.IH
  by (simp add: subst_apply_pairs_def subst_apply_fv_unfold)
qed (simp add: subst_apply_pairs_def)

lemma fv_pairs_subst_subset:
  assumes " $\text{fv\_pairs } (F \cdot_{\text{pairs}} \delta) \subseteq \text{subst\_domain } \sigma$ "
  shows " $\text{fv\_pairs } F \subseteq \text{subst\_domain } \sigma \cup \text{subst\_domain } \delta$ "
  using assms
proof (induction F)
  case (Cons g G)
  hence IH: " $\text{fv\_pairs } G \subseteq \text{subst\_domain } \sigma \cup \text{subst\_domain } \delta$ "
  by (simp add: subst_apply_pairs_def)
  obtain t t' where  $g = (t, t')$  by (metis surj_pair)
  hence " $\text{fv } (t \cdot \delta) \subseteq \text{subst\_domain } \sigma$ " " $\text{fv } (t' \cdot \delta) \subseteq \text{subst\_domain } \sigma$ "
  using Cons.prem by (simp_all add: subst_apply_pairs_def)
  hence " $\text{fv } t \subseteq \text{subst\_domain } \sigma \cup \text{subst\_domain } \delta$ " " $\text{fv } t' \subseteq \text{subst\_domain } \sigma \cup \text{subst\_domain } \delta$ "
  using subst_apply_fv_unfold[of _  $\delta$ ] by force+
  thus ?case using IH g by (simp add: subst_apply_pairs_def)
qed (simp add: subst_apply_pairs_def)

lemma pairs_subst_comp: " $F \cdot_{\text{pairs}} \delta \circ_s \vartheta = ((F \cdot_{\text{pairs}} \delta) \cdot_{\text{pairs}} \vartheta)$ "
by (induct F) (auto simp add: subst_apply_pairs_def)

lemma pairs_substI'[intro]:

```

```

"subst_domain  $\vartheta \cap fv_{pairs} F = \{\} \implies F \cdot_{pairs} \vartheta = F"$ 
by (induct F) (force simp add: subst_apply_pairs_def)+

lemma subst_pair_compose[simp]: " $d \cdot_p (\delta \circ_s \mathcal{I}) = d \cdot_p \delta \cdot_p \mathcal{I}$ "
proof -
  obtain t s where "d = (t,s)" by moura
  thus ?thesis by auto
qed

lemma subst_pairs_compose[simp]: " $D \cdot_{pset} (\delta \circ_s \mathcal{I}) = D \cdot_{pset} \delta \cdot_{pset} \mathcal{I}$ "
by auto

lemma subst_apply_pair_pair: " $(t, s) \cdot_p \mathcal{I} = (t \cdot \mathcal{I}, s \cdot \mathcal{I})$ "
by (rule prod.case)

lemma subst_apply_pairs_nil[simp]: " $[] \cdot_{pairs} \delta = []$ "
unfolding subst_apply_pairs_def by simp

lemma subst_apply_pairs_singleton[simp]: " $[(t,s)] \cdot_{pairs} \delta = [(t \cdot \delta, s \cdot \delta)]$ "
unfolding subst_apply_pairs_def by simp

lemma subst_apply_pairs_Var[iff]: " $F \cdot_{pairs} Var = F$ " by (simp add: subst_apply_pairs_def)

lemma subst_apply_pairs_pset_subst: " $set (F \cdot_{pairs} \vartheta) = set F \cdot_{pset} \vartheta$ "
unfolding subst_apply_pairs_def by force

```

### 2.3.4 Finite Substitutions

```

inductive_set fsubst:: "('a,'b) subst set" where
  fvar:      "Var  $\in$  fsubst"
| FUpdate:  "[ $\vartheta \in$  fsubst;  $v \notin$  subst_domain  $\vartheta$ ;  $t \neq$  Var v]  $\implies \vartheta(v := t) \in$  fsubst"

lemma finite_dom_iff_fsubst:
  "finite (subst_domain  $\vartheta$ )  $\longleftrightarrow \vartheta \in$  fsubst"
proof
  assume "finite (subst_domain  $\vartheta$ )" thus " $\vartheta \in$  fsubst"
  proof (induction "subst_domain  $\vartheta$ " arbitrary:  $\vartheta$  rule: finite.induct)
    case emptyI
    hence " $\vartheta = Var$ " using empty_dom_iff_empty_subst by metis
    thus ?case using fvar by simp
  next
    case (insertI  $\vartheta'_{dom} v$ ) thus ?case
    proof (cases " $v \in \vartheta'_{dom}$ ")
      case True
      hence " $\vartheta'_{dom} =$  subst_domain  $\vartheta$ " using (insert v  $\vartheta'_{dom} =$  subst_domain  $\vartheta$ ) by auto
      thus ?thesis using insertI.hyps(2) by metis
    next
      case False
      let ? $\vartheta' = \lambda w. if w \in \vartheta'_{dom} then \vartheta w else Var w$ 
      have "subst_domain ? $\vartheta' = \vartheta'_{dom}$ "
      using (v  $\notin \vartheta'_{dom}$ ) (insert v  $\vartheta'_{dom} =$  subst_domain  $\vartheta$ )
      by (auto simp add: subst_domain_def)
      hence "? $\vartheta' \in$  fsubst" using insertI.hyps(2) by simp
      moreover have "? $\vartheta'(v := \vartheta v) = (\lambda w. if w \in$  insert v  $\vartheta'_{dom}$  then  $\vartheta w$  else Var w)" by auto
      hence "? $\vartheta'(v := \vartheta v) = \vartheta$ "
      using (insert v  $\vartheta'_{dom} =$  subst_domain  $\vartheta$ )
      by (auto simp add: subst_domain_def)
      ultimately show ?thesis
      using FUpdate[of ? $\vartheta'$  v " $\vartheta v$ "] False insertI.hyps(3)
      by (auto simp add: subst_domain_def)
    qed
  qed
next

```

```

  assume "θ ∈ fsubst" thus "finite (subst_domain θ)"
  by (induct θ, simp, metis subst_dom_insert_finite)
qed

lemma fsubst_induct[case_names fvar FUpdate, induct set: finite]:
  assumes "finite (subst_domain δ)" "P Var"
  and "⋀θ v t. [[finite (subst_domain θ); v ∉ subst_domain θ; t ≠ Var v; P θ]] ⟹ P (θ(v := t))"
  shows "P δ"
using assms finite_dom_iff_fsubst fsubst.induct by metis

lemma fun_upd_fsubst: "s(v := t) ∈ fsubst ⟷ s ∈ fsubst"
using subst_dom_insert_finite[of s] finite_dom_iff_fsubst by blast

lemma finite_img_if_fsubst: "s ∈ fsubst ⟹ finite (subst_range s)"
using finite_dom_iff_fsubst finite_subst_img_if_finite_dom' by blast

```

### 2.3.5 Unifiers and Most General Unifiers (MGUs)

abbreviation  $\text{Unifier}::('f, 'v) \text{subst} \Rightarrow ('f, 'v) \text{term} \Rightarrow ('f, 'v) \text{term} \Rightarrow \text{bool}$  where  
 $\text{"Unifier } \sigma \ t \ u \equiv (t \cdot \sigma = u \cdot \sigma)"$

abbreviation  $\text{MGU}::('f, 'v) \text{subst} \Rightarrow ('f, 'v) \text{term} \Rightarrow ('f, 'v) \text{term} \Rightarrow \text{bool}$  where  
 $\text{"MGU } \sigma \ t \ u \equiv \text{Unifier } \sigma \ t \ u \wedge (\forall \vartheta. \text{Unifier } \vartheta \ t \ u \longrightarrow \sigma \preceq_o \vartheta)"$

```

lemma MGUI[intro]:
  shows "[[t · σ = u · σ; ⋀θ::('f, 'v) subst. t · θ = u · θ ⟹ σ ⪯_o θ]] ⟹ MGU σ t u"
by auto

```

```

lemma UnifierD[dest]:
  fixes σ::('f, 'v) subst and f g::'f and X Y::('f, 'v) term list
  assumes "Unifier σ (Fun f X) (Fun g Y)"
  shows "f = g" "length X = length Y"
proof -
  from assms show "f = g" by auto

  from assms have "Fun f X · σ = Fun g Y · σ" by auto
  hence "length (map (λx. x · σ) X) = length (map (λx. x · σ) Y)" by auto
  thus "length X = length Y" by auto
qed

```

```

lemma MGUD[dest]:
  fixes σ::('f, 'v) subst and f g::'f and X Y::('f, 'v) term list
  assumes "MGU σ (Fun f X) (Fun g Y)"
  shows "f = g" "length X = length Y"
using assms by (auto intro!: UnifierD[of f X σ g Y])

```

```

lemma MGU_sym[sym]: "MGU σ s t ⟹ MGU σ t s" by auto
lemma Unifier_sym[sym]: "Unifier σ s t ⟹ Unifier σ t s" by auto

```

```

lemma MGU_nil: "MGU Var s t ⟷ s = t" by fastforce

```

```

lemma Unifier_comp: "Unifier (θ ∘_s δ) t u ⟹ Unifier δ (t · θ) (u · θ)"
by simp

```

```

lemma Unifier_comp': "Unifier δ (t · θ) (u · θ) ⟹ Unifier (θ ∘_s δ) t u"
by simp

```

```

lemma Unifier_excludes_subterm:
  assumes θ: "Unifier θ t u"
  shows "¬t ⊆ u"
proof
  assume "t ⊆ u"
  hence "t · θ ⊆ u · θ" using subst_mono_neq by metis

```

```

  hence "t ·  $\vartheta \neq u \cdot \vartheta$ " by simp
  moreover from  $\vartheta$  have "t ·  $\vartheta = u \cdot \vartheta$ " by auto
  ultimately show False ..
qed

```

```

lemma MGU_is_Unifier: "MGU  $\sigma$  t u  $\implies$  Unifier  $\sigma$  t u" by (rule conjunct1)

```

```

lemma MGU_Var1:
  assumes " $\neg \text{Var } v \sqsubset t$ "
  shows "MGU (Var(v := t)) (Var v) t"
proof (intro MGUI exI)
  show "Var v · (Var(v := t)) = t · (Var(v := t))" using assms subst_no_occs by fastforce
next
  fix  $\vartheta::('a, 'b)$  subst" assume th: "Var v ·  $\vartheta = t \cdot \vartheta$ "
  show " $\vartheta = (\text{Var}(v := t)) \circ_s \vartheta$ "
  proof
    fix s show "s ·  $\vartheta = s \cdot ((\text{Var}(v := t)) \circ_s \vartheta)$ " using th by (induct s) auto
  qed
qed

```

```

lemma MGU_Var2: " $v \notin \text{fv } t \implies \text{MGU } (\text{Var}(v := t)) (\text{Var } v) t$ "
by (metis (no_types) MGU_Var1 vars_iff_subterm_or_eq)

```

```

lemma MGU_Var3: "MGU Var (Var v) (Var w)  $\longleftrightarrow v = w$ " by fastforce

```

```

lemma MGU_Const1: "MGU Var (Fun c []) (Fun d [])  $\longleftrightarrow c = d$ " by fastforce

```

```

lemma MGU_Const2: "MGU  $\vartheta$  (Fun c []) (Fun d [])  $\implies c = d$ " by auto

```

```

lemma MGU_Fun:
  assumes "MGU  $\vartheta$  (Fun f X) (Fun g Y)"
  shows "f = g" "length X = length Y"
proof -
  let ?F = " $\lambda \vartheta$  X. map ( $\lambda x. x \cdot \vartheta$ ) X"
  from assms have
    "[f = g; ?F  $\vartheta$  X = ?F  $\vartheta$  Y;  $\forall \vartheta'. f = g \wedge ?F \vartheta' X = ?F \vartheta' Y \longrightarrow \vartheta \preceq_o \vartheta'$ ]"  $\implies$  "length X = length Y"
    using map_eq_imp_length_eq by auto
  thus "f = g" "length X = length Y" using assms by auto
qed

```

```

lemma Unifier_Fun:
  assumes "Unifier  $\vartheta$  (Fun f (x#X)) (Fun g (y#Y))"
  shows "Unifier  $\vartheta$  x y" "Unifier  $\vartheta$  (Fun f X) (Fun g Y)"
using assms by simp_all

```

```

lemma Unifier_subst_idem_subst:
  "subst_idem r  $\implies$  Unifier s (t · r) (u · r)  $\implies$  Unifier (r  $\circ_s$  s) (t · r) (u · r)"
by (metis (no_types, lifting) subst_idem_def subst_subst_compose)

```

```

lemma subst_idem_comp:
  "subst_idem r  $\implies$  Unifier s (t · r) (u · r)  $\implies$ 
  ( $\bigwedge q. \text{Unifier } q (t \cdot r) (u \cdot r) \implies s \circ_s q = q$ )  $\implies$ 
  subst_idem (r  $\circ_s$  s)"
by (frule Unifier_subst_idem_subst, blast, metis subst_idem_def subst_compose_assoc)

```

```

lemma Unifier_mgt: "[Unifier  $\delta$  t u;  $\delta \preceq_o \vartheta$ ]  $\implies$  Unifier  $\vartheta$  t u" by auto

```

```

lemma Unifier_support: "[Unifier  $\delta$  t u;  $\delta$  supports  $\vartheta$ ]  $\implies$  Unifier  $\vartheta$  t u"
using subst_supportD Unifier_mgt by metis

```

```

lemma MGU_mgt: "[MGU  $\sigma$  t u; MGU  $\delta$  t u]  $\implies \sigma \preceq_o \delta$ " by auto

```

```

lemma Unifier_trm_fv_bound:

```

```

"[[Unifier s t u; v ∈ fv t]] ⇒ v ∈ subst_domain s ∪ range_vars s ∪ fv u"
proof (induction t arbitrary: s u)
  case (Fun f X)
  hence "v ∈ fv (u · s) ∨ v ∈ subst_domain s" by (metis subst_not_dom_fixed)
  thus ?case by (metis (no_types) Un_iff contra_subsetD subst_sends_fv_to_img)
qed (metis (no_types) UnI1 UnI2 subsetCE no_var_subterm subst_sends_dom_to_img
      subst_to_var_is_var trm_subst_ident' vars_iff_subterm_or_eq)

lemma Unifier_rm_var: "[[Unifier ϑ s t; v ∉ fv s ∪ fv t]] ⇒ Unifier (rm_var v ϑ) s t"
by (auto simp add: repl_invariance)

lemma Unifier_ground_rm_vars:
  assumes "ground (subst_range s)" "Unifier (rm_vars X s) t t'"
  shows "Unifier s t t'"
by (rule Unifier_support[OF assms(2) rm_vars_ground_supports[OF assms(1)]]])

lemma Unifier_dom_restrict:
  assumes "Unifier s t t'" "fv t ∪ fv t' ⊆ S"
  shows "Unifier (rm_vars (UNIV - S) s) t t'"
proof -
  let ?s = "rm_vars (UNIV - S) s"
  show ?thesis using term_subst_eq_conv[of t s ?s] term_subst_eq_conv[of t' s ?s] assms by auto
qed

```

### 2.3.6 Well-formedness of Substitutions and Unifiers

```

inductive_set wf_subst_set::('a,'b) subst set where
  Empty[simp]: "Var ∈ wf_subst_set"
/ Insert[simp]:
  "[[ϑ ∈ wf_subst_set; v ∉ subst_domain ϑ;
    v ∉ range_vars ϑ; fv t ∩ (insert v (subst_domain ϑ)) = {}]]
  ⇒ ϑ(v := t) ∈ wf_subst_set"

definition wf_subst::('a,'b) subst ⇒ bool where
  "wf_subst ϑ ≡ subst_domain ϑ ∩ range_vars ϑ = {} ∧ finite (subst_domain ϑ)"

definition wf_MGU::('a,'b) subst ⇒ ('a,'b) term ⇒ ('a,'b) term ⇒ bool where
  "wf_MGU ϑ s t ≡ wf_subst ϑ ∧ MGU ϑ s t ∧ subst_domain ϑ ∪ range_vars ϑ ⊆ fv s ∪ fv t"

lemma wf_subst_subst_idem: "wf_subst ϑ ⇒ subst_idem ϑ" using subst_idemI[of ϑ] unfolding
wf_subst_def by fast

lemma wf_subst_properties: "ϑ ∈ wf_subst_set = wf_subst ϑ"
proof
  show "wf_subst ϑ ⇒ ϑ ∈ wf_subst_set" unfolding wf_subst_def
  proof -
    assume "subst_domain ϑ ∩ range_vars ϑ = {} ∧ finite (subst_domain ϑ)"
    hence "finite (subst_domain ϑ)" "subst_domain ϑ ∩ range_vars ϑ = {}"
    by auto
    thus "ϑ ∈ wf_subst_set"
  proof (induction ϑ rule: fsubst_induct)
    case fvar thus ?case by simp
  next
    case (FUpdate δ v t)
    have "subst_domain δ ⊆ subst_domain (δ(v := t))" "range_vars δ ⊆ range_vars (δ(v := t))"
    using FUpdate.hyps(2,3) subst_img_update
    unfolding range_vars_alt_def by (fastforce simp add: subst_domain_def)+
    hence "subst_domain δ ∩ range_vars δ = {}" using FUpdate.prem(1) by blast
    hence "δ ∈ wf_subst_set" using FUpdate.IH by metis

    have *: "range_vars (δ(v := t)) = range_vars δ ∪ fv t"
    using FUpdate.hyps(2) subst_img_update[OF _ FUpdate.hyps(3)]
    by fastforce
  end
end

```

```

    hence "fv t  $\cap$  insert v (subst_domain  $\delta$ ) = {}"
    using FUpdate.premis subst_dom_update2[OF FUpdate.hyps(3)] by blast
  moreover have "subst_domain ( $\delta$ (v := t)) = insert v (subst_domain  $\delta$ )"
    by (meson FUpdate.hyps(3) subst_dom_update2)
  hence "v  $\notin$  range_vars  $\delta$ " using FUpdate.premis * by blast
  ultimately show ?case using Insert[OF ( $\delta \in \text{wf\_subst\_set}$ ) (v  $\notin$  subst_domain  $\delta$ )] by metis
qed
qed

show " $\vartheta \in \text{wf\_subst\_set} \implies \text{wf\_subst } \vartheta$ " unfolding wf_subst_def
proof (induction  $\vartheta$  rule: wf_subst_set.induct)
  case Empty thus ?case by simp
next
  case (Insert  $\sigma$  v t)
  hence 1: "subst_domain  $\sigma \cap$  range_vars  $\sigma$  = {}" by simp
  hence 2: "subst_domain ( $\sigma$ (v := t))  $\cap$  range_vars  $\sigma$  = {}"
    using Insert.hyps(3) by (auto simp add: subst_domain_def)
  have 3: "fv t  $\cap$  subst_domain ( $\sigma$ (v := t)) = {}"
    using Insert.hyps(4) by (auto simp add: subst_domain_def)
  have 4: " $\sigma$  v = Var v" using (v  $\notin$  subst_domain  $\sigma$ ) by (simp add: subst_domain_def)

  from Insert.IH have "finite (subst_domain  $\sigma$ )" by simp
  hence 5: "finite (subst_domain ( $\sigma$ (v := t)))" using subst_dom_insert_finite[of  $\sigma$ ] by simp

  have "subst_domain ( $\sigma$ (v := t))  $\cap$  range_vars ( $\sigma$ (v := t)) = {}"
  proof (cases "t = Var v")
    case True
    hence "range_vars ( $\sigma$ (v := t)) = range_vars  $\sigma$ "
      using 4 fun_upd_triv term.inject(1)
      unfolding range_vars_alt_def by (auto simp add: subst_domain_def)
    thus "subst_domain ( $\sigma$ (v := t))  $\cap$  range_vars ( $\sigma$ (v := t)) = {}"
      using 1 2 3 by auto
  next
    case False
    hence "range_vars ( $\sigma$ (v := t)) = fv t  $\cup$  (range_vars  $\sigma$ )"
      using 4 subst_img_update[of  $\sigma$  v] by auto
    thus "subst_domain ( $\sigma$ (v := t))  $\cap$  range_vars ( $\sigma$ (v := t)) = {}" using 1 2 3 by blast
  qed
  thus ?case using 5 by blast
qed
qed

lemma wf_subst_induct[consumes 1, case_names Empty Insert]:
  assumes "wf_subst  $\delta$ " "P Var"
  and " $\bigwedge \vartheta$  v t.  $\llbracket \text{wf\_subst } \vartheta; P \vartheta; v \notin \text{subst\_domain } \vartheta; v \notin \text{range\_vars } \vartheta; \text{fv } t \cap \text{insert } v (\text{subst\_domain } \vartheta) = \{\} \rrbracket \implies P (\vartheta(v := t))$ "
  shows "P  $\delta$ "
proof -
  from assms(1,3) wf_subst_properties have
    " $\delta \in \text{wf\_subst\_set}$ "
    " $\bigwedge \vartheta$  v t.  $\llbracket \vartheta \in \text{wf\_subst\_set}; P \vartheta; v \notin \text{subst\_domain } \vartheta; v \notin \text{range\_vars } \vartheta; \text{fv } t \cap \text{insert } v (\text{subst\_domain } \vartheta) = \{\} \rrbracket \implies P (\vartheta(v := t))$ "
  by blast+
  thus "P  $\delta$ " using wf_subst_set.induct assms(2) by blast
qed

lemma wf_subst_fsubst: "wf_subst  $\delta \implies \delta \in \text{fsubst}$ "
unfolding wf_subst_def using finite_dom_iff_fsubst by blast

lemma wf_subst_nil: "wf_subst Var" unfolding wf_subst_def by simp

```

```

lemma wf_MGU_nil: "MGU Var s t  $\implies$  wf_MGU Var s t"
using wf_subst_nil subst_domain_Var range_vars_Var
unfolding wf_MGU_def by fast

lemma wf_MGU_dom_bound: "wf_MGU  $\vartheta$  s t  $\implies$  subst_domain  $\vartheta \subseteq$  fv s  $\cup$  fv t" unfolding wf_MGU_def by
blast

lemma wf_subst_single:
  assumes "v  $\notin$  fv t" " $\sigma$  v = t" " $\bigwedge w. v \neq w \implies \sigma w = \text{Var } w$ "
  shows "wf_subst  $\sigma$ "
proof -
  have *: "subst_domain  $\sigma = \{v\}$ " by (metis subst_fv_dom_img_single(1)[OF assms])

  have "subst_domain  $\sigma \cap$  range_vars  $\sigma = \{\}$ "
  using * assms subst_fv_dom_img_single(2)
  by (metis inf_bot_left insert_disjoint(1))
  moreover have "finite (subst_domain  $\sigma$ )" using * by simp
  ultimately show ?thesis by (metis wf_subst_def)
qed

lemma wf_subst_reduction:
  "wf_subst s  $\implies$  wf_subst (rm_var v s)"
proof -
  assume "wf_subst s"
  moreover have "subst_domain (rm_var v s)  $\subseteq$  subst_domain s" by (auto simp add: subst_domain_def)
  moreover have "range_vars (rm_var v s)  $\subseteq$  range_vars s"
  unfolding range_vars_alt_def by (auto simp add: subst_domain_def)
  ultimately have "subst_domain (rm_var v s)  $\cap$  range_vars (rm_var v s) =  $\{\}$ "
  by (meson compl_le_compl_iff disjoint_eq_subset_Compl subset_trans wf_subst_def)
  moreover have "finite (subst_domain (rm_var v s))"
  using  $\langle$ subst_domain (rm_var v s)  $\subseteq$  subst_domain s  $\rangle$   $\langle$ wf_subst s  $\rangle$  rev_finite_subset
  unfolding wf_subst_def by blast
  ultimately show "wf_subst (rm_var v s)" by (metis wf_subst_def)
qed

lemma wf_subst_compose:
  assumes "wf_subst  $\vartheta_1$ " "wf_subst  $\vartheta_2$ "
  and "subst_domain  $\vartheta_1 \cap$  subst_domain  $\vartheta_2 = \{\}$ "
  and "subst_domain  $\vartheta_1 \cap$  range_vars  $\vartheta_2 = \{\}$ "
  shows "wf_subst ( $\vartheta_1 \circ_s \vartheta_2$ )"
using assms
proof (induction  $\vartheta_1$  rule: wf_subst_induct)
  case Empty thus ?case unfolding wf_subst_def by simp
next
  case (Insert  $\sigma_1$  v t)
  have "t  $\neq$  Var v" using Insert.hyps(4) by auto
  hence domlv_unfold: "subst_domain ( $\sigma_1(v := t)$ ) = insert v (subst_domain  $\sigma_1$ )"
  using subst_dom_update2 by metis
  hence doms_disj: "subst_domain  $\sigma_1 \cap$  subst_domain  $\vartheta_2 = \{\}$ "
  using Insert.prem(2) disjoint_insert(1) by blast
  moreover have dom_img_disj: "subst_domain  $\sigma_1 \cap$  range_vars  $\vartheta_2 = \{\}$ "
  using Insert.hyps(2) Insert.prem(3)
  by (fastforce simp add: subst_domain_def)
  ultimately have "wf_subst ( $\sigma_1 \circ_s \vartheta_2$ )" using Insert.IH[OF  $\langle$ wf_subst  $\vartheta_2$  $\rangle$ ] by metis

  have dom_comp_is_union: "subst_domain ( $\sigma_1 \circ_s \vartheta_2$ ) = subst_domain  $\sigma_1 \cup$  subst_domain  $\vartheta_2$ "
  using subst_dom_comp_eq[OF dom_img_disj] .

  have "v  $\notin$  subst_domain  $\vartheta_2$ "
  using Insert.prem(2)  $\langle$ t  $\neq$  Var v $\rangle$ 
  by (fastforce simp add: subst_domain_def)
  hence " $\vartheta_2$  v = Var v" " $\sigma_1$  v = Var v" using Insert.hyps(2) by (simp_all add: subst_domain_def)
  hence " $(\sigma_1 \circ_s \vartheta_2)$  v = Var v" " $(\sigma_1(v := t) \circ_s \vartheta_2)$  v = t  $\cdot \vartheta_2$ " " $((\sigma_1 \circ_s \vartheta_2)(v := t))$  v = t"

```

```

unfolding subst_compose_def by simp_all

have fv_t2_bound: "fv (t ·  $\vartheta$ 2)  $\subseteq$  fv t  $\cup$  range_vars  $\vartheta$ 2" by (meson subst_sends_fv_to_img)

have 1: "v  $\notin$  subst_domain ( $\sigma$ 1  $\circ_s$   $\vartheta$ 2)"
  using  $\langle (\sigma$ 1  $\circ_s$   $\vartheta$ 2) v = Var v  $\rangle$ 
  by (auto simp add: subst_domain_def)

have "insert v (subst_domain  $\sigma$ 1)  $\cap$  range_vars  $\vartheta$ 2 = {}"
  using Insert.premis(3) domlv_unfold by blast
hence "v  $\notin$  range_vars  $\sigma$ 1  $\cup$  range_vars  $\vartheta$ 2" using Insert.hyps(3) by blast
hence 2: "v  $\notin$  range_vars ( $\sigma$ 1  $\circ_s$   $\vartheta$ 2)" by (meson set_rev_mp subst_img_comp_subset)

have "subst_domain  $\vartheta$ 2  $\cap$  range_vars  $\vartheta$ 2 = {}"
  using  $\langle wf_{subst}$   $\vartheta$ 2  $\rangle$  unfolding wf_subst_def by simp
hence "fv (t ·  $\vartheta$ 2)  $\cap$  subst_domain  $\vartheta$ 2 = {}"
  using subst_dom_elim unfolding range_vars_alt_def by simp
moreover have "v  $\notin$  range_vars  $\vartheta$ 2" using Insert.premis(3) domlv_unfold by blast
hence "v  $\notin$  fv t  $\cup$  range_vars  $\vartheta$ 2" using Insert.hyps(4) by blast
hence "v  $\notin$  fv (t ·  $\vartheta$ 2)" using  $\langle fv$  (t ·  $\vartheta$ 2)  $\subseteq$  fv t  $\cup$  range_vars  $\vartheta$ 2  $\rangle$  by blast
moreover have "fv (t ·  $\vartheta$ 2)  $\cap$  subst_domain  $\sigma$ 1 = {}"
  using dom_img_disj fv_t2_bound  $\langle fv$  t  $\cap$  insert v (subst_domain  $\sigma$ 1) = {}  $\rangle$  by blast
ultimately have 3: "fv (t ·  $\vartheta$ 2)  $\cap$  insert v (subst_domain ( $\sigma$ 1  $\circ_s$   $\vartheta$ 2)) = {}"
  using dom_comp_is_union by blast

have " $\sigma$ 1(v := t)  $\circ_s$   $\vartheta$ 2 = ( $\sigma$ 1  $\circ_s$   $\vartheta$ 2)(v := t ·  $\vartheta$ 2)" using subst_comp_upd1[of  $\sigma$ 1 v t  $\vartheta$ 2] .
moreover have "wf_subst (( $\sigma$ 1  $\circ_s$   $\vartheta$ 2)(v := t ·  $\vartheta$ 2))"
  using "wf_subst_set.Insert"[OF _ 1 2 3]  $\langle wf_{subst}$  ( $\sigma$ 1  $\circ_s$   $\vartheta$ 2)  $\rangle$  wf_subst_properties by metis
ultimately show ?case by presburger
qed

lemma wf_subst_append:
  fixes  $\vartheta$ 1  $\vartheta$ 2:: "('f, 'v) subst"
  assumes "wf_subst  $\vartheta$ 1" "wf_subst  $\vartheta$ 2"
    and "subst_domain  $\vartheta$ 1  $\cap$  subst_domain  $\vartheta$ 2 = {}"
    and "subst_domain  $\vartheta$ 1  $\cap$  range_vars  $\vartheta$ 2 = {}"
    and "range_vars  $\vartheta$ 1  $\cap$  subst_domain  $\vartheta$ 2 = {}"
  shows "wf_subst ( $\lambda v. \text{if } \vartheta$ 1 v = Var v then  $\vartheta$ 2 v else  $\vartheta$ 1 v)"
using assms
proof (induction  $\vartheta$ 1 rule: wf_subst_induct)
  case Empty thus ?case unfolding wf_subst_def by simp
next
  case (Insert  $\sigma$ 1 v t)
  let ?if = " $\lambda w. \text{if } \sigma$ 1 w = Var w then  $\vartheta$ 2 w else  $\sigma$ 1 w"
  let ?if_upd = " $\lambda w. \text{if } (\sigma$ 1(v := t)) w = Var w then  $\vartheta$ 2 w else ( $\sigma$ 1(v := t)) w"

  from Insert.hyps(4) have "?if_upd = ?if(v := t)" by fastforce

  have dom_insert: "subst_domain ( $\sigma$ 1(v := t)) = insert v (subst_domain  $\sigma$ 1)"
    using Insert.hyps(4) by (auto simp add: subst_domain_def)

  have " $\sigma$ 1 v = Var v" "t  $\neq$  Var v" using Insert.hyps(2,4) by auto
  hence img_insert: "range_vars ( $\sigma$ 1(v := t)) = range_vars  $\sigma$ 1  $\cup$  fv t"
    using subst_img_update by metis

  from Insert.premis(2) dom_insert have "subst_domain  $\sigma$ 1  $\cap$  subst_domain  $\vartheta$ 2 = {}"
    by (auto simp add: subst_domain_def)
  moreover have "subst_domain  $\sigma$ 1  $\cap$  range_vars  $\vartheta$ 2 = {}"
    using Insert.premis(3) dom_insert
    by (simp add: subst_domain_def)
  moreover have "range_vars  $\sigma$ 1  $\cap$  subst_domain  $\vartheta$ 2 = {}"
    using Insert.premis(4) img_insert
    by blast

```



```

ultimately have "wf_subst ?if" using Insert.IH[OF Insert.prem(1)] by metis

have dom_union: "subst_domain ?if = subst_domain  $\sigma_1 \cup$  subst_domain  $\vartheta_2$ "
  by (auto simp add: subst_domain_def)
hence "v  $\notin$  subst_domain ?if"
  using Insert.hyps(2) Insert.prem(2) dom_insert
  by (auto simp add: subst_domain_def)
moreover have "v  $\notin$  range_vars ?if"
  using Insert.prem(3) Insert.hyps(3) dom_insert
  unfolding range_vars_alt_def by (auto simp add: subst_domain_def)
moreover have "fv t  $\cap$  insert v (subst_domain ?if) = {}"
  using Insert.hyps(4) Insert.prem(4) img_insert
  unfolding range_vars_alt_def by (fastforce simp add: subst_domain_def)
ultimately show ?case
  using wf_subst_set.Insert (wf_subst ?if) (?if_upd = ?if(v := t)) wf_subst_properties
  by (metis (no_types, lifting))
qed

lemma wf_subst_elim_append:
  assumes "wf_subst  $\vartheta$ " "subst_elim  $\vartheta$  v" "v  $\notin$  fv t"
  shows "subst_elim ( $\vartheta(w := t)$ ) v"
using assms
proof (induction  $\vartheta$  rule: wf_subst_induct)
  case (Insert  $\vartheta$  v' t')
  hence " $\bigwedge q. v \notin \text{fv}(\text{Var } q \cdot \vartheta(v' := t'))$ " using subst_elimD by blast
  hence " $\bigwedge q. v \notin \text{fv}(\text{Var } q \cdot \vartheta(v' := t', w := t))$ " using (v  $\notin$  fv t) by simp
  thus ?case by (metis subst_elimI' subst_apply_term.simps(1))
qed (simp add: subst_elim_def)

lemma wf_subst_elim_dom:
  assumes "wf_subst  $\vartheta$ "
  shows " $\forall v \in \text{subst\_domain } \vartheta. \text{subst\_elim } \vartheta v$ "
using assms
proof (induction  $\vartheta$  rule: wf_subst_induct)
  case (Insert  $\vartheta$  w t)
  have dom_insert: "subst_domain ( $\vartheta(w := t)$ )  $\subseteq$  insert w (subst_domain  $\vartheta$ )"
    by (auto simp add: subst_domain_def)
  hence " $\forall v \in \text{subst\_domain } \vartheta. \text{subst\_elim } (\vartheta(w := t)) v$ " using Insert.IH Insert.hyps(2,4)
    by (metis Insert.hyps(1) IntI disjoint_insert(2) empty_iff wf_subst_elim_append)
  moreover have "w  $\notin$  fv t" using Insert.hyps(4) by simp
  hence " $\bigwedge q. w \notin \text{fv}(\text{Var } q \cdot \vartheta(w := t))$ "
    by (metis fv_simps(1) fv_in_subst_img Insert.hyps(3) contra_subsetD
      fun_upd_def singletonD subst_apply_term.simps(1))
  hence "subst_elim ( $\vartheta(w := t)$ ) w" by (metis subst_elimI')
  ultimately show ?case using dom_insert by blast
qed simp

lemma wf_subst_support_iff_mgt: "wf_subst  $\vartheta \implies \vartheta$  supports  $\delta \iff \vartheta \preceq_o \delta$ "
using subst_support_def subst_support_if_mgt_subst_idem wf_subst_subst_idem by blast

```

### 2.3.7 Interpretations

abbreviation `interpretation_subst::('a,'b) subst  $\Rightarrow$  bool` where  
`"interpretation_subst  $\vartheta \equiv \text{subst\_domain } \vartheta = \text{UNIV} \wedge \text{ground } (\text{subst\_range } \vartheta)$ "`

```

lemma interpretation_substI:
  " $(\bigwedge v. \text{fv } (\vartheta v) = \{\}) \implies \text{interpretation\_subst } \vartheta$ "
proof -
  assume " $\bigwedge v. \text{fv } (\vartheta v) = \{\}$ "
  moreover { fix v assume "fv ( $\vartheta v$ ) = {}" hence "v  $\in$  subst_domain  $\vartheta$ " by auto }
  ultimately show ?thesis by auto
qed

```

```

lemma interpretation_grounds[simp]:
  "interpretationsubst  $\vartheta \implies \text{fv } (t \cdot \vartheta) = \{\}$ "
using subst_fv_dom_ground_if_ground_img[of t  $\vartheta$ ] by blast

lemma interpretation_grounds_all:
  "interpretationsubst  $\vartheta \implies (\bigwedge v. \text{fv } (\vartheta v) = \{\})"$ 
by (metis range_vars_alt_def UNIV_I fv_in_subst_img subset_empty subst_dom_vars_in_subst)

lemma interpretation_grounds_all':
  "interpretationsubst  $\vartheta \implies \text{ground } (M \cdot_{\text{set}} \vartheta)"$ 
using subst_fv_dom_ground_if_ground_img[of _  $\vartheta$ ]
by simp

lemma interpretation_comp:
  assumes "interpretationsubst  $\vartheta"$ 
  shows "interpretationsubst  $(\sigma \circ_s \vartheta)"$  "interpretationsubst  $(\vartheta \circ_s \sigma)"$ 
proof -
  have  $\vartheta_{\text{fv}}$ : " $\text{fv } (\vartheta v) = \{\}$ " for  $v$  using interpretation_grounds_all[OF assms] by simp
  hence  $\vartheta_{\text{fv}}$ : " $\text{fv } (t \cdot \vartheta) = \{\}$ " for  $t$ 
    by (metis all_not_in_conv subst_elimD subst_elimI' subst_apply_term.simps(1))

  from assms have " $(\sigma \circ_s \vartheta) v \neq \text{Var } v$ " for  $v$ 
    unfolding subst_compose_def by (metis fv_simps(1)  $\vartheta_{\text{fv}}$  insert_not_empty)
  hence " $\text{subst\_domain } (\sigma \circ_s \vartheta) = \text{UNIV}$ " by (simp add: subst_domain_def)
  moreover have " $\text{fv } ((\sigma \circ_s \vartheta) v) = \{\}$ " for  $v$  unfolding subst_compose_def using  $\vartheta_{\text{fv}}$  by simp
  hence " $\text{ground } (\text{subst\_range } (\sigma \circ_s \vartheta))"$  by simp
  ultimately show "interpretationsubst  $(\sigma \circ_s \vartheta)"$  ..

  from assms have " $(\vartheta \circ_s \sigma) v \neq \text{Var } v$ " for  $v$ 
    unfolding subst_compose_def by (metis fv_simps(1)  $\vartheta_{\text{fv}}$  insert_not_empty subst_to_var_is_var)
  hence " $\text{subst\_domain } (\vartheta \circ_s \sigma) = \text{UNIV}$ " by (simp add: subst_domain_def)
  moreover have " $\text{fv } ((\vartheta \circ_s \sigma) v) = \{\}$ " for  $v$ 
    unfolding subst_compose_def by (simp add:  $\vartheta_{\text{fv}}$  trm_subst_ident)
  hence " $\text{ground } (\text{subst\_range } (\vartheta \circ_s \sigma))"$  by simp
  ultimately show "interpretationsubst  $(\vartheta \circ_s \sigma)"$  ..
qed

lemma interpretation_subst_exists:
  " $\exists \mathcal{I}::('f, 'v) \text{subst. interpretation}_{\text{subst}} \mathcal{I}$ "
proof -
  obtain  $c::''f''$  where " $c \in \text{UNIV}$ " by simp
  then obtain  $\mathcal{I}::('f, 'v) \text{subst}$  where " $\bigwedge v. \mathcal{I} v = \text{Fun } c []$ " by simp
  hence " $\text{subst\_domain } \mathcal{I} = \text{UNIV}$ " "ground  $(\text{subst\_range } \mathcal{I})"$ 
    by (simp_all add: subst_domain_def)
  thus ?thesis by auto
qed

lemma interpretation_subst_exists':
  " $\exists \vartheta::('f, 'v) \text{subst. subst\_domain } \vartheta = X \wedge \text{ground } (\text{subst\_range } \vartheta)"$ 
proof -
  obtain  $\mathcal{I}::('f, 'v) \text{subst}$  where  $\mathcal{I}$ : " $\text{subst\_domain } \mathcal{I} = \text{UNIV}$ " "ground  $(\text{subst\_range } \mathcal{I})"$ 
    using interpretation_subst_exists by mouna
  let  $?\vartheta = \text{"rm\_vars } (\text{UNIV} - X) \mathcal{I}"$ 
  have 1: " $\text{subst\_domain } ?\vartheta = X$ " using  $\mathcal{I}$  by (auto simp add: subst_domain_def)
  hence 2: " $\text{ground } (\text{subst\_range } ?\vartheta)"$  using  $\mathcal{I}$  by force
  show ?thesis using 1 2 by blast
qed

lemma interpretation_subst_idem:
  "interpretationsubst  $\vartheta \implies \text{subst\_idem } \vartheta"$ 
unfolding subst_idem_def
using interpretation_grounds_all[of  $\vartheta$ ] trm_subst_ident subst_eq_if_eq_vars
by fastforce

```

```

lemma subst_idem_comp_upd_eq:
  assumes "v ∉ subst_domain I" "subst_idem ∅"
  shows "I ∘s ∅ = I(v := ∅ v) ∘s ∅"
proof -
  from assms(1) have "(I ∘s ∅) v = ∅ v" unfolding subst_compose_def by auto
  moreover have "∧w. w ≠ v ⟹ (I ∘s ∅) w = (I(v := ∅ v) ∘s ∅) w" unfolding subst_compose_def by
  auto
  moreover have "(I(v := ∅ v) ∘s ∅) v = ∅ v" using assms(2) unfolding subst_idem_def
  subst_compose_def
  by (metis fun_upd_same)
  ultimately show ?thesis by (metis fun_upd_same fun_upd_triv subst_comp_upd1)
qed

```

```

lemma interpretation_dom_img_disjoint:
  "interpretationsubst I ⟹ subst_domain I ∩ range_vars I = {}"
unfolding range_vars_alt_def by auto

```

### 2.3.8 Basic Properties of MGUs

```

lemma MGU_is_mgu_singleton: "MGU ∅ t u = is_mgu ∅ {(t,u)}"
unfolding is_mgu_def unifiers_def by auto

```

```

lemma Unifier_in_unifiers_singleton: "Unifier ∅ s t ⟷ ∅ ∈ unifiers {(s,t)}"
unfolding unifiers_def by auto

```

```

lemma subst_list_singleton_fv_subset:
  "(⋃ x ∈ set (subst_list (subst v t) E). fv (fst x) ∪ fv (snd x))
   ⊆ fv t ∪ (⋃ x ∈ set E. fv (fst x) ∪ fv (snd x))"
proof (induction E)
  case (Cons x E)
  let ?fvs = "λL. ⋃ x ∈ set L. fv (fst x) ∪ fv (snd x)"
  let ?fvx = "fv (fst x) ∪ fv (snd x)"
  let ?fvxsubst = "fv (fst x · Var(v := t)) ∪ fv (snd x · Var(v := t))"
  have "?fvs (subst_list (subst v t) (x#E)) = ?fvxsubst ∪ ?fvs (subst_list (subst v t) E)"
  unfolding subst_list_def subst_def by auto
  hence "?fvs (subst_list (subst v t) (x#E)) ⊆ ?fvxsubst ∪ fv t ∪ ?fvs E"
  using Cons.IH by blast
  moreover have "?fvs (x#E) = ?fvx ∪ ?fvs E" by auto
  moreover have "?fvxsubst ⊆ ?fvx ∪ fv t" using subst_fv_bound_singleton[of _ v t] by blast
  ultimately show ?case unfolding range_vars_alt_def by auto
qed (simp add: subst_list_def)

```

```

lemma subst_of_dom_subset: "subst_domain (subst_of L) ⊆ set (map fst L)"
proof (induction L rule: List.rev_induct)
  case (snoc x L)
  then obtain v t where x: "x = (v,t)" by (metis surj_pair)
  hence "subst_of (L@[x]) = Var(v := t) ∘s subst_of L"
  unfolding subst_of_def subst_def by (induct L) auto
  hence "subst_domain (subst_of (L@[x])) ⊆ insert v (subst_domain (subst_of L))"
  using x subst_domain_compose[of "Var(v := t)" "subst_of L"]
  by (auto simp add: subst_domain_def)
  thus ?case using snoc.IH x by auto
qed simp

```

```

lemma wf_MGU_is_imgu_singleton: "wf_MGU ∅ s t ⟹ is_imgu ∅ {(s,t)}"
proof -
  assume 1: "wf_MGU ∅ s t"

  have 2: "subst_idem ∅" by (metis wf_subst_subst_idem 1 wf_MGU_def)

  have 3: "∀∅' ∈ unifiers {(s,t)}. ∅ ≼o ∅'" "∅ ∈ unifiers {(s,t)}"
  by (metis 1 Unifier_in_unifiers_singleton wf_MGU_def)+

```

```

have "∀τ ∈ unifiers {(s,t)}. τ = ϑ ∘s τ" by (metis 2 3 subst_idem_def subst_compose_assoc)
thus "is_imgu ϑ {(s,t)}" by (metis is_imgu_def ⟨ϑ ∈ unifiers {(s,t)}⟩)
qed

```

```

lemma mgu_subst_range_vars:
  assumes "mgu s t = Some σ" shows "range_vars σ ⊆ vars_term s ∪ vars_term t"
proof -
  obtain xs where *: "Unification.unify [(s, t)] [] = Some xs" and [simp]: "subst_of xs = σ"
  using assms by (simp split: option.splits)
  from unify_Some_UNIF [OF *] obtain ss
  where "compose ss = σ" and "UNIF ss {#(s, t)#} {#}" by auto
  with UNIF_range_vars_subset [of ss "{#(s, t)#}" "{#}"]
  show ?thesis by (metis vars_mset_singleton fst_conv snd_conv)
qed

```

```

lemma mgu_subst_domain_range_vars_disjoint:
  assumes "mgu s t = Some σ" shows "subst_domain σ ∩ range_vars σ = {}"
proof -
  have "is_imgu σ {(s, t)}" using assms mgu_sound by simp
  hence "σ = σ ∘s σ" unfolding is_imgu_def by blast
  thus ?thesis by (metis subst_idemp_iff)
qed

```

```

lemma mgu_same_empty: "mgu (t::('a,'b) term) t = Some Var"
proof -
  { fix E::"('a,'b) equation list" and U::"('b × ('a,'b) term) list"
    assume "∀(s,t) ∈ set E. s = t"
    hence "Unification.unify E U = Some U"
    proof (induction E U rule: Unification.unify.induct)
      case (2 f S g T E U)
      hence *: "f = g" "S = T" by auto
      moreover have "∀(s,t) ∈ set (zip T T). s = t" by (induct T) auto
      hence "∀(s,t) ∈ set (zip T T@E). s = t" using "2.prem1" by auto
      moreover have "zip_option S T = Some (zip S T)" using ⟨S = T⟩ by auto
      hence **: "decompose (Fun f S) (Fun g T) = Some (zip S T)"
        using ⟨f = g⟩ unfolding decompose_def by auto
      ultimately have "Unification.unify (zip S T@E) U = Some U" using "2.IH" * by auto
      thus ?case using ** by auto
    qed auto
  }
  hence "Unification.unify [(t,t)] [] = Some []" by auto
  thus ?thesis by auto
qed

```

```

lemma mgu_var: assumes "x ∉ fv t" shows "mgu (Var x) t = Some (Var(x := t))"
proof -
  have "unify [(Var x,t)] [] = Some [(x,t)]" using assms by (auto simp add: subst_list_def)
  moreover have "subst_of [(x,t)] = Var(x := t)" unfolding subst_of_def subst_def by simp
  ultimately show ?thesis by simp
qed

```

```

lemma mgu_gives_wellformed_subst:
  assumes "mgu s t = Some ϑ" shows "wf_subst ϑ"
using mgu_finite_subst_domain[OF assms] mgu_subst_domain_range_vars_disjoint[OF assms]
unfolding wf_subst_def
by auto

```

```

lemma mgu_gives_wellformed_MGU:
  assumes "mgu s t = Some ϑ" shows "wf_MGU ϑ s t"
using mgu_subst_domain[OF assms] mgu_sound[OF assms] mgu_subst_range_vars [OF assms]
MGU_is_mgu_singleton[of s ϑ t] is_imgu_imp_is_mgu[of ϑ "{(s,t)}"]
mgu_gives_wellformed_subst[OF assms]

```

```

unfolding wf_MGU_def by blast

lemma mgu_vars_bounded[dest?]:
  "mgu M N = Some  $\sigma \implies \text{subst\_domain } \sigma \cup \text{range\_vars } \sigma \subseteq \text{fv } M \cup \text{fv } N"$ 
using mgu_gives_wellformed_MGU unfolding wf_MGU_def by blast

lemma mgu_gives_subst_idem: "mgu s t = Some  $\vartheta \implies \text{subst\_idem } \vartheta"$ 
using mgu_sound[of s t  $\vartheta$ ] unfolding is_imgu_def subst_idem_def by auto

lemma mgu_always_unifies: "Unifier  $\vartheta$  M N  $\implies \exists \delta. \text{mgu } M N = \text{Some } \delta"$ 
using mgu_complete Unifier_in_unifiers_singleton by blast

lemma mgu_gives_MGU: "mgu s t = Some  $\vartheta \implies \text{MGU } \vartheta$  s t"
using mgu_sound[of s t  $\vartheta$ , THEN is_imgu_imp_is_mgu] MGU_is_mgu_singleton by metis

lemma mgu_elimimates[dest?]:
  assumes "mgu M N = Some  $\sigma$ "
  shows "( $\exists v \in \text{fv } M \cup \text{fv } N. \text{subst\_elim } \sigma v$ )  $\vee \sigma = \text{Var}$ "
  (is "?P M N  $\sigma$ ")
proof (cases " $\sigma = \text{Var}$ ")
  case False
  then obtain v where v: " $v \in \text{subst\_domain } \sigma$ " by auto
  hence " $v \in \text{fv } M \cup \text{fv } N$ " using mgu_vars_bounded[OF assms] by blast
  thus ?thesis using wf_subst_elim_dom[OF mgu_gives_wellformed_subst[OF assms]] v by blast
qed simp

lemma mgu_elimimates_dom:
  assumes "mgu x y = Some  $\vartheta$ " " $v \in \text{subst\_domain } \vartheta$ "
  shows "subst_elim  $\vartheta$  v"
using mgu_gives_wellformed_subst[OF assms(1)]
unfolding wf_MGU_def wf_subst_def subst_elim_def
by (metis disjoint_iff_not_equal subst_dom_elim assms(2))

lemma unify_list_distinct:
  assumes "Unification.unify E B = Some U" "distinct (map fst B)"
  and "( $\bigcup x \in \text{set } E. \text{fv } (\text{fst } x) \cup \text{fv } (\text{snd } x)$ )  $\cap \text{set } (\text{map fst } B) = \{\}$ "
  shows "distinct (map fst U)"
using assms
proof (induction E B arbitrary: U rule: Unification.unify.induct)
  case 1 thus ?case by simp
next
  case (2 f X g Y E B U)
  let ?fvs = " $\lambda L. \bigcup x \in \text{set } L. \text{fv } (\text{fst } x) \cup \text{fv } (\text{snd } x)$ "
  from "2.prem"(1) obtain E' where *: "decompose (Fun f X) (Fun g Y) = Some E'"
  and [simp]: "f = g" "length X = length Y" "E' = zip X Y"
  and **: "Unification.unify (E'@E) B = Some U"
  by (auto split: option.splits)
  hence " $\bigwedge t t'. (t, t') \in \text{set } E' \implies \text{fv } t \subseteq \text{fv } (\text{Fun } f X) \wedge \text{fv } t' \subseteq \text{fv } (\text{Fun } g Y)$ "
  by (metis zip_arg_subterm subterm_eq_vars_subset)
  hence "?fvs E'  $\subseteq \text{fv } (\text{Fun } f X) \cup \text{fv } (\text{Fun } g Y)$ " by fastforce
  moreover have " $\text{fv } (\text{Fun } f X) \cap \text{set } (\text{map fst } B) = \{\}$ " " $\text{fv } (\text{Fun } g Y) \cap \text{set } (\text{map fst } B) = \{\}$ "
  using "2.prem"(3) by auto
  ultimately have "?fvs E'  $\cap \text{set } (\text{map fst } B) = \{\}$ " by blast
  moreover have "?fvs E  $\cap \text{set } (\text{map fst } B) = \{\}$ " using "2.prem"(3) by auto
  ultimately have "?fvs (E'@E)  $\cap \text{set } (\text{map fst } B) = \{\}$ " by auto
  thus ?case using "2.IH"[OF * ** "2.prem"(2)] by metis
next
  case (3 v t E B)
  let ?fvs = " $\lambda L. \bigcup x \in \text{set } L. \text{fv } (\text{fst } x) \cup \text{fv } (\text{snd } x)$ "
  let ?E' = "subst_list (subst v t) E"
  from "3.prem"(3) have "v  $\notin \text{set } (\text{map fst } B)$ " "fv t  $\cap \text{set } (\text{map fst } B) = \{\}$ " by force+
  hence *: "distinct (map fst ((v, t)#B))" using "3.prem"(2) by auto

```

```

show ?case
proof (cases "t = Var v")
  case True thus ?thesis using "3.prem" "3.IH"(1) by auto
next
  case False
  hence "v ∉ fv t" using "3.prem"(1) by auto
  hence "Unification.unify (subst_list (subst v t) E) ((v, t)#B) = Some U"
    using "t ≠ Var v" "3.prem"(1) by auto
  moreover have "?fvs ?E' ∩ set (map fst ((v, t)#B)) = {}"
  proof -
    have "v ∉ ?fvs ?E'"
      unfolding subst_list_def subst_def
      by (simp add: ⟨v ∉ fv t⟩ subst_remove_var)
    moreover have "?fvs ?E' ⊆ fv t ∪ ?fvs E" by (metis subst_list_singleton_fv_subset)
    hence "?fvs ?E' ∩ set (map fst B) = {}" using "3.prem"(3) by auto
    ultimately show ?thesis by auto
  qed
  ultimately show ?thesis using "3.IH"(2)[OF ⟨t ≠ Var v⟩ ⟨v ∉ fv t⟩ _ *] by metis
qed
next
  case (4 f X v E B U)
  let ?fvs = "λL. ⋃ x ∈ set L. fv (fst x) ∪ fv (snd x)"
  let ?E' = "subst_list (subst v (Fun f X)) E"
  have *: "?fvs E ∩ set (map fst B) = {}" using "4.prem"(3) by auto
  from "4.prem"(1) have "v ∉ fv (Fun f X)" by force
  from "4.prem"(3) have **: "v ∉ set (map fst B)" "fv (Fun f X) ∩ set (map fst B) = {}" by force+
  hence ***: "distinct (map fst ((v, Fun f X)#B))" using "4.prem"(2) by auto
  from "4.prem"(3) have ****: "?fvs ?E' ∩ set (map fst ((v, Fun f X)#B)) = {}"
  proof -
    have "v ∉ ?fvs ?E'"
      unfolding subst_list_def subst_def
      using ⟨v ∉ fv (Fun f X)⟩ subst_remove_var[of v "Fun f X"] by simp
    moreover have "?fvs ?E' ⊆ fv (Fun f X) ∪ ?fvs E" by (metis subst_list_singleton_fv_subset)
    hence "?fvs ?E' ∩ set (map fst B) = {}" using * ** by blast
    ultimately show ?thesis by auto
  qed
  have "Unification.unify (subst_list (subst v (Fun f X)) E) ((v, Fun f X) # B) = Some U"
    using ⟨v ∉ fv (Fun f X)⟩ "4.prem"(1) by auto
  thus ?case using "4.IH"[OF ⟨v ∉ fv (Fun f X)⟩ _ *** ****] by metis
qed

lemma mgu_None_is_subst_neq:
  fixes s t :: "('a, 'b) term" and δ :: "('a, 'b) subst"
  assumes "mgu s t = None"
  shows "s · δ ≠ t · δ"
using assms mgu_always_unifies by force

lemma mgu_None_if_neq_ground:
  assumes "t ≠ t'" "fv t = {}" "fv t' = {}"
  shows "mgu t t' = None"
proof (rule ccontr)
  assume "mgu t t' ≠ None"
  then obtain δ where δ: "mgu t t' = Some δ" by auto
  hence "t · δ = t" "t' · δ = t'" using assms subst_ground_ident by auto
  thus False using assms(1) MGU_is_Unifier[OF mgu_gives_MGU[OF δ]] by auto
qed

lemma mgu_None_commutes:
  "mgu s t = None ⟹ mgu t s = None"
using mgu_complete[of s t]
  Unifier_in_unifiers_singleton[of s _ t]
  Unifier_sym[of t _ s]
  Unifier_in_unifiers_singleton[of t _ s]

```

```

    mgu_sound[of t s]
unfolding is_imgu_def
by fastforce

lemma mgu_img_subterm_subst:
  fixes  $\delta :: ('f, 'v) \text{subst}$  and  $s \ t \ u :: ('f, 'v) \text{term}$ 
  assumes "mgu s t = Some  $\delta$ " "u  $\in$  subtermsset (subst_range  $\delta$ ) - range Var"
  shows "u  $\in$  ((subterms s  $\cup$  subterms t) - range Var)  $\cdot_{\text{set}}$   $\delta$ "
proof -
  define subterms_tuples :: "('f, 'v) equation list  $\Rightarrow$  ('f, 'v) terms" where subtt_def:
    "subterms_tuples  $\equiv$   $\lambda E$ . subtermsset (fst ' set E)  $\cup$  subtermsset (snd ' set E)"
  define subterms_img :: "('f, 'v) subst  $\Rightarrow$  ('f, 'v) terms" where subti_def:
    "subterms_img  $\equiv$   $\lambda d$ . subtermsset (subst_range d)"

  define d where "d  $\equiv$   $\lambda v \ t$ . subst v t :: ('f, 'v) subst"
  define V where "V  $\equiv$  range Var :: ('f, 'v) terms"
  define R where "R  $\equiv$   $\lambda d :: ('f, 'v) \text{subst}$ . ((subterms s  $\cup$  subterms t) - V)  $\cdot_{\text{set}}$  d"
  define M where "M  $\equiv$   $\lambda E \ d$ . subterms_tuples E  $\cup$  subterms_img d"
  define Q where "Q  $\equiv$  ( $\lambda E \ d$ . M E d - V  $\subseteq$  R d - V)"
  define Q' where "Q'  $\equiv$  ( $\lambda E \ d \ d'$ . (M E d - V)  $\cdot_{\text{set}}$  d'  $\subseteq$  (R d - V)  $\cdot_{\text{set}}$  (d' :: ('f, 'v) subst))"

  have Q_subst: "Q (subst_list (subst v t') E) (subst_of ((v, t')#B))"
    when v_fv: "v  $\notin$  fv t'" and Q_assm: "Q ((Var v, t')#E) (subst_of B)"
    for v t' E B
  proof -
    define E' where "E'  $\equiv$  subst_list (subst v t') E"
    define B' where "B'  $\equiv$  subst_of ((v, t')#B)"

    have E': "E' = subst_list (d v t') E"
      and B': "B' = subst_of B  $\circ_s$  d v t'"
      using subst_of_simps(3)[of "(v, t')"]
      unfolding subst_def E'_def B'_def d_def by simp_all

    have vt_img_subst: "subtermsset (subst_range (d v t')) = subterms t'"
      and vt_dom: "subst_domain (d v t') = {v}"
      using v_fv by (auto simp add: subst_domain_def d_def subst_def)

    have *: "subterms u1  $\subseteq$  subtermsset (fst ' set E)" "subterms u2  $\subseteq$  subtermsset (snd ' set E)"
      when "(u1,u2)  $\in$  set E" for u1 u2
      using that by auto

    have **: "subtermsset (d v t' ' (fv u  $\cap$  subst_domain (d v t'))))  $\subseteq$  subterms t'"
      for u :: "('f, 'v) term"
      using vt_dom unfolding d_def by force

    have 1: "subterms_tuples E' - V  $\subseteq$  (subterms t' - V)  $\cup$  (subterms_tuples E - V  $\cdot_{\text{set}}$  d v t')"
      (is "?A  $\subseteq$  ?B")
    proof
      fix u assume "u  $\in$  ?A"
      then obtain u1 u2 where u12:
        "(u1,u2)  $\in$  set E"
        "u  $\in$  (subterms (u1  $\cdot$  (d v t')) - V)  $\cup$  (subterms (u2  $\cdot$  (d v t')) - V)"
        unfolding subtt_def subst_list_def E'_def d_def by moura
      hence "u  $\in$  (subterms t' - V)  $\cup$  ((subterms_tuples E)  $\cdot_{\text{set}}$  d v t' - V)"
        using subterms_subst[of u1 "d v t'"] subterms_subst[of u2 "d v t'"]
        *[OF u12(1)] **[of u1] **[of u2]
        unfolding subtt_def subst_list_def by auto
      moreover have
        "(subterms_tuples E  $\cdot_{\text{set}}$  d v t') - V  $\subseteq$ 
        (subterms_tuples E - V  $\cdot_{\text{set}}$  d v t')  $\cup$  {t'}"
        unfolding subst_def subtt_def V_def d_def by force
      ultimately show "u  $\in$  ?B" using u12 v_fv by auto
    qed
  qed

```

```

have 2: "subterms_img B' - V  $\subseteq$ 
  (subterms t' - V)  $\cup$  (subterms_img (subst_of B) - V  $\cdot_{set}$  d v t')"
  using B' vt_img_subt subst_img_comp_subset'' [of "subst_of B" "d v t'"]
  unfolding subti_def subst_def V_def by argo

have 3: "subterms_tuples ((Var v, t')#E) - V = (subterms t' - V)  $\cup$  (subterms_tuples E - V)"
  by (auto simp add: subst_def subtt_def V_def)

have "fvset (subterms t' - V)  $\cap$  subst_domain (d v t') = {}"
  using v_fv vt_dom fv_subterms [of t'] by fastforce
hence 4: "subterms t' - V  $\cdot_{set}$  d v t' = subterms t' - V"
  using set_subst_ident [of "subterms t' - range Var" "d v t'"] by (simp add: V_def)

have "M E' B' - V  $\subseteq$  M ((Var v, t')#E) (subst_of B) - V  $\cdot_{set}$  d v t'"
  using 1 2 3 4 unfolding M_def by blast
moreover have "Q' ((Var v, t')#E) (subst_of B) (d v t')"
  using Q_assm unfolding Q_def Q'_def by auto
moreover have "R (subst_of B)  $\cdot_{set}$  d v t' = R (subst_of ((v, t')#B))"
  unfolding R_def d_def by auto
ultimately have
  "M (subst_list (d v t') E) (subst_of ((v, t')#B)) - V  $\subseteq$  R (subst_of ((v, t')#B)) - V"
  unfolding Q'_def E'_def B'_def d_def by blast
thus ?thesis unfolding Q_def M_def R_def d_def by blast
qed

have "u  $\in$  subterms s  $\cup$  subterms t - V  $\cdot_{set}$  subst_of U"
  when assms':
    "unify E B = Some U"
    "u  $\in$  subtermsset (subst_range (subst_of U)) - V"
    "Q E (subst_of B)"
  for E B U and T::('f, 'v) term list
  using assms'
proof (induction E B arbitrary: U rule: Unification.unify.induct)
  case (1 B) thus ?case by (auto simp add: Q_def M_def R_def subti_def)
next
  case (2 g X h Y E B U)
  from "2.prem"(1) obtain E' where E':
    "decompose (Fun g X) (Fun h Y) = Some E'"
    "g = h" "length X = length Y" "E' = zip X Y"
    "Unification.unify (E'@E) B = Some U"
  by (auto split: option.splits)
  moreover have "subterms_tuples (E'@E)  $\subseteq$  subterms_tuples ((Fun g X, Fun h Y)#E)"
  proof
    fix u assume "u  $\in$  subterms_tuples (E'@E)"
    then obtain u1 u2 where u12: "(u1, u2)  $\in$  set (E'@E)" "u  $\in$  subterms u1  $\cup$  subterms u2"
    unfolding subtt_def by fastforce
    thus "u  $\in$  subterms_tuples ((Fun g X, Fun h Y)#E)"
    proof (cases "(u1, u2)  $\in$  set E'")
    case True
    hence "subterms u1  $\subseteq$  subterms (Fun g X)" "subterms u2  $\subseteq$  subterms (Fun h Y)"
      using E' (4) subterms_subset params_subterms subsetCE
      by (metis set_zip_leftD, metis set_zip_rightD)
    thus ?thesis using u12 unfolding subtt_def by auto
    next
    case False thus ?thesis using u12 unfolding subtt_def by fastforce
    qed
  qed
  hence "Q (E'@E) (subst_of B)" using "2.prem"(3) unfolding Q_def M_def by blast
  ultimately show ?case using "2.IH" [of E' U] "2.prem" by meson
next
  case (3 v t' E B)
  show ?case

```



```

proof (cases "t' = Var v")
  case True thus ?thesis
    using "3.prem" "3.IH"(1) unfolding Q_def M_def V_def subtt_def by auto
next
  case False
  hence 1: "v ∉ fv t'" using "3.prem"(1) by auto
  hence "unify (subst_list (subst v t')) E ((v, t')#B) = Some U"
    using False "3.prem"(1) by auto
  thus ?thesis
    using Q_subst[OF 1 "3.prem"(3)]
      "3.IH"(2)[OF False 1 _ "3.prem"(2)]
    by metis
qed
next
  case (4 g X v E B U)
  have 1: "v ∉ fv (Fun g X)" using "4.prem"(1) not_None_eq by fastforce
  hence 2: "unify (subst_list (subst v (Fun g X)) E) ((v, Fun g X)#B) = Some U"
    using "4.prem"(1) by auto

  have 3: "Q ((Var v, Fun g X)#E) (subst_of B)"
    using "4.prem"(3) unfolding Q_def M_def subtt_def by auto

  show ?case
    using Q_subst[OF 1 3] "4.IH"[OF 1 2 "4.prem"(2)]
    by metis
qed
moreover obtain D where "unify [(s, t)] [] = Some D" "δ = subst_of D"
  using assms(1) by (auto split: option.splits)
moreover have "Q [(s,t)] (subst_of [])"
  unfolding Q_def M_def R_def subtt_def subti_def
  by force
ultimately show ?thesis using assms(2) unfolding V_def by auto
qed

lemma mgu_img_consts:
  fixes δ::('f,'v) subst and s t::('f,'v) term and c::'f and z::'v
  assumes "mgu s t = Some δ" "Fun c [] ∈ subtermsset (subst_range δ)"
  shows "Fun c [] ∈ subterms s ∪ subterms t"
proof -
  obtain u where "u ∈ (subterms s ∪ subterms t) - range Var" "u · δ = Fun c []"
    using mgu_img_subterm_subst[OF assms(1), of "Fun c []"] assms(2) by force
  thus ?thesis by (cases u) auto
qed

lemma mgu_img_consts':
  fixes δ::('f,'v) subst and s t::('f,'v) term and c::'f and z::'v
  assumes "mgu s t = Some δ" "δ z = Fun c []"
  shows "Fun c [] ⊆ s ∨ Fun c [] ⊆ t"
using mgu_img_consts[OF assms(1)] assms(2)
by (metis Un_iff in_subterms_Union subst_imgI term.distinct(1))

lemma mgu_img_composed_var_term:
  fixes δ::('f,'v) subst and s t::('f,'v) term and f::'f and Z::'v list
  assumes "mgu s t = Some δ" "Fun f (map Var Z) ∈ subtermsset (subst_range δ)"
  shows "∃ Z'. map δ Z' = map Var Z ∧ Fun f (map Var Z') ∈ subterms s ∪ subterms t"
proof -
  obtain u where u: "u ∈ (subterms s ∪ subterms t) - range Var" "u · δ = Fun f (map Var Z)"
    using mgu_img_subterm_subst[OF assms(1), of "Fun f (map Var Z)"] assms(2) by fastforce
  then obtain T where T: "u = Fun f T" "map (λt. t · δ) T = map Var Z" by (cases u) auto
  have "∀ t ∈ set T. ∃ x. t = Var x" using T(2) by (induct T arbitrary: Z) auto
  then obtain Z' where Z': "map Var Z' = T" by (metis ex_map_conv)
  hence "map δ Z' = map Var Z" using T(2) by (induct Z' arbitrary: T Z) auto
  thus ?thesis using u(1) T(1) Z' by auto

```

qed

### 2.3.9 Lemmata: The "Inequality Lemmata"

Subterm injectivity (a stronger injectivity property)

definition subterm\_inj\_on where

"subterm\_inj\_on f A  $\equiv \forall x \in A. \forall y \in A. (\exists v. v \sqsubseteq f\ x \wedge v \sqsubseteq f\ y) \longrightarrow x = y$ "

lemma subterm\_inj\_on\_imp\_inj\_on: "subterm\_inj\_on f A  $\implies$  inj\_on f A"

unfolding subterm\_inj\_on\_def inj\_on\_def by fastforce

lemma subst\_inj\_on\_is\_bij\_betw:

"inj\_on  $\vartheta$  (subst\_domain  $\vartheta$ ) = bij\_betw  $\vartheta$  (subst\_domain  $\vartheta$ ) (subst\_range  $\vartheta$ )"

unfolding inj\_on\_def bij\_betw\_def by auto

lemma subterm\_inj\_on\_alt\_def:

"subterm\_inj\_on f A  $\longleftrightarrow$

(inj\_on f A  $\wedge (\forall s \in f'A. \forall u \in f'A. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u)$ )"

(is "?A  $\longleftrightarrow$  ?B")

unfolding subterm\_inj\_on\_def inj\_on\_def by fastforce

lemma subterm\_inj\_on\_alt\_def':

"subterm\_inj\_on  $\vartheta$  (subst\_domain  $\vartheta$ )  $\longleftrightarrow$

(inj\_on  $\vartheta$  (subst\_domain  $\vartheta$ )  $\wedge$

( $\forall s \in \text{subst\_range } \vartheta. \forall u \in \text{subst\_range } \vartheta. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u$ )"

(is "?A  $\longleftrightarrow$  ?B")

by (metis subterm\_inj\_on\_alt\_def subst\_range.simps)

lemma subterm\_inj\_on\_subset:

assumes "subterm\_inj\_on f A"

and "B  $\subseteq$  A"

shows "subterm\_inj\_on f B"

proof -

have "inj\_on f A" " $\forall s \in f' A. \forall u \in f' A. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u$ "

using subterm\_inj\_on\_alt\_def[of f A] assms(1) by auto

moreover have "f' B  $\subseteq$  f' A" using assms(2) by auto

ultimately have "inj\_on f B" " $\forall s \in f' B. \forall u \in f' B. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u$ "

using inj\_on\_subset[of f A] assms(2) by blast+

thus ?thesis by (metis subterm\_inj\_on\_alt\_def)

qed

lemma inj\_subst\_unif\_consts:

fixes  $\mathcal{I} \ \vartheta \ \sigma :: ('f, 'v) \text{subst}$  and  $s \ t :: ('f, 'v) \text{term}$

assumes  $\vartheta$ : "subterm\_inj\_on  $\vartheta$  (subst\_domain  $\vartheta$ )" " $\forall x \in (fv\ s \cup fv\ t) - X. \exists c. \vartheta\ x = \text{Fun } c\ []$ "

"subterms<sub>set</sub> (subst\_range  $\vartheta$ )  $\cap$  (subterms s  $\cup$  subterms t) = {}" "ground (subst\_range  $\vartheta$ )"

"subst\_domain  $\vartheta \cap X = \{\}$ "

and  $\mathcal{I}$ : "ground (subst\_range  $\mathcal{I}$ )" "subst\_domain  $\mathcal{I} = \text{subst\_domain } \vartheta$ "

and unif: "Unifier  $\sigma$  (s  $\cdot$   $\vartheta$ ) (t  $\cdot$   $\vartheta$ )"

shows " $\exists \delta. \text{Unifier } \delta$  (s  $\cdot$   $\mathcal{I}$ ) (t  $\cdot$   $\mathcal{I}$ )"

proof -

let ?xs = "subst\_domain  $\vartheta$ "

let ?ys = "(fv s  $\cup$  fv t) - ?xs"

have " $\exists \delta :: ('f, 'v) \text{subst}. s \cdot \delta = t \cdot \delta$ " by (metis subst\_subst\_compose unif)

then obtain  $\delta :: ('f, 'v) \text{subst}$  where  $\delta$ : "mgu s t = Some  $\delta$ "

using mgu\_always\_unifies by mouna

have 1: " $\exists \sigma :: ('f, 'v) \text{subst}. s \cdot \vartheta \cdot \sigma = t \cdot \vartheta \cdot \sigma$ " by (metis unif)

have 2: " $\bigwedge \gamma :: ('f, 'v) \text{subst}. s \cdot \vartheta \cdot \gamma = t \cdot \vartheta \cdot \gamma \implies \delta \preceq_o \vartheta \circ_s \gamma$ " using mgu\_gives\_MGU[OF  $\delta$ ] by

simp

have 3: " $\bigwedge (z :: 'v) (c :: 'f). \delta\ z = \text{Fun } c\ [] \implies \text{Fun } c\ [] \sqsubseteq s \vee \text{Fun } c\ [] \sqsubseteq t$ "

by (rule mgu\_img\_consts'[OF  $\delta$ ])

have 4: "subst\_domain  $\delta \cap \text{range\_vars } \delta = \{\}$ "

```

by (metis mgu_gives_wellformed_subst[OF  $\delta$ ] wf_subst_def)
have 5: "subst_domain  $\delta \cup \text{range\_vars } \delta \subseteq \text{fv } s \cup \text{fv } t"$ 
by (metis mgu_gives_wellformed_MGU[OF  $\delta$ ] wf_MGU_def)

{ fix x and  $\gamma::('f, 'v) \text{ subst}$  assume " $x \in \text{subst\_domain } \vartheta$ "
  hence " $(\vartheta \circ_s \gamma) x = \vartheta x$ "
    using  $\vartheta(4)$  ident_comp_subst_trm_if_disj[of  $\gamma \vartheta$ ]
    unfolding range_vars_alt_def by fast
}
then obtain  $\tau::('f, 'v) \text{ subst}$  where  $\tau: \forall x \in \text{subst\_domain } \vartheta. \vartheta x = (\delta \circ_s \tau) x$  using 1 2 by
moura

have *: " $\bigwedge x. x \in \text{subst\_domain } \delta \cap \text{subst\_domain } \vartheta \implies \exists y \in ?ys. \delta x = \text{Var } y$ "
proof -
  fix x assume " $x \in \text{subst\_domain } \delta \cap ?xs$ "
  hence x: " $x \in \text{subst\_domain } \delta$ " " $x \in \text{subst\_domain } \vartheta$ " by auto
  then obtain c where c: " $\vartheta x = \text{Fun } c []$ " using  $\vartheta(2,5)$  5 by moura
  hence *: " $(\delta \circ_s \tau) x = \text{Fun } c []$ " using  $\tau x$  by fastforce
  hence **: " $x \in \text{subst\_domain } (\delta \circ_s \tau)$ " " $\text{Fun } c [] \in \text{subst\_range } (\delta \circ_s \tau)$ "
    by (auto simp add: subst_domain_def)
  have " $\delta x = \text{Fun } c [] \vee (\exists z. \delta x = \text{Var } z \wedge \tau z = \text{Fun } c [])$ "
    by (rule subst_img_comp_subset_const'[OF *])
  moreover have " $\delta x \neq \text{Fun } c []$ "
  proof (rule ccontr)
    assume " $\neg \delta x \neq \text{Fun } c []$ "
    hence " $\text{Fun } c [] \subseteq s \vee \text{Fun } c [] \subseteq t$ " using 3 by metis
    moreover have " $\forall u \in \text{subst\_range } \vartheta. u \notin \text{subterms } s \cup \text{subterms } t$ "
      using  $\vartheta(3)$  by force
    hence " $\text{Fun } c [] \notin \text{subterms } s \cup \text{subterms } t$ "
      by (metis c (ground (subst_range  $\vartheta$ ))x(2) ground_subst_dom_iff_img)
    ultimately show False by auto
  qed
  moreover have " $\forall x' \in \text{subst\_domain } \vartheta. \delta x \neq \text{Var } x'$ "
  proof (rule ccontr)
    assume " $\neg (\forall x' \in \text{subst\_domain } \vartheta. \delta x \neq \text{Var } x')$ "
    then obtain x' where x': " $x' \in \text{subst\_domain } \vartheta$ " " $\delta x = \text{Var } x'$ " by moura
    hence " $\tau x' = \text{Fun } c []$ " " $(\delta \circ_s \tau) x' = \text{Fun } c []$ " using * unfolding subst_compose_def by auto
    moreover have " $x \neq x'$ "
      using x(1) x'(2) 4
      by (auto simp add: subst_domain_def)
    moreover have " $x' \notin \text{subst\_domain } \delta$ "
      using x'(2) mgu_eliminate_dom[OF  $\delta$ ]
      by (metis (no_types) subst_elim_def subst_apply_term.simps(1) vars_iff_subterm_or_eq)
    moreover have " $(\delta \circ_s \tau) x = \vartheta x$ " " $(\delta \circ_s \tau) x' = \vartheta x'$ " using  $\tau x(2)$  x'(1) by auto
    ultimately show False
      using subterm_inj_on_imp_inj_on[OF  $\vartheta(1)$ ] *
      by (simp add: inj_on_def subst_compose_def x'(2) subst_domain_def)
  qed
  ultimately show " $\exists y \in ?ys. \delta x = \text{Var } y$ "
    by (metis 5 x(2) subterm_eqI' vars_iff_subterm_eq DiffI Un_iff subst_fv_imgI sup.orderE)
qed

have **: " $\text{inj\_on } \delta (\text{subst\_domain } \delta \cap ?xs)$ "
proof (intro inj_onI)
  fix x y assume *:
    " $x \in \text{subst\_domain } \delta \cap \text{subst\_domain } \vartheta$ " " $y \in \text{subst\_domain } \delta \cap \text{subst\_domain } \vartheta$ " " $\delta x = \delta y$ "
  hence " $(\delta \circ_s \tau) x = (\delta \circ_s \tau) y$ " unfolding subst_compose_def by auto
  hence " $\vartheta x = \vartheta y$ " using  $\tau$  * by auto
  thus " $x = y$ " using inj_onD[OF subterm_inj_on_imp_inj_on[OF  $\vartheta(1)$ ]] *(1,2) by simp
qed

define  $\alpha$  where " $\alpha = (\lambda y'. \text{if } \text{Var } y' \in \delta \text{ ' } (\text{subst\_domain } \delta \cap ?xs)$ 
  then  $\text{Var } ((\text{inv\_into } (\text{subst\_domain } \delta \cap ?xs) \delta) (\text{Var } y'))$ "

```

```

      else Var y'::('f,'v) term)"
have a1: "Unifier ( $\delta \circ_s \alpha$ ) s t" using mgu_gives_MGU[OF  $\delta$ ] by auto

define  $\delta'$  where " $\delta' = \delta \circ_s \alpha$ "
have d1: "subst_domain  $\delta' \subseteq ?ys$ "
proof
  fix z assume z: "z  $\in$  subst_domain  $\delta'$ "
  have "z  $\in$  ?xs  $\implies$  z  $\notin$  subst_domain  $\delta'$ "
  proof (cases "z  $\in$  subst_domain  $\delta$ ")
    case True
    moreover assume "z  $\in$  ?xs"
    ultimately have z_in: "z  $\in$  subst_domain  $\delta \cap ?xs$ " by simp
    then obtain y where y: " $\delta$  z = Var y" "y  $\in$  ?ys" using * by moura
    hence " $\alpha$  y = Var ((inv_into (subst_domain  $\delta \cap ?xs$ )  $\delta$ ) (Var y))"
      using  $\alpha\_def$  z_in by simp
    hence " $\alpha$  y = Var z" by (metis y(1) z_in ** inv_into_f_eq)
    hence " $\delta'$  z = Var z" using  $\delta'\_def$  y(1) subst_compose_def[of  $\delta$   $\alpha$ ] by simp
    thus ?thesis by (simp add: subst_domain_def)
  case False
  hence " $\delta$  z = Var z" by (simp add: subst_domain_def)
  moreover assume "z  $\in$  ?xs"
  hence " $\alpha$  z = Var z" using  $\alpha\_def$  * by force
  ultimately show ?thesis
    using  $\delta'\_def$  subst_compose_def[of  $\delta$   $\alpha$ ]
    by (simp add: subst_domain_def)
qed
moreover have "subst_domain  $\alpha \subseteq$  range_vars  $\delta$ "
  unfolding  $\delta'\_def$   $\alpha\_def$  range_vars_alt_def
  by (auto simp add: subst_domain_def)
hence "subst_domain  $\delta' \subseteq$  subst_domain  $\delta \cup$  range_vars  $\delta$ "
  using subst_domain_compose[of  $\delta$   $\alpha$ ] unfolding  $\delta'\_def$  by blast
ultimately show "z  $\in$  ?ys" using 5 z by auto
qed
have d2: "Unifier ( $\delta' \circ_s \mathcal{I}$ ) s t" using a1  $\delta'\_def$  by auto
have d3: " $\mathcal{I} \circ_s \delta' \circ_s \mathcal{I} = \delta' \circ_s \mathcal{I}$ "
proof -
  { fix z::'v assume z: "z  $\in$  ?xs"
    then obtain u where u: " $\mathcal{I}$  z = u" "fv u = {}" using  $\mathcal{I}$  by auto
    hence " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z = u" by (simp add: subst_compose subst_ground_ident)
    moreover have "z  $\notin$  subst_domain  $\delta'$ " using d1 z by auto
    hence " $\delta'$  z = Var z" by (simp add: subst_domain_def)
    hence " $(\delta' \circ_s \mathcal{I})$  z = u" using u(1) by (simp add: subst_compose)
    ultimately have " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z =  $(\delta' \circ_s \mathcal{I})$  z" by metis
  } moreover {
    fix z::'v assume "z  $\in$  ?ys"
    hence "z  $\notin$  subst_domain  $\mathcal{I}$ " using  $\mathcal{I}(2)$  by auto
    hence " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z =  $(\delta' \circ_s \mathcal{I})$  z" by (simp add: subst_compose subst_domain_def)
  } moreover {
    fix z::'v assume "z  $\notin$  ?xs" "z  $\notin$  ?ys"
    hence " $\mathcal{I}$  z = Var z" " $\delta'$  z = Var z" using  $\mathcal{I}(2)$  d1 by blast+
    hence " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z =  $(\delta' \circ_s \mathcal{I})$  z" by (simp add: subst_compose)
  } ultimately show ?thesis by auto
qed

from d2 d3 have "Unifier ( $\delta' \circ_s \mathcal{I}$ ) (s  $\cdot$   $\mathcal{I}$ ) (t  $\cdot$   $\mathcal{I}$ )" by (metis subst_subst_compose)
thus ?thesis by metis
qed

lemma inj_subst_unif_comp_terms:
  fixes  $\mathcal{I} \vartheta \sigma::('f,'v)$  subst and s t::('f,'v) term"
  assumes  $\vartheta$ : "subterm_inj_on  $\vartheta$  (subst_domain  $\vartheta$ )" "ground (subst_range  $\vartheta$ )"
    "subtermsset (subst_range  $\vartheta$ )  $\cap$  (subterms s  $\cup$  subterms t) = {}"

```

```

      "(fv s ∪ fv t) - subst_domain ∅ ⊆ X"
and tfr: "∀f U. Fun f U ∈ subterms s ∪ subterms t ⟶ U = [] ∨ (∃u ∈ set U. u ∉ Var 'X)"
and I: "ground (subst_range I)" "subst_domain I = subst_domain ∅"
and unif: "Unifier σ (s · ∅) (t · ∅)"
shows "∃δ. Unifier δ (s · I) (t · I)"
proof -
  let ?xs = "subst_domain ∅"
  let ?ys = "(fv s ∪ fv t) - ?xs"

  have "ground (subst_range ∅)" using ∅(2) by auto

  have "∃δ::('f,'v) subst. s · δ = t · δ" by (metis subst_subst_compose unif)
  then obtain δ::('f,'v) subst where δ: "mgu s t = Some δ"
    using mgu_always_unifies by moura
  have 1: "∃σ::('f,'v) subst. s · ∅ · σ = t · ∅ · σ" by (metis unif)
  have 2: "∧γ::('f,'v) subst. s · ∅ · γ = t · ∅ · γ ⟹ δ ≼o ∅ ∘s γ" using mgu_gives_MGU[OF δ] by
simp
  have 3: "∧(z::'v) (c::'f). Fun c [] ⊆ δ z ⟹ Fun c [] ⊆ s ∨ Fun c [] ⊆ t"
    using mgu_img_consts[OF δ] by force
  have 4: "subst_domain δ ∩ range_vars δ = {}"
    using mgu_gives_wellformed_subst[OF δ]
    by (metis wf_subst_def)
  have 5: "subst_domain δ ∪ range_vars δ ⊆ fv s ∪ fv t"
    using mgu_gives_wellformed_MGU[OF δ]
    by (metis wf_MGU_def)

  { fix x and γ::('f,'v) subst assume "x ∈ subst_domain ∅"
    hence "(∅ ∘s γ) x = ∅ x"
      using ⟨ground (subst_range ∅)⟩ ident_comp_subst_trm_if_disj[of γ ∅ x]
      unfolding range_vars_alt_def by blast
  }
  then obtain τ::('f,'v) subst where τ: "∀x ∈ subst_domain ∅. ∅ x = (δ ∘s τ) x" using 1 2 by
moura

  have **: "∧x. x ∈ subst_domain δ ∩ subst_domain ∅ ⟹ fv (δ x) ⊆ ?ys"
  proof -
    fix x assume "x ∈ subst_domain δ ∩ ?xs"
    hence x: "x ∈ subst_domain δ" "x ∈ subst_domain ∅" by auto
    moreover have "¬(∃x' ∈ ?xs. x' ∈ fv (δ x))"
    proof (rule ccontr)
      assume "¬¬(∃x' ∈ ?xs. x' ∈ fv (δ x))"
      then obtain x' where x': "x' ∈ fv (δ x)" "x' ∈ ?xs" by metis
      have "x ≠ x'" "x' ∉ subst_domain δ" "δ x' = Var x'"
        using 4 x(1) x'(1) unfolding range_vars_alt_def by auto
      hence "(δ ∘s τ) x' ⊆ (δ ∘s τ) x" "τ x' = (δ ∘s τ) x'"
        using τ x(2) x'(2)
        by (metis subst_compose subst_mono vars_iff_subtermeq x'(1),
            metis subst_apply_term.simps(1) subst_compose_def)
      hence "∅ x' ⊆ ∅ x" using τ x(2) x'(2) by auto
      thus False
        using ∅(1) x'(2) x(2) ⟨x ≠ x'⟩
        unfolding subterm_inj_on_def
        by (meson subtermeqI')
    qed
    ultimately show "fv (δ x) ⊆ ?ys"
      using 5 subst_dom_vars_in_subst[of x δ] subst_fv_imgI[of δ x]
      by blast
  qed

  have **: "inj_on δ (subst_domain δ ∩ ?xs)"
  proof (intro inj_onI)
    fix x y assume *:
      "x ∈ subst_domain δ ∩ subst_domain ∅" "y ∈ subst_domain δ ∩ subst_domain ∅" "δ x = δ y"

```

```

hence "( $\delta \circ_s \tau$ ) x = ( $\delta \circ_s \tau$ ) y" unfolding subst_compose_def by auto
hence " $\vartheta$  x =  $\vartheta$  y" using  $\tau$  * by auto
thus "x = y" using inj_onD[OF subterm_inj_on_imp_inj_on[OF  $\vartheta(1)$ ]] *(1,2) by simp
qed

have *: " $\bigwedge x. x \in \text{subst\_domain } \delta \cap \text{subst\_domain } \vartheta \implies \exists y \in ?ys. \delta x = \text{Var } y$ "
proof (rule ccontr)
  fix xi assume xi_assms: "xi  $\in$  subst_domain  $\delta \cap$  subst_domain  $\vartheta$ " " $\neg(\exists y \in ?ys. \delta xi = \text{Var } y)$ "
  hence xi_ $\vartheta$ : "xi  $\in$  subst_domain  $\vartheta$ " and  $\delta\_xi\_comp$ : " $\neg(\exists y. \delta xi = \text{Var } y)$ "
  using ***[of xi] 5 by auto
  then obtain f T where f: " $\delta xi = \text{Fun } f T$ " by (cases " $\delta xi$ ") moura

  have " $\exists g Y'. Y' \neq [] \wedge \text{Fun } g (\text{map Var } Y') \sqsubseteq \delta xi \wedge \text{set } Y' \subseteq ?ys$ "
  proof -
    have " $\forall c. \text{Fun } c [] \sqsubseteq \delta xi \longrightarrow \text{Fun } c [] \sqsubseteq \vartheta xi$ "
    using  $\tau$  xi_ $\vartheta$  by (metis const_subterm_subst subst_compose)
    hence 1: " $\forall c. \neg(\text{Fun } c [] \sqsubseteq \delta xi)$ "
    using 3[of _ xi] xi_ $\vartheta$   $\vartheta(3)$ 
    by auto

    have " $\neg(\exists x. \delta xi = \text{Var } x)$ " using f by auto
    hence " $\exists g S. \text{Fun } g S \sqsubseteq \delta xi \wedge (\forall s \in \text{set } S. (\exists c. s = \text{Fun } c []) \vee (\exists x. s = \text{Var } x))$ "
    using nonvar_term_has_composed_shallow_term[of " $\delta xi$ "] by auto
    then obtain g S where gS: " $\text{Fun } g S \sqsubseteq \delta xi$ " " $\forall s \in \text{set } S. (\exists c. s = \text{Fun } c []) \vee (\exists x. s = \text{Var } x)$ "
    by moura

    have " $\forall s \in \text{set } S. \exists x. s = \text{Var } x$ "
    using 1 term.order_trans gS
    by (metis (no_types, lifting) UN_I term.order_refl subsetCE subterms.simps(2) sup_ge2)
    then obtain S' where 2: " $\text{map Var } S' = S$ " by (metis ex_map_conv)

    have " $S \neq []$ " using 1 term.order_trans[OF _ gS(1)] by fastforce
    hence 3: " $S' \neq []$ " " $\text{Fun } g (\text{map Var } S') \sqsubseteq \delta xi$ " using gS(1) 2 by auto

    have " $\text{set } S' \subseteq \text{fv } (\text{Fun } g (\text{map Var } S'))$ " by simp
    hence 4: " $\text{set } S' \subseteq \text{fv } (\delta xi)$ " using 3(2) fv_subterms by force

    show ?thesis using ***[OF xi_assms(1)] 2 3 4 by auto
  qed

  then obtain g Y' where g: " $Y' \neq []$ " " $\text{Fun } g (\text{map Var } Y') \sqsubseteq \delta xi$ " " $\text{set } Y' \subseteq ?ys$ " by moura
  then obtain X where X: " $\text{map } \delta X = \text{map Var } Y'$ " " $\text{Fun } g (\text{map Var } X) \in \text{subterms } s \cup \text{subterms } t$ "
  using mgu_img_composed_var_term[OF  $\delta$ , of g Y'] by force
  hence " $\exists (u:('f, 'v) \text{ term}) \in \text{set } (\text{map Var } X). u \notin \text{Var } ' ?ys$ "
  using  $\vartheta(4)$  tfr g(1) by fastforce
  then obtain j where j: " $j < \text{length } X$ " " $X ! j \notin ?ys$ "
  by (metis image_iff[of _ Var "fv s  $\cup$  fv t - subst_domain  $\vartheta$ "] nth_map[of _ X Var]
  in_set_conv_nth[of _ "map Var X"] length_map[of Var X])

  define yj' where yj': " $yj' \equiv Y' ! j$ "
  define xj where xj: " $xj \equiv X ! j$ "

  have " $xj \in \text{fv } s \cup \text{fv } t$ "
  using j X(1) g(3) 5 xj yj'
  by (metis length_map nth_map term.simps(1) in_set_conv_nth le_supE subsetCE subst_domI)
  hence xj_ $\vartheta$ : " $xj \in \text{subst\_domain } \vartheta$ " using j unfolding xj by simp

  have len: " $\text{length } X = \text{length } Y'$ " by (rule map_eq_imp_length_eq[OF X(1)])

  have " $\text{Var } yj' \sqsubseteq \delta xi$ "
  using term.order_trans[OF _ g(2)] j(1) len unfolding yj' by auto
  hence " $\tau yj' \sqsubseteq \vartheta xi$ "
  using  $\tau$  xi_ $\vartheta$  by (metis subst_apply_term.simps(1) subst_compose_def subst_mono)

```

```

moreover have  $\delta_{xj\_var}$ : "Var yj' =  $\delta$  xj"
  using X(1) len j(1) nth_map
  unfolding xj yj' by metis
hence " $\tau$  yj' =  $\vartheta$  xj" using  $\tau$  xj_ $\vartheta$  by (metis subst_apply_term.simps(1) subst_compose_def)
moreover have " $x_i \neq x_j$ " using  $\delta_{xi\_comp}$   $\delta_{xj\_var}$  by auto
ultimately show False using  $\vartheta(1)$   $x_i$ _ $\vartheta$   $x_j$ _ $\vartheta$  unfolding subterm_inj_on_def by blast
qed

define  $\alpha$  where " $\alpha = (\lambda y'. \text{if } \text{Var } y' \in \delta \text{ ' (subst\_domain } \delta \cap ?xs) \text{ then Var ((inv\_into (subst\_domain } \delta \cap ?xs) \delta) (\text{Var } y')) \text{ else Var } y'::('f, 'v) \text{ term})$ "
have a1: "Unifier ( $\delta \circ_s \alpha$ ) s t" using mgu_gives_MGU[OF  $\delta$ ] by auto

define  $\delta'$  where " $\delta' = \delta \circ_s \alpha$ "
have d1: "subst_domain  $\delta' \subseteq ?ys$ "
proof
  fix z assume z: "z  $\in$  subst_domain  $\delta'$ "
  have "z  $\in$  ?xs  $\implies$  z  $\notin$  subst_domain  $\delta'$ "
  proof (cases "z  $\in$  subst_domain  $\delta$ ")
    case True
    moreover assume "z  $\in$  ?xs"
    ultimately have z_in: "z  $\in$  subst_domain  $\delta \cap ?xs$ " by simp
    then obtain y where y: " $\delta$  z = Var y" "y  $\in$  ?ys" using * by moura
    hence " $\alpha$  y = Var ((inv_into (subst_domain  $\delta \cap ?xs$ )  $\delta$ ) (Var y))"
      using  $\alpha\_def$  z_in by simp
    hence " $\alpha$  y = Var z" by (metis y(1) z_in ** inv_into_feq)
    hence " $\delta'$  z = Var z" using  $\delta'\_def$  y(1) subst_compose_def[of  $\delta$   $\alpha$ ] by simp
    thus ?thesis by (simp add: subst_domain_def)
  next
    case False
    hence " $\delta$  z = Var z" by (simp add: subst_domain_def)
    moreover assume "z  $\in$  ?xs"
    hence " $\alpha$  z = Var z" using  $\alpha\_def$  * by force
    ultimately show ?thesis using  $\delta'\_def$  subst_compose_def[of  $\delta$   $\alpha$ ] by (simp add: subst_domain_def)
  qed
next
  case False
  hence " $\delta$  z = Var z" by (simp add: subst_domain_def)
  moreover assume "z  $\in$  ?xs"
  hence " $\alpha$  z = Var z" using  $\alpha\_def$  * by force
  ultimately show ?thesis using  $\delta'\_def$  subst_compose_def[of  $\delta$   $\alpha$ ] by (simp add: subst_domain_def)
qed
moreover have "subst_domain  $\alpha \subseteq \text{range\_vars } \delta$ "
  unfolding  $\delta'\_def$   $\alpha\_def$  range_vars_alt_def subst_domain_def
  by auto
hence "subst_domain  $\delta' \subseteq \text{subst\_domain } \delta \cup \text{range\_vars } \delta$ "
  using subst_domain_compose[of  $\delta$   $\alpha$ ]
  unfolding  $\delta'\_def$  by blast
ultimately show "z  $\in$  ?ys" using 5 z by blast
qed
have d2: "Unifier ( $\delta' \circ_s \mathcal{I}$ ) s t" using a1  $\delta'\_def$  by auto
have d3: " $\mathcal{I} \circ_s \delta' \circ_s \mathcal{I} = \delta' \circ_s \mathcal{I}$ "
proof -
  { fix z::'v assume z: "z  $\in$  ?xs"
    then obtain u where u: " $\mathcal{I}$  z = u" "fv u = {}" using  $\mathcal{I}$  by auto
    hence " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z = u" by (simp add: subst_compose subst_ground_ident)
    moreover have "z  $\notin$  subst_domain  $\delta'$ " using d1 z by auto
    hence " $\delta'$  z = Var z" by (simp add: subst_domain_def)
    hence " $(\delta' \circ_s \mathcal{I})$  z = u" using u(1) by (simp add: subst_compose)
    ultimately have " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z =  $(\delta' \circ_s \mathcal{I})$  z" by metis
  } moreover {
    fix z::'v assume "z  $\in$  ?ys"
    hence "z  $\notin$  subst_domain  $\mathcal{I}$ " using  $\mathcal{I}(2)$  by auto
    hence " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z =  $(\delta' \circ_s \mathcal{I})$  z" by (simp add: subst_compose subst_domain_def)
  } moreover {
    fix z::'v assume "z  $\notin$  ?xs" "z  $\notin$  ?ys"
    hence " $\mathcal{I}$  z = Var z" " $\delta'$  z = Var z" using  $\mathcal{I}(2)$  d1 by blast+
    hence " $(\mathcal{I} \circ_s \delta' \circ_s \mathcal{I})$  z =  $(\delta' \circ_s \mathcal{I})$  z" by (simp add: subst_compose)
  } ultimately show ?thesis by auto
qed

```

```

from d2 d3 have "Unifier ( $\delta' \circ_s \mathcal{I}$ ) ( $s \cdot \mathcal{I}$ ) ( $t \cdot \mathcal{I}$ )" by (metis subst_subst_compose)
thus ?thesis by metis
qed

context
begin
private lemma sat_ineq_subterm_inj_subst_aux:
  fixes  $\mathcal{I}::('f, 'v)$  subst
  assumes "Unifier  $\sigma$  ( $s \cdot \mathcal{I}$ ) ( $t \cdot \mathcal{I}$ )" "ground (subst_range  $\mathcal{I}$ )"
    "(fv  $s \cup \text{fv } t$ ) -  $X \subseteq \text{subst\_domain } \mathcal{I}$ " "subst_domain  $\mathcal{I} \cap X = \{\}$ "
  shows " $\exists \delta::('f, 'v)$  subst. subst_domain  $\delta = X \wedge \text{ground (subst\_range } \delta) \wedge s \cdot \delta \cdot \mathcal{I} = t \cdot \delta \cdot \mathcal{I}$ "
proof -
  have " $\exists \sigma$ . Unifier  $\sigma$  ( $s \cdot \mathcal{I}$ ) ( $t \cdot \mathcal{I}$ )  $\wedge \text{interpretation}_{\text{subst}} \sigma$ "
  proof -
    obtain  $\mathcal{I}'::('f, 'v)$  subst where *: "interpretationsubst  $\mathcal{I}'$ "
      using interpretation_subst_exists by metis
    hence "Unifier ( $\sigma \circ_s \mathcal{I}'$ ) ( $s \cdot \mathcal{I}$ ) ( $t \cdot \mathcal{I}$ )" using assms(1) by simp
    thus ?thesis using * interpretation_comp by blast
  qed
  then obtain  $\sigma'$  where  $\sigma'$ : "Unifier  $\sigma'$  ( $s \cdot \mathcal{I}$ ) ( $t \cdot \mathcal{I}$ )" "interpretationsubst  $\sigma'$ " by moura

  define  $\sigma''$  where " $\sigma'' = \text{rm\_vars (UNIV - X)} \sigma'$ "

  have *: "fv ( $s \cdot \mathcal{I}$ )  $\subseteq X$ " "fv ( $t \cdot \mathcal{I}$ )  $\subseteq X$ "
    using assms(2,3) subst_fv_unfold_ground_img[of  $\mathcal{I}$ ]
    unfolding range_vars_alt_def
    by (simp_all add: Diff_subset_conv Un_commute)
  hence **: "subst_domain  $\sigma'' = X$ " "ground (subst_range  $\sigma''$ )"
    using rm_vars_img_subset[of "UNIV - X"  $\sigma'$ ] rm_vars_dom[of "UNIV - X"  $\sigma'$ ]  $\sigma'$ (2)
    unfolding  $\sigma''$ _def by auto
  hence " $\bigwedge t. t \cdot \mathcal{I} \cdot \sigma'' = t \cdot \sigma'' \cdot \mathcal{I}$ "
    using subst_eq_if_disjoint_vars_ground[OF _ _ assms(2)] assms(4) by blast
  moreover have "Unifier  $\sigma''$  ( $s \cdot \mathcal{I}$ ) ( $t \cdot \mathcal{I}$ )"
    using Unifier_dom_restrict[OF  $\sigma'$ (1)]  $\sigma''$ _def * by blast
  ultimately show ?thesis using ** by auto
qed

```

The "inequality lemma": This lemma gives sufficient syntactic conditions for finding substitutions  $\vartheta$  under which terms  $s$  and  $t$  are not unifiable.

This is useful later when establishing the typing results since we there want to find well-typed solutions to inequality constraints / "negative checks" constraints, and this lemma gives conditions for protocols under which such constraints are well-typed satisfiable if satisfiable.

```

lemma sat_ineq_subterm_inj_subst:
  fixes  $\vartheta \mathcal{I} \delta::('f, 'v)$  subst
  assumes  $\vartheta$ : "subterm_inj_on  $\vartheta$  (subst_domain  $\vartheta$ )"
    "ground (subst_range  $\vartheta$ )"
    "subst_domain  $\vartheta \cap X = \{\}$ "
    "subtermsset (subst_range  $\vartheta$ )  $\cap$  (subterms  $s \cup \text{subterms } t$ ) =  $\{\}$ "
    "(fv  $s \cup \text{fv } t$ ) - subst_domain  $\vartheta \subseteq X$ "
  and tfr: " $(\forall x \in (\text{fv } s \cup \text{fv } t) - X. \exists c. \vartheta x = \text{Fun } c []) \vee$ 
     $(\forall f U. \text{Fun } f U \in \text{subterms } s \cup \text{subterms } t \longrightarrow U = [] \vee (\exists u \in \text{set } U. u \notin \text{Var 'X}))$ "
  and  $\mathcal{I}$ : " $\forall \delta::('f, 'v)$  subst. subst_domain  $\delta = X \wedge \text{ground (subst\_range } \delta) \longrightarrow s \cdot \delta \cdot \mathcal{I} \neq t \cdot \delta \cdot \mathcal{I}$ "
    "(fv  $s \cup \text{fv } t$ ) -  $X \subseteq \text{subst\_domain } \mathcal{I}$ " "subst_domain  $\mathcal{I} \cap X = \{\}$ " "ground (subst_range  $\mathcal{I}$ )"
    "subst_domain  $\mathcal{I} = \text{subst\_domain } \vartheta$ "
  and  $\delta$ : "subst_domain  $\delta = X$ " "ground (subst_range  $\delta$ )"
  shows " $s \cdot \delta \cdot \vartheta \neq t \cdot \delta \cdot \vartheta$ "
proof -
  have " $\forall \sigma. \neg \text{Unifier } \sigma$  ( $s \cdot \mathcal{I}$ ) ( $t \cdot \mathcal{I}$ )"
    by (metis  $\mathcal{I}$ (1) sat_ineq_subterm_inj_subst_aux[OF _  $\mathcal{I}$ (4,2,3)])
  hence " $\neg \text{Unifier } \delta$  ( $s \cdot \vartheta$ ) ( $t \cdot \vartheta$ )"
    using inj_subst_unif_consts[OF  $\vartheta$ (1) _  $\vartheta$ (4,2,3)  $\mathcal{I}$ (4,5)]
    inj_subst_unif_comp_terms[OF  $\vartheta$ (1,2,4,5) _  $\mathcal{I}$ (4,5)]

```



```

      tfr
    by metis
  moreover have "subst_domain  $\delta \cap \text{subst\_domain } \vartheta = \{\}$ " using  $\vartheta(2,3)$   $\delta(1)$  by auto
  ultimately show ?thesis using  $\delta$  subst_eq_if_disjoint_vars_ground[OF  $\vartheta(2)$   $\delta(2)$ ] by metis
qed
end

lemma ineq_subterm_inj_cond_subst:
  assumes " $X \cap \text{range\_vars } \vartheta = \{\}$ "
  and " $\forall f T. \text{Fun } f T \in \text{subterms\_set } S \longrightarrow T = [] \vee (\exists u \in \text{set } T. u \notin \text{Var}'X)$ "
  shows " $\forall f T. \text{Fun } f T \in \text{subterms\_set } (S \cdot_{\text{set}} \vartheta) \longrightarrow T = [] \vee (\exists u \in \text{set } T. u \notin \text{Var}'X)$ "
proof (intro allI impI)
  let ?M = " $\lambda S. \text{subterms\_set } S \cdot_{\text{set}} \vartheta$ "
  let ?N = " $\lambda S. \text{subterms\_set } (\vartheta \cdot (\text{fv\_set } S \cap \text{subst\_domain } \vartheta))$ "

  fix f T assume " $\text{Fun } f T \in \text{subterms\_set } (S \cdot_{\text{set}} \vartheta)$ "
  hence 1: " $\text{Fun } f T \in ?M S \vee \text{Fun } f T \in ?N S$ "
    using subterms_subst[of  $\vartheta$ ] by auto

  have 2: " $\text{Fun } f T \in \text{subterms\_set } (\text{subst\_range } \vartheta) \implies \forall u \in \text{set } T. u \notin \text{Var}'X$ "
    using fv_subset_subterms[of " $\text{Fun } f T$ " " $\text{subst\_range } \vartheta$ "] assms(1)
    unfolding range_vars_alt_def by force

  have 3: " $\forall x \in \text{subst\_domain } \vartheta. \vartheta x \notin \text{Var}'X$ "
  proof
    fix x assume " $x \in \text{subst\_domain } \vartheta$ "
    hence " $\text{fv } (\vartheta x) \subseteq \text{range\_vars } \vartheta$ "
      using subst_dom_vars_in_subst subst_fv_imgI
      unfolding range_vars_alt_def by auto
    thus " $\vartheta x \notin \text{Var}'X$ " using assms(1) by auto
  qed

  show " $T = [] \vee (\exists s \in \text{set } T. s \notin \text{Var}'X)$ " using 1
  proof
    assume " $\text{Fun } f T \in ?M S$ "
    then obtain u where u: " $u \in \text{subterms\_set } S$ " " $u \cdot \vartheta = \text{Fun } f T$ " by fastforce
    show ?thesis
    proof (cases u)
      case (Var x)
      hence " $\text{Fun } f T \in \text{subst\_range } \vartheta$ " using u(2) by (simp add: subst_domain_def)
      hence " $\forall u \in \text{set } T. u \notin \text{Var}'X$ " using 2 by force
      thus ?thesis by auto
    next
      case (Fun g S)
      hence " $S = [] \vee (\exists u \in \text{set } S. u \notin \text{Var}'X)$ " using assms(2) u(1) by metis
      thus ?thesis
      proof
        assume " $S = []$ " thus ?thesis using u(2) Fun by simp
      next
        assume " $\exists u \in \text{set } S. u \notin \text{Var}'X$ "
        then obtain u' where u': " $u' \in \text{set } S$ " " $u' \notin \text{Var}'X$ " by moura
        hence " $u' \cdot \vartheta \in \text{set } T$ " using u(2) Fun by auto
        thus ?thesis using u'(2) 3 by (cases u') force+
      qed
    qed
  next
    assume " $\text{Fun } f T \in ?N S$ "
    thus ?thesis using 2 by force
  qed
qed

```

### 2.3.10 Lemmata: Sufficient Conditions for Term Matching

Injective substitutions from variables to variables are invertible

**definition** `subst_var_inv` where

"`subst_var_inv`  $\delta$   $X \equiv (\lambda x. \text{if } \text{Var } x \in \delta \text{ ' } X \text{ then } \text{Var } ((\text{inv\_into } X \delta) (\text{Var } x)) \text{ else } \text{Var } x)"$

**lemma** `inj_var_ran_subst_is_invertible`:

assumes  $\delta_{\text{inj\_on\_t}}$ : "`inj_on`  $\delta$  (fv  $t$ )"

and  $\delta_{\text{var\_on\_t}}$ : " $\delta \text{ ' } \text{fv } t \subseteq \text{range Var}$ "

shows " $t = t \cdot \delta \circ_s \text{subst\_var\_inv } \delta$  (fv  $t$ )"

**proof** -

have " $\delta x \cdot \text{subst\_var\_inv } \delta$  (fv  $t$ ) =  $\text{Var } x$ " when  $x$ : " $x \in \text{fv } t$ " for  $x$

**proof** -

obtain  $y$  where  $y$ : " $\delta x = \text{Var } y$ " using  $x$   $\delta_{\text{var\_on\_t}}$  by auto

hence " $\text{Var } y \in \delta \text{ ' } (\text{fv } t)$ " using  $x$  by simp

thus ?thesis using  $y$  `inv_into_f_eq[OF  $\delta_{\text{inj\_on\_t}}$   $x$   $y$ ]` unfolding `subst_var_inv_def` by simp

qed

thus ?thesis by (simp add: `subst_compose_def` `trm_subst_ident''`)

qed

Sufficient conditions for matching unifiable terms

**lemma** `inj_var_ran_unifiable_has_subst_match`:

assumes " $t \cdot \delta = s \cdot \delta$ " "`inj_on`  $\delta$  (fv  $t$ )" " $\delta \text{ ' } \text{fv } t \subseteq \text{range Var}$ "

shows " $t = s \cdot \delta \circ_s \text{subst\_var\_inv } \delta$  (fv  $t$ )"

using `assms` `inj_var_ran_subst_is_invertible` by `fastforce`

end

## 2.4 Dolev-Yao Intruder Model (Intruder Deduction)

**theory** `Intruder_Deduction`

**imports** `Messages` `More_Unification`

**begin**

### 2.4.1 Syntax for the Intruder Deduction Relations

**consts** `INTRUDER_SYNT`:: " $('f, 'v) \text{ terms} \Rightarrow ('f, 'v) \text{ term} \Rightarrow \text{bool}$ " (**infix** " `$\vdash_c$` " 50)

**consts** `INTRUDER_DEDUCT`:: " $('f, 'v) \text{ terms} \Rightarrow ('f, 'v) \text{ term} \Rightarrow \text{bool}$ " (**infix** " `$\vdash$` " 50)

### 2.4.2 Intruder Model Locale

The intruder model is parameterized over arbitrary function symbols (e.g, cryptographic operators) and variables. It requires three functions: - `arity` that assigns an arity to each function symbol. - `public` that partitions the function symbols into those that will be available to the intruder and those that will not. - `Ana`, the analysis interface, that defines how messages can be decomposed (e.g., decryption).

**locale** `intruder_model` =

fixes `arity` :: " $'\text{fun} \Rightarrow \text{nat}$ "

and `public` :: " $'\text{fun} \Rightarrow \text{bool}$ "

and `Ana` :: " $('f, 'var) \text{ term} \Rightarrow (('f, 'var) \text{ term list} \times ('f, 'var) \text{ term list})$ "

assumes `Ana_keys_fv`: " $\bigwedge t K R. \text{Ana } t = (K, R) \implies \text{fv}_{\text{set}} (\text{set } K) \subseteq \text{fv } t$ "

and `Ana_keys_wf`: " $\bigwedge t k K R f T.$

$\text{Ana } t = (K, R) \implies (\bigwedge g S. \text{Fun } g S \sqsubseteq t \implies \text{length } S = \text{arity } g)$

$\implies k \in \text{set } K \implies \text{Fun } f T \sqsubseteq k \implies \text{length } T = \text{arity } f$ "

and `Ana_var[simp]`: " $\bigwedge x. \text{Ana } (\text{Var } x) = ([], [])$ "

and `Ana_fun_subterm`: " $\bigwedge f T K R. \text{Ana } (\text{Fun } f T) = (K, R) \implies \text{set } R \subseteq \text{set } T$ "

and `Ana_subst`: " $\bigwedge t \delta K R. [\![\text{Ana } t = (K, R); K \neq [] \vee R \neq []]\!] \implies \text{Ana } (t \cdot \delta) = (K \cdot_{\text{list}} \delta, R \cdot_{\text{list}}$

$\delta)$ "

**begin**

**lemma** `Ana_subterm`: assumes " $\text{Ana } t = (K, T)$ " shows " $\text{set } T \subseteq \text{subterms } t$ "

using `assms`

```

by (cases t)
  (simp add: psubsetI,
    metis Ana_fun_subterm Fun_gt_params UN_I term.order_refl
      params_subterms psubsetI subset_antisym subset_trans)

lemma Ana_subterm': "s ∈ set (snd (Ana t)) ⟹ s ⊆ t"
using Ana_subterm by (cases "Ana t") auto

lemma Ana_vars: assumes "Ana t = (K,M)" shows "fv_set (set K) ⊆ fv t" "fv_set (set M) ⊆ fv t"
by (rule Ana_keys_fv[OF assms]) (use Ana_subterm[OF assms] subterm_eq_vars_subset in auto)

abbreviation V where "V ≡ UNIV::'var set"
abbreviation Σn ("Σn-") where "Σn ≡ {f::'fun. arity f = n}"
abbreviation Σpub ("Σpubn-") where "Σpubn ≡ {f. public f} ∩ Σn"
abbreviation Σpriv ("Σprivn-") where "Σprivn ≡ {f. ¬public f} ∩ Σn"
abbreviation Σpub where "Σpub ≡ (⋃n. Σpubn)"
abbreviation Σpriv where "Σpriv ≡ (⋃n. Σprivn)"
abbreviation Σ where "Σ ≡ (⋃n. Σn)"
abbreviation C where "C ≡ Σ0"
abbreviation Cpub where "Cpub ≡ {f. public f} ∩ C"
abbreviation Cpriv where "Cpriv ≡ {f. ¬public f} ∩ C"
abbreviation Σf where "Σf ≡ Σ - C"
abbreviation Σfpub where "Σfpub ≡ Σf ∩ Σpub"
abbreviation Σfpriv where "Σfpriv ≡ Σf ∩ Σpriv"

lemma disjoint_fun_syms: "Σf ∩ C = {}" by auto
lemma id_union_univ: "Σf ∪ C = UNIV" "Σ = UNIV" by auto
lemma const_arity_eq_zero[dest]: "c ∈ C ⟹ arity c = 0" by simp
lemma const_pub_arity_eq_zero[dest]: "c ∈ Cpub ⟹ arity c = 0 ∧ public c" by simp
lemma const_priv_arity_eq_zero[dest]: "c ∈ Cpriv ⟹ arity c = 0 ∧ ¬public c" by simp
lemma fun_arity_gt_zero[dest]: "f ∈ Σf ⟹ arity f > 0" by fastforce
lemma pub_fun_public[dest]: "f ∈ Σfpub ⟹ public f" by fastforce
lemma pub_fun_arity_gt_zero[dest]: "f ∈ Σfpub ⟹ arity f > 0" by fastforce

lemma Σf_unfold: "Σf = {f::'fun. arity f > 0}" by auto
lemma C_unfold: "C = {f::'fun. arity f = 0}" by auto
lemma Cpub_unfold: "Cpub = {f::'fun. arity f = 0 ∧ public f}" by auto
lemma Cpriv_unfold: "Cpriv = {f::'fun. arity f = 0 ∧ ¬public f}" by auto
lemma Σpub_unfold: "(Σpubn) = {f::'fun. arity f = n ∧ public f}" by auto
lemma Σpriv_unfold: "(Σprivn) = {f::'fun. arity f = n ∧ ¬public f}" by auto
lemma Σfpub_unfold: "Σfpub = {f::'fun. arity f > 0 ∧ public f}" by auto
lemma Σfpriv_unfold: "Σfpriv = {f::'fun. arity f > 0 ∧ ¬public f}" by auto
lemma Σn_m_eq: "[ (Σn) ≠ {}; (Σn) = (Σm) ] ⟹ n = m" by auto

```

### 2.4.3 Term Well-formedness

definition "wf<sub>trm</sub> t ≡ ∀f T. Fun f T ⊆ t ⟶ length T = arity f"

abbreviation "wf<sub>trms</sub> T ≡ ∀t ∈ T. wf<sub>trm</sub> t"

lemma Ana\_keys\_wf': "Ana t = (K,T) ⟹ wf<sub>trm</sub> t ⟹ k ∈ set K ⟹ wf<sub>trm</sub> k"

using Ana\_keys\_wf unfolding wf<sub>trm</sub>\_def by metis

lemma wf\_trm\_Var[simp]: "wf<sub>trm</sub> (Var x)" unfolding wf<sub>trm</sub>\_def by simp

lemma wf\_trm\_subst\_range\_Var[simp]: "wf<sub>trms</sub> (subst\_range Var)" by simp

lemma wf\_trm\_subst\_range\_iff: "(∀x. wf<sub>trm</sub> (ϑ x)) ⟷ wf<sub>trms</sub> (subst\_range ϑ)"

by force

lemma wf\_trm\_subst\_ranged: "wf<sub>trms</sub> (subst\_range ϑ) ⟹ wf<sub>trm</sub> (ϑ x)"

by (metis wf\_trm\_subst\_range\_iff)

```

lemma wf_trm_subst_rangeI[intro]:
  "(\x. wf_trm (\delta x)) \implies wf_trm_s (subst_range \delta)"
by (metis wf_trm_subst_range_iff)

lemma wf_trmI[intro]:
  assumes "\t. t \in set T \implies wf_trm t" "length T = arity f"
  shows "wf_trm (Fun f T)"
using assms unfolding wf_trm_def by auto

lemma wf_trm_subterm: "[wf_trm t; s \sqsubseteq t] \implies wf_trm s"
unfolding wf_trm_def by (induct t) auto

lemma wf_trm_subtermeq:
  assumes "wf_trm t" "s \sqsubseteq t"
  shows "wf_trm s"
proof (cases "s = t")
  case False thus "wf_trm s" using assms(2) wf_trm_subterm[OF assms(1)] by simp
qed (metis assms(1))

lemma wf_trm_param:
  assumes "wf_trm (Fun f T)" "t \in set T"
  shows "wf_trm t"
by (meson assms subtermeqI' wf_trm_subtermeq)

lemma wf_trm_param_idx:
  assumes "wf_trm (Fun f T)"
  and "i < length T"
  shows "wf_trm (T ! i)"
using wf_trm_param[OF assms(1), of "T ! i"] assms(2)
by fastforce

lemma wf_trm_subst:
  assumes "wf_trm_s (subst_range \delta)"
  shows "wf_trm t = wf_trm (t \cdot \delta)"
proof
  show "wf_trm t \implies wf_trm (t \cdot \delta)"
  proof (induction t)
    case (Fun f T)
    hence "\t. t \in set T \implies wf_trm t"
    by (meson wf_trm_def Fun_param_is_subterm term.order_trans)
    hence "\t. t \in set T \implies wf_trm (t \cdot \delta)" using Fun.IH by auto
    moreover have "length (map (\t. t \cdot \delta) T) = arity f"
    using Fun.prem unfolding wf_trm_def by auto
    ultimately show ?case by fastforce
  qed (simp add: wf_trm_subst_rangeD[OF assms])

  show "wf_trm (t \cdot \delta) \implies wf_trm t"
  proof (induction t)
    case (Fun f T)
    hence "wf_trm t" when "t \in set (map (\s. s \cdot \delta) T)" for t
    by (metis that wf_trm_def Fun_param_is_subterm term.order_trans subst_apply_term.simps(2))
    hence "wf_trm t" when "t \in set T" for t using that Fun.IH by auto
    moreover have "length (map (\t. t \cdot \delta) T) = arity f"
    using Fun.prem unfolding wf_trm_def by auto
    ultimately show ?case by fastforce
  qed (simp add: assms)
qed

lemma wf_trm_subst_singleton:
  assumes "wf_trm t" "wf_trm t'" shows "wf_trm (t \cdot Var(v := t'))"
proof -
  have "wf_trm ((Var(v := t')) w)" for w using assms(2) unfolding wf_trm_def by simp
  thus ?thesis using assms(1) wf_trm_subst[of "Var(v := t')" t, OF wf_trm_subst_rangeI] by simp

```

qed

```

lemma wf_trm_subst_rm_vars:
  assumes "wf_trm (t · δ)"
  shows "wf_trm (t · rm_vars X δ)"
using assms
proof (induction t)
  case (Fun f T)
  have "wf_trm (t · δ)" when "t ∈ set T" for t
    using that wf_trm_param[of f "map (λt. t · δ) T"] Fun.prem
    by auto
  hence "wf_trm (t · rm_vars X δ)" when "t ∈ set T" for t using that Fun.IH by simp
  moreover have "length T = arity f" using Fun.prem unfolding wf_trm_def by auto
  ultimately show ?case unfolding wf_trm_def by auto
qed simp

```

```

lemma wf_trm_subst_rm_vars': "wf_trm (δ v) ⇒ wf_trm (rm_vars X δ v)"
by auto

```

```

lemma wf_trms_subst:
  assumes "wf_trms (subst_range δ)" "wf_trms M"
  shows "wf_trms (M ·set δ)"
by (metis (no_types, lifting) assms imageE wf_trm_subst)

```

```

lemma wf_trms_subst_rm_vars:
  assumes "wf_trms (M ·set δ)"
  shows "wf_trms (M ·set rm_vars X δ)"
using assms wf_trm_subst_rm_vars by blast

```

```

lemma wf_trms_subst_rm_vars':
  assumes "wf_trms (subst_range δ)"
  shows "wf_trms (subst_range (rm_vars X δ))"
using assms by force

```

```

lemma wf_trms_subst_compose:
  assumes "wf_trms (subst_range ϑ)" "wf_trms (subst_range δ)"
  shows "wf_trms (subst_range (ϑ ∘s δ))"
using assms subst_img_comp_subset' wf_trm_subst by blast

```

```

lemma wf_trm_subst_compose:
  fixes δ::('fun, 'v) subst
  assumes "wf_trm (ϑ x)" "⋀x. wf_trm (δ x)"
  shows "wf_trm ((ϑ ∘s δ) x)"
using wf_trm_subst[of δ "ϑ x", OF wf_trm_subst_rangeI[OF assms(2)]] assms(1)
  subst_subst_compose[of "Var x" ϑ δ]
  subst_apply_term.simps(1)[of x ϑ]
  subst_apply_term.simps(1)[of x "ϑ ∘s δ"]
by argo

```

```

lemma wf_trms_Var_range:
  assumes "subst_range δ ⊆ range Var"
  shows "wf_trms (subst_range δ)"
using assms by fastforce

```

```

lemma wf_trms_subst_compose_Var_range:
  assumes "wf_trms (subst_range ϑ)"
  and "subst_range δ ⊆ range Var"
  shows "wf_trms (subst_range (δ ∘s ϑ))"
  and "wf_trms (subst_range (ϑ ∘s δ))"
using assms wf_trms_subst_compose wf_trms_Var_range by metis+

```

```

lemma wf_trm_subst_inv: "wf_trm (t · δ) ⇒ wf_trm t"
unfolding wf_trm_def by (induct t) auto

```

```

lemma wf_trms_subst_inv: "wf_trms (M ·set δ) ⇒ wf_trms M"
using wf_trm_subst_inv by fast

lemma wf_trm_subterms: "wf_trm t ⇒ wf_trms (subterms t)"
using wf_trm_subterm by blast

lemma wf_trms_subterms: "wf_trms M ⇒ wf_trms (subtermsset M)"
using wf_trm_subterms by blast

lemma wf_trm_arity: "wf_trm (Fun f T) ⇒ length T = arity f"
unfolding wf_trm_def by blast

lemma wf_trm_subterm_arity: "wf_trm t ⇒ Fun f T ⊆ t ⇒ length T = arity f"
unfolding wf_trm_def by blast

lemma unify_list_wf_trm:
  assumes "Unification.unify E B = Some U" "∀(s,t) ∈ set E. wf_trm s ∧ wf_trm t"
  and "∀(v,t) ∈ set B. wf_trm t"
  shows "∀(v,t) ∈ set U. wf_trm t"
using assms
proof (induction E B arbitrary: U rule: Unification.unify.induct)
  case (1 B U) thus ?case by auto
next
  case (2 f T g S E B U)
  have wf_fun: "wf_trm (Fun f T)" "wf_trm (Fun g S)" using "2.prem1" by auto
  from "2.prem1" obtain E' where *: "decompose (Fun f T) (Fun g S) = Some E'"
  and [simp]: "f = g" "length T = length S" "E' = zip T S"
  and **: "Unification.unify (E'@E) B = Some U"
  by (auto split: option.splits)
  hence "t ⊆ Fun f T" "t' ⊆ Fun g S" when "(t,t') ∈ set E'" for t t'
  using that by (metis zip_arg_subterm(1), metis zip_arg_subterm(2))
  hence "wf_trm t" "wf_trm t'" when "(t,t') ∈ set E'" for t t'
  using wf_trm_subterm wf_fun (f = g) that by blast+
  thus ?case using "2.IH"[OF * ** _ "2.prem3"] "2.prem2" by fastforce
next
  case (3 v t E B)
  hence *: "∀(w,x) ∈ set ((v, t) # B). wf_trm x"
  and **: "∀(s,t) ∈ set E. wf_trm s ∧ wf_trm t" "wf_trm t"
  by auto

  show ?case
  proof (cases "t = Var v")
    case True thus ?thesis using "3.prem1" "3.IH"(1) by auto
  next
    case False
    hence "v ∉ fv t" using "3.prem1" by auto
    hence "Unification.unify (subst_list (subst v t) E) ((v, t)#B) = Some U"
    using (t ≠ Var v) "3.prem1" by auto
    moreover have "∀(s, t) ∈ set (subst_list (subst v t) E). wf_trm s ∧ wf_trm t"
    using wf_trm_subst_singleton[OF _ (wf_trm t)] "3.prem2"
    unfolding subst_list_def subst_def by auto
    ultimately show ?thesis using "3.IH"(2)[OF (t ≠ Var v) (v ∉ fv t) _ _] by metis
  qed
next
  case (4 f T v E B U)
  hence *: "∀(w,x) ∈ set ((v, Fun f T) # B). wf_trm x"
  and **: "∀(s,t) ∈ set E. wf_trm s ∧ wf_trm t" "wf_trm (Fun f T)"
  by auto

  have "v ∉ fv (Fun f T)" using "4.prem1" by force
  hence "Unification.unify (subst_list (subst v (Fun f T)) E) ((v, Fun f T)#B) = Some U"
  using "4.prem1" by auto

```

```

moreover have "∀ (s, t) ∈ set (subst_list (subst v (Fun f T)) E). wf_trm s ∧ wf_trm t"
  using wf_trm_subst_singleton[OF _ ⟨wf_trm (Fun f T)⟩] "4.premis"(2)
  unfolding subst_list_def subst_def by auto
ultimately show ?case using "4.IH"[OF ⟨v ∉ fv (Fun f T)⟩ _ _] by metis
qed

lemma mgu_wf_trm:
  assumes "mgu s t = Some σ" "wf_trm s" "wf_trm t"
  shows "wf_trm (σ v)"
proof -
  from assms obtain σ' where "subst_of σ' = σ" "∀ (v,t) ∈ set σ'. wf_trm t"
  using unify_list_wf_trm[of "[(s,t)]" "[]"] by (auto split: option.splits)
  thus ?thesis
  proof (induction σ' arbitrary: σ v rule: List.rev_induct)
    case (snoc x σ' σ v)
    define ϑ where "ϑ = subst_of σ'"
    hence "wf_trm (ϑ v)" for v using snoc.premis(2) snoc.IH[of ϑ] by fastforce
    moreover obtain w t where x: "x = (w,t)" by (metis surj_pair)
    hence σ: "σ = Var(w := t) ∘s ϑ" using snoc.premis(1) by (simp add: subst_def ϑ_def)
    moreover have "wf_trm t" using snoc.premis(2) x by auto
    ultimately show ?case using wf_trm_subst[of _ t] unfolding subst_compose_def by auto
  qed (simp add: wf_trm_def)
qed

lemma mgu_wf_trms:
  assumes "mgu s t = Some σ" "wf_trm s" "wf_trm t"
  shows "wf_trms (subst_range σ)"
using mgu_wf_trm[OF assms] by simp

```

### 2.4.4 Definitions: Intruder Deduction Relations

A standard Dolev-Yao intruder.

```

inductive intruder_deduct:: "('fun,'var) terms ⇒ ('fun,'var) term ⇒ bool"
where
  Axiom[simp]: "t ∈ M ⇒ intruder_deduct M t"
| Compose[simp]: "[length T = arity f; public f; ∧ t. t ∈ set T ⇒ intruder_deduct M t]
  ⇒ intruder_deduct M (Fun f T)"
| Decompose: "[intruder_deduct M t; Ana t = (K, T); ∧ k. k ∈ set K ⇒ intruder_deduct M k;
  t_i ∈ set T]
  ⇒ intruder_deduct M t_i"

```

A variant of the intruder relation which limits the intruder to composition only.

```

inductive intruder_synth:: "('fun,'var) terms ⇒ ('fun,'var) term ⇒ bool"
where
  AxiomC[simp]: "t ∈ M ⇒ intruder_synth M t"
| ComposeC[simp]: "[length T = arity f; public f; ∧ t. t ∈ set T ⇒ intruder_synth M t]
  ⇒ intruder_synth M (Fun f T)"

```

```

adhoc_overloading INTRUDER_DEDUCT intruder_deduct
adhoc_overloading INTRUDER_SYNTH intruder_synth

```

```

lemma intruder_deduct_induct[consumes 1, case_names Axiom Compose Decompose]:
  assumes "M ⊢ t" "∧ t. t ∈ M ⇒ P M t"
  "∧ T f. [length T = arity f; public f;
    ∧ t. t ∈ set T ⇒ M ⊢ t;
    ∧ t. t ∈ set T ⇒ P M t] ⇒ P M (Fun f T)"
  "∧ t K T t_i. [M ⊢ t; P M t; Ana t = (K, T); ∧ k. k ∈ set K ⇒ M ⊢ k;
    ∧ k. k ∈ set K ⇒ P M k; t_i ∈ set T] ⇒ P M t_i"
  shows "P M t"
using assms by (induct rule: intruder_deduct.induct) blast+

```

```

lemma intruder_synth_induct[consumes 1, case_names AxiomC ComposeC]:

```

```

fixes M::("('fun,'var) terms" and t::("('fun,'var) term"
assumes "M ⊢c t" "⋀t. t ∈ M ⇒ P M t"
      "⋀T f. [length T = arity f; public f;
              ⋀t. t ∈ set T ⇒ M ⊢c t;
              ⋀t. t ∈ set T ⇒ P M t] ⇒ P M (Fun f T)"
shows "P M t"
using assms by (induct rule: intruder_synth.induct) auto

```

## 2.4.5 Definitions: Analyzed Knowledge and Public Ground Well-formed Terms (PGWTs)

```

definition analyzed::("('fun,'var) terms ⇒ bool" where
  "analyzed M ≡ ∀t. M ⊢ t ⇔ M ⊢c t"

```

```

definition analyzed_in where
  "analyzed_in t M ≡ ∀K R. (Ana t = (K,R) ∧ (∀k ∈ set K. M ⊢c k)) → (∀r ∈ set R. M ⊢c r)"

```

```

definition decomp_closure::("('fun,'var) terms ⇒ ('fun,'var) terms ⇒ bool" where
  "decomp_closure M M' ≡ ∀t. M ⊢ t ∧ (∃t' ∈ M. t ⊆ t') ⇔ t ∈ M'"

```

```

inductive public_ground_wf_term::("('fun,'var) term ⇒ bool" where
  PGWT[simp]: "⌊public f; arity f = length T;
               ⋀t. t ∈ set T ⇒ public_ground_wf_term t⌋
               ⇒ public_ground_wf_term (Fun f T)"

```

```

abbreviation "public_ground_wf_terms ≡ {t. public_ground_wf_term t}"

```

```

lemma public_const_deduct:
  assumes "c ∈ Cpub"
  shows "M ⊢ Fun c []" "M ⊢c Fun c []"
proof -
  have "arity c = 0" "public c" using const_arity_eq_zero ⟨c ∈ Cpub⟩ by auto
  thus "M ⊢ Fun c []" "M ⊢c Fun c []"
    using intruder_synth.ComposeC[OF _ ⟨public c⟩, of "[]"]
          intruder_deduct.Compose[OF _ ⟨public c⟩, of "[]"]
    by auto
qed

```

```

lemma public_const_deduct'[simp]:
  assumes "arity c = 0" "public c"
  shows "M ⊢ Fun c []" "M ⊢c Fun c []"
using intruder_deduct.Compose[of "[]" c] intruder_synth.ComposeC[of "[]" c] assms by simp_all

```

```

lemma private_fun_deduct_in_ik:
  assumes t: "M ⊢ t" "Fun f T ∈ subterms t"
  and f: "¬public f"
  shows "Fun f T ∈ subtermsset M"
using t
proof (induction t rule: intruder_deduct.induct)
  case Decompose thus ?case by (meson Ana_subterm psubsetD term.order_trans)
qed (auto simp add: f in_subterms_Union)

```

```

lemma private_fun_deduct_in_ik':
  assumes t: "M ⊢ Fun f T"
  and f: "¬public f"
  and M: "Fun f T ∈ subtermsset M ⇒ Fun f T ∈ M"
  shows "Fun f T ∈ M"
by (rule M[OF private_fun_deduct_in_ik[OF t term.order_refl f]])

```

```

lemma pgwt_public: "⌊public_ground_wf_term t; Fun f T ⊆ t⌋ ⇒ public f"
by (induct t rule: public_ground_wf_term.induct) auto

```

```

lemma pgwt_ground: "public_ground_wf_term t ⇒ fv t = {}"
by (induct t rule: public_ground_wf_term.induct) auto

```



```

lemma pgwt_fun: "public_ground_wf_term t  $\implies \exists f T. t = \text{Fun } f T$ "
using pgwt_ground[of t] by (cases t) auto

lemma pgwt_arity: "[public_ground_wf_term t; Fun f T  $\sqsubseteq$  t]  $\implies \text{arity } f = \text{length } T$ "
by (induct t rule: public_ground_wf_term.induct) auto

lemma pgwt_wellformed: "public_ground_wf_term t  $\implies \text{wf}_{trm} t$ "
by (induct t rule: public_ground_wf_term.induct) auto

lemma pgwt_deducible: "public_ground_wf_term t  $\implies M \vdash_c t$ "
by (induct t rule: public_ground_wf_term.induct) auto

lemma pgwt_is_empty_synth: "public_ground_wf_term t  $\longleftrightarrow \{\} \vdash_c t$ "
proof -
  { fix M:: "('fun,'var) term set" assume "M  $\vdash_c t$ " "M = {}" hence "public_ground_wf_term t"
    by (induct t rule: intruder_synth.induct) auto
  }
  thus ?thesis using pgwt_deducible by auto
qed

lemma ideduct_synth_subst_apply:
  fixes M:: "('fun,'var) terms" and t:: "('fun,'var) term"
  assumes "{ }  $\vdash_c t$ " " $\bigwedge v. M \vdash_c \vartheta v$ "
  shows "M  $\vdash_c t \cdot \vartheta$ "
proof -
  { fix M':: "('fun,'var) term set" assume "M'  $\vdash_c t$ " "M' = {}" hence "M  $\vdash_c t \cdot \vartheta$ "
    proof (induction t rule: intruder_synth.induct)
      case (ComposeC T f M')
        hence "length (map ( $\lambda t. t \cdot \vartheta$ ) T) = arity f" " $\bigwedge x. x \in \text{set } (\text{map } (\lambda t. t \cdot \vartheta) T) \implies M \vdash_c x$ "
        by auto
        thus ?case using intruder_synth.ComposeC[of "map ( $\lambda t. t \cdot \vartheta$ ) T" f M] (public f) by fastforce
      qed simp
    }
    thus ?thesis using assms by metis
  }
qed

```

#### 2.4.6 Lemmata: Monotonicity, deduction private constants, etc.

```

context
begin
lemma ideduct_mono:
  "[[M  $\vdash t$ ; M  $\subseteq M'$ ]]  $\implies M' \vdash t$ "
proof (induction rule: intruder_deduct.induct)
  case (Decompose M t K T t_i)
  have " $\forall k. k \in \text{set } K \longrightarrow M' \vdash k$ " using Decompose.IH (M  $\subseteq M'$ ) by simp
  moreover have "M'  $\vdash t$ " using Decompose.IH (M  $\subseteq M'$ ) by simp
  ultimately show ?case using Decompose.hyps intruder_deduct.Decompose by blast
qed auto

lemma ideduct_synth_mono:
  fixes M:: "('fun,'var) terms" and t:: "('fun,'var) term"
  shows "[[M  $\vdash_c t$ ; M  $\subseteq M'$ ]]  $\implies M' \vdash_c t$ "
by (induct rule: intruder_synth.induct) auto

lemma ideduct_reduce:
  "[[M  $\cup M' \vdash t$ ;  $\bigwedge t'. t' \in M' \implies M \vdash t'$ ]]  $\implies M \vdash t$ "
proof (induction rule: intruder_deduct.induct)
  case Decompose thus ?case using intruder_deduct.Decompose by blast
qed auto

lemma ideduct_synth_reduce:
  fixes M:: "('fun,'var) terms" and t:: "('fun,'var) term"

```

```

shows "⟦M ∪ M' ⊢c t; ∧t'. t' ∈ M' ⇒ M ⊢c t'⟧ ⇒ M ⊢c t"
by (induct rule: intruder_synth_induct) auto

lemma ideduct_mono_eq:
  assumes "∀t. M ⊢ t ⇔ M' ⊢ t" shows "M ∪ N ⊢ t ⇔ M' ∪ N ⊢ t"
proof
  show "M ∪ N ⊢ t ⇒ M' ∪ N ⊢ t"
  proof (induction t rule: intruder_deduct_induct)
    case (Axiom t) thus ?case
    proof (cases "t ∈ M")
      case True
      hence "M ⊢ t" using intruder_deduct.Axiom by metis
      thus ?thesis using assms ideduct_mono[of M' t "M' ∪ N"] by simp
    qed auto
  next
    case (Compose T f) thus ?case using intruder_deduct.Compose by auto
  next
    case (Decompose t K T ti) thus ?case using intruder_deduct.Decompose[of "M' ∪ N" t K T] by auto
  qed

  show "M' ∪ N ⊢ t ⇒ M ∪ N ⊢ t"
  proof (induction t rule: intruder_deduct_induct)
    case (Axiom t) thus ?case
    proof (cases "t ∈ M'")
      case True
      hence "M' ⊢ t" using intruder_deduct.Axiom by metis
      thus ?thesis using assms ideduct_mono[of M t "M ∪ N"] by simp
    qed auto
  next
    case (Compose T f) thus ?case using intruder_deduct.Compose by auto
  next
    case (Decompose t K T ti) thus ?case using intruder_deduct.Decompose[of "M ∪ N" t K T] by auto
  qed

lemma deduct_synth_subterm:
  fixes M::"('fun,'var) terms" and t::"('fun,'var) term"
  assumes "M ⊢c t" "s ∈ subterms t" "∀m ∈ M. ∀s ∈ subterms m. M ⊢c s"
  shows "M ⊢c s"
using assms by (induct t rule: intruder_synth_induct) auto

lemma deduct_if_synth[intro, dest]: "M ⊢c t ⇒ M ⊢ t"
by (induct rule: intruder_synth_induct) auto

private lemma ideduct_ik_eq: assumes "∀t ∈ M. M' ⊢ t" shows "M' ⊢ t ⇔ M' ∪ M ⊢ t"
by (meson assms ideduct_mono ideduct_reduce sup_ge1)

private lemma synth_if_deduct_empty: "{} ⊢ t ⇒ {} ⊢c t"
proof (induction t rule: intruder_deduct_induct)
  case (Decompose t K M m)
  then obtain f T where "t = Fun f T" "m ∈ set T"
  using Ana_fun_subterm Ana_var by (cases t) fastforce+
  with Decompose.IH(1) show ?case by (induction rule: intruder_synth_induct) auto
qed auto

private lemma ideduct_deduct_synth_mono_eq:
  assumes "∀t. M ⊢ t ⇔ M' ⊢c t" "M ⊆ M'"
  and "∀t. M' ∪ N ⊢ t ⇔ M' ∪ N ∪ D ⊢c t"
  shows "M ∪ N ⊢ t ⇔ M' ∪ N ∪ D ⊢c t"
proof -
  have "∀m ∈ M'. M ⊢ m" using assms(1) by auto
  hence "∀t. M ⊢ t ⇔ M' ⊢ t" by (metis assms(1,2) deduct_if_synth ideduct_reduce sup.absorb2)
  hence "∀t. M' ∪ N ⊢ t ⇔ M ∪ N ⊢ t" by (meson ideduct_mono_eq)

```

```

    thus ?thesis by (meson assms(3))
qed

lemma ideduct_subst: "M ⊢ t ⇒ M ·set δ ⊢ t · δ"
proof (induction t rule: intruder_deduct_induct)
  case (Compose T f)
  hence "length (map (λt. t · δ) T) = arity f" "⋀t. t ∈ set T ⇒ M ·set δ ⊢ t · δ" by auto
  thus ?case using intruder_deduct.Compose[OF _ Compose.hyps(2), of "map (λt. t · δ) T"] by auto
next
  case (Decompose t K M' m')
  hence "Ana (t · δ) = (K ·list δ, M' ·list δ)"
    "⋀k. k ∈ set (K ·list δ) ⇒ M ·set δ ⊢ k"
    "m' · δ ∈ set (M' ·list δ)"
  using Ana_subst[OF Decompose.hyps(2)] by fastforce+
  thus ?case using intruder_deduct.Decompose[OF Decompose.IH(1)] by metis
qed simp

lemma ideduct_synth_subst:
  fixes M::"('fun,'var) terms" and t::"('fun,'var) term" and δ::"('fun,'var) subst"
  shows "M ⊢c t ⇒ M ·set δ ⊢c t · δ"
proof (induction t rule: intruder_synth_induct)
  case (ComposeC T f)
  hence "length (map (λt. t · δ) T) = arity f" "⋀t. t ∈ set T ⇒ M ·set δ ⊢c t · δ" by auto
  thus ?case using intruder_synth.ComposeC[OF _ ComposeC.hyps(2), of "map (λt. t · δ) T"] by auto
qed simp

lemma ideduct_vars:
  assumes "M ⊢ t"
  shows "fv t ⊆ fvset M"
using assms
proof (induction t rule: intruder_deduct_induct)
  case (Decompose t K T ti) thus ?case
    using Ana_vars(2) fv_subset by blast
qed auto

lemma ideduct_synth_vars:
  fixes M::"('fun,'var) terms" and t::"('fun,'var) term"
  assumes "M ⊢c t"
  shows "fv t ⊆ fvset M"
using assms by (induct t rule: intruder_synth_induct) auto

lemma ideduct_synth_priv_fun_in_ik:
  fixes M::"('fun,'var) terms" and t::"('fun,'var) term"
  assumes "M ⊢c t" "f ∈ funs_term t" "¬public f"
  shows "f ∈ ⋃ (funs_term ' M)"
using assms by (induct t rule: intruder_synth_induct) auto

lemma ideduct_synth_priv_const_in_ik:
  fixes M::"('fun,'var) terms" and t::"('fun,'var) term"
  assumes "M ⊢c Fun c []" "¬public c"
  shows "Fun c [] ∈ M"
using intruder_synth.cases[OF assms(1)] assms(2) by fast

lemma ideduct_synth_ik_replace:
  fixes M::"('fun,'var) terms" and t::"('fun,'var) term"
  assumes "∀t ∈ M. N ⊢c t"
  and "M ⊢c t"
  shows "N ⊢c t"
using assms(2,1) by (induct t rule: intruder_synth_induct) auto
end

```

### 2.4.7 Lemmata: Analyzed Intruder Knowledge Closure

lemma deducts\_eq\_if\_analyzed: "analyzed  $M \implies M \vdash t \longleftrightarrow M \vdash_c t$ "  
 unfolding analyzed\_def by auto

lemma closure\_is\_superset: "decomp\_closure  $M M' \implies M \subseteq M'$ "  
 unfolding decomp\_closure\_def by force

lemma deduct\_if\_closure\_deduct: " $\llbracket M' \vdash t; \text{decomp\_closure } M M' \rrbracket \implies M \vdash t$ "  
 proof (induction t rule: intruder\_deduct.induct)  
 case (Decompose  $M' t K T t_i$ )  
 thus ?case using intruder\_deduct.Decompose[OF  $\langle \text{Ana } t = (K, T) \rangle \_ \langle t_i \in \text{set } T \rangle$ ] by simp  
 qed (auto simp add: decomp\_closure\_def)

lemma deduct\_if\_closure\_synth: " $\llbracket \text{decomp\_closure } M M'; M' \vdash_c t \rrbracket \implies M \vdash t$ "  
 using deduct\_if\_closure\_deduct by blast

lemma decomp\_closure\_subterms\_composable:  
 assumes "decomp\_closure  $M M'$ "  
 and " $M' \vdash_c t'$ " " $M' \vdash t$ " " $t \sqsubseteq t'$ "  
 shows " $M' \vdash_c t$ "  
 using  $\langle M' \vdash_c t' \rangle$  assms  
 proof (induction t' rule: intruder\_synth.induct)  
 case (AxiomC  $t' M'$ )  
 have " $M \vdash t$ " using  $\langle M' \vdash t \rangle$  deduct\_if\_closure\_deduct AxiomC.prem1 by blast  
 moreover  
 { have " $\exists s \in M. t' \sqsubseteq s$ " using  $\langle t' \in M' \rangle$  AxiomC.prem1 unfolding decomp\_closure\_def by blast  
 hence " $\exists s \in M. t \sqsubseteq s$ " using  $\langle t \sqsubseteq t' \rangle$  term.order\_trans by auto  
 }  
 ultimately have " $t \in M$ " using AxiomC.prem1 unfolding decomp\_closure\_def by blast  
 thus ?case by simp  
 next  
 case (ComposeC  $T f M'$ )  
 let ?t' = "Fun f T"  
 { assume " $t = ?t'$ " have " $M' \vdash_c t$ " using  $\langle M' \vdash_c ?t' \rangle \langle t = ?t' \rangle$  by simp }  
 moreover  
 { assume " $t \neq ?t'$ "  
 have " $\exists x \in \text{set } T. t \sqsubseteq x$ " using  $\langle t \sqsubseteq ?t' \rangle \langle t \neq ?t' \rangle$  by simp  
 hence " $M' \vdash_c t$ " using ComposeC.IH ComposeC.prem1,3 ComposeC.hyps(3) by blast  
 }  
 ultimately show ?case using cases\_simp[of " $t = ?t'$ " " $M' \vdash_c t$ "] by simp  
 qed

lemma decomp\_closure\_analyzed:  
 assumes "decomp\_closure  $M M'$ "  
 shows "analyzed  $M'$ "  
 proof -  
 { fix t assume " $M' \vdash t$ " have " $M' \vdash_c t$ " using  $\langle M' \vdash t \rangle$  assms  
 proof (induction t rule: intruder\_deduct.induct)  
 case (Decompose  $M' t K T t_i$ )  
 hence " $M' \vdash t_i$ " using Decompose.hyps intruder\_deduct.Decompose by blast  
 moreover have " $t_i \sqsubseteq t$ "  
 using Decompose.hyps(4) Ana\_subterm[OF Decompose.hyps(2)] by blast  
 moreover have " $M' \vdash_c t$ " using Decompose.IH(1) Decompose.prem1 by blast  
 ultimately show " $M' \vdash_c t_i$ " using decomp\_closure\_subterms\_composable Decompose.prem1 by blast  
 qed auto  
 }  
 moreover have " $\forall t. M \vdash_c t \longrightarrow M \vdash t$ " by auto  
 ultimately show ?thesis by (auto simp add: decomp\_closure\_def analyzed\_def)  
 qed

lemma analyzed\_if\_all\_analyzed\_in:  
 assumes  $M: \forall t \in M. \text{analyzed\_in } t M$

```

shows "analyzed M"
proof (unfold analyzed_def, intro allI iffI)
  fix t
  assume t: "M ⊢ t"
  thus "M ⊢c t"
  proof (induction t rule: intruder_deduct_induct)
    case (Decompose t K T ti)
    { assume "t ∈ M"
      hence ?case
        using M Decompose.IH(2) Decompose.hyps(2,4)
        unfolding analyzed_in_def by fastforce
    } moreover {
      fix f S assume "t = Fun f S" "⋀s. s ∈ set S ⇒ M ⊢c s"
      hence ?case using Ana_fun_subterm[of f S] Decompose.hyps(2,4) by blast
    } ultimately show ?case using intruder_synth.cases[OF Decompose.IH(1), of ?case] by blast
  qed simp_all
qed auto

lemma analyzed_is_all_analyzed_in:
  "(∀t ∈ M. analyzed_in t M) ⟷ analyzed M"
proof
  show "analyzed M ⇒ ∀t ∈ M. analyzed_in t M"
    unfolding analyzed_in_def analyzed_def
    by (auto intro: intruder_deduct.Decompose[OF intruder_deduct.Axiom])
qed (rule analyzed_if_all_analyzed_in)

lemma ik_has_synth_ik_closure:
  fixes M :: "('fun,'var) terms"
  shows "∃M'. (∀t. M ⊢ t ⟷ M' ⊢c t) ∧ decomp_closure M M' ∧ (finite M ⟶ finite M')"
proof -
  let ?M' = "{t. M ⊢ t ∧ (∃t' ∈ M. t ⊆ t')}"

  have M'_closes: "decomp_closure M ?M'" unfolding decomp_closure_def by simp
  hence "M ⊆ ?M'" using closure_is_superset by simp

  have "∀t. ?M' ⊢c t ⟶ M ⊢ t" using deduct_if_closure_synth[OF M'_closes] by blast
  moreover have "∀t. M ⊢ t ⟶ ?M' ⊢ t" using ideduct_mono[OF _ ⟨M ⊆ ?M'⟩] by simp
  moreover have "analyzed ?M'" using decomp_closure_analyzed[OF M'_closes] .
  ultimately have "∀t. M ⊢ t ⟷ ?M' ⊢c t" unfolding analyzed_def by blast
  moreover have "finite M ⟶ finite ?M'" by auto
  ultimately show ?thesis using M'_closes by blast
qed

```

### 2.4.8 Intruder Variants: Numbered and Composition-Restricted Intruder Deduction Relations

A variant of the intruder relation which restricts composition to only those terms that satisfy a given predicate  $Q$ .

```

inductive intruder_deduct_restricted::
  "('fun,'var) terms ⇒ ((('fun,'var) term ⇒ bool) ⇒ ('fun,'var) term ⇒ bool)"
  ("⟨_,_⟩ ⊢r _" 50)
where
  AxiomR[simp]: "t ∈ M ⟹ ⟨M; Q⟩ ⊢r t"
  | ComposeR[simp]: "[length T = arity f; public f; ⋀t. t ∈ set T ⟹ ⟨M; Q⟩ ⊢r t; Q (Fun f T)]
    ⟹ ⟨M; Q⟩ ⊢r Fun f T"
  | DecomposeR: "[⟨M; Q⟩ ⊢r t; Ana t = (K, T); ⋀k. k ∈ set K ⟹ ⟨M; Q⟩ ⊢r k; ti ∈ set T]
    ⟹ ⟨M; Q⟩ ⊢r ti"

```

A variant of the intruder relation equipped with a number representing the height of the derivation tree (i.e.,  $\langle M; k \rangle \vdash_n t$  iff  $k$  is the maximum number of applications of the compose and decompose rules in any path of the derivation tree for  $M \vdash t$ ).

```

inductive intruder_deduct_num::

```

```

"('fun,'var) terms  $\Rightarrow$  nat  $\Rightarrow$  ('fun,'var) term  $\Rightarrow$  bool"
("<_> _>  $\vdash_n$  _" 50)
where
  AxiomN[simp]: "t  $\in$  M  $\Rightarrow$  <M; 0>  $\vdash_n$  t"
  | ComposeN[simp]: "[length T = arity f; public f;  $\bigwedge t. t \in \text{set } T \Rightarrow$  <M; steps t>  $\vdash_n$  t]
 $\Rightarrow$  <M; Suc (Max (insert 0 (steps ' set T)))>  $\vdash_n$  Fun f T"
  | DecomposeN: "[<M; n>  $\vdash_n$  t; Ana t = (K, T);  $\bigwedge k. k \in \text{set } K \Rightarrow$  <M; steps k>  $\vdash_n$  k; ti  $\in$  set T]
 $\Rightarrow$  <M; Suc (Max (insert n (steps ' set K)))>  $\vdash_n$  ti"

lemma intruder_deduct_restricted_induct[consumes 1, case_names AxiomR ComposeR DecomposeR]:
  assumes "<M; Q>  $\vdash_r$  t" " $\bigwedge t. t \in M \Rightarrow P M Q t$ "
  " $\bigwedge T f. [length T = arity f; public f;
 $\bigwedge t. t \in \text{set } T \Rightarrow$  <M; Q>  $\vdash_r$  t;
 $\bigwedge t. t \in \text{set } T \Rightarrow P M Q t$ ; Q (Fun f T)
]  $\Rightarrow P M Q$  (Fun f T)"
  " $\bigwedge K T T t_i. [<M; Q> \vdash_r t$ ;  $P M Q t$ ; Ana t = (K, T);  $\bigwedge k. k \in \text{set } K \Rightarrow$  <M; Q>  $\vdash_r$  k;
 $\bigwedge k. k \in \text{set } K \Rightarrow P M Q k$ ; ti  $\in$  set T]  $\Rightarrow P M Q t_i$ "
  shows "P M Q t"
using assms by (induct t rule: intruder_deduct_restricted.induct) blast+

lemma intruder_deduct_num_induct[consumes 1, case_names AxiomN ComposeN DecomposeN]:
  assumes "<M; n>  $\vdash_n$  t" " $\bigwedge t. t \in M \Rightarrow P M 0 t$ "
  " $\bigwedge T f \text{ steps. } [length T = arity f; public f;
 $\bigwedge t. t \in \text{set } T \Rightarrow$  <M; steps t>  $\vdash_n$  t;
 $\bigwedge t. t \in \text{set } T \Rightarrow P M (\text{steps } t) t$ ]
 $\Rightarrow P M$  (Suc (Max (insert 0 (steps ' set T)))) (Fun f T)"
  " $\bigwedge t K T t_i \text{ steps } n. [<M; n> \vdash_n t$ ;  $P M n t$ ; Ana t = (K, T);
 $\bigwedge k. k \in \text{set } K \Rightarrow$  <M; steps k>  $\vdash_n$  k;
ti  $\in$  set T;  $\bigwedge k. k \in \text{set } K \Rightarrow P M (\text{steps } k) k$ ]
 $\Rightarrow P M$  (Suc (Max (insert n (steps ' set K)))) ti"
  shows "P M n t"
using assms by (induct rule: intruder_deduct_num.induct) blast+

lemma ideduct_restricted_mono:
  "[<M; P>  $\vdash_r$  t; M  $\subseteq M'$ ]  $\Rightarrow$  <M'; P>  $\vdash_r$  t"
proof (induction rule: intruder_deduct_restricted_induct)
  case (DecomposeR t K T ti)
  have " $\forall k. k \in \text{set } K \rightarrow$  <M'; P>  $\vdash_r$  k" using DecomposeR.IH <M  $\subseteq$  M'> by simp
  moreover have "<M'; P>  $\vdash_r$  t" using DecomposeR.IH <M  $\subseteq$  M'> by simp
  ultimately show ?case
  using DecomposeR
  intruder_deduct_restricted.DecomposeR[OF _ DecomposeR.hyps(2) _ DecomposeR.hyps(4)]
  by blast
qed auto$$ 
```

## 2.4.9 Lemmata: Intruder Deduction Equivalences

```

lemma deduct_if_restricted_deduct: "<M;P>  $\vdash_r$  m  $\Rightarrow$  M  $\vdash$  m"
proof (induction m rule: intruder_deduct_restricted_induct)
  case (DecomposeR t K T ti) thus ?case using intruder_deduct.Decompose by blast
qed simp_all

lemma restricted_deduct_if_restricted_ik:
  assumes "<M;P>  $\vdash_r$  m" " $\forall m \in M. P m$ "
  and P: " $\forall t t'. P t \rightarrow t' \sqsubseteq t \rightarrow P t'$ "
  shows "P m"
using assms(1)
proof (induction m rule: intruder_deduct_restricted_induct)
  case (DecomposeR t K T ti)
  obtain f S where "t = Fun f S" using Ana_var <ti  $\in$  set T> <Ana t = (K, T)> by (cases t) auto
  thus ?case using DecomposeR assms(2) P Ana_subterm by blast

```

```

qed (simp_all add: assms(2))

lemma deduct_restricted_if_synth:
  assumes P: "P m" "∀ t t'. P t ⟶ t' ⊆ t ⟶ P t'"
  and m: "M ⊢c m"
  shows "⟨M; P⟩ ⊢r m"
using m P(1)
proof (induction m rule: intruder_synth_induct)
  case (ComposeC T f)
  hence "⟨M; P⟩ ⊢r t" when t: "t ∈ set T" for t
    using t P(2) subtermEqI''[of _ T f]
    by fastforce
  thus ?case
    using intruder_deduct_restricted.ComposeR[OF ComposeC.hyps(1,2)] ComposeC.prem(1)
    by metis
qed simp

lemma deduct_zero_in_ik:
  assumes "⟨M; 0⟩ ⊢n t" shows "t ∈ M"
proof -
  { fix k assume "⟨M; k⟩ ⊢n t" hence "k > 0 ∨ t ∈ M" by (induct t) auto
  } thus ?thesis using assms by auto
qed

lemma deduct_if_deduct_num: "⟨M; k⟩ ⊢n t ⟹ M ⊢ t"
by (induct t rule: intruder_deduct_num_induct)
  (metis intruder_deduct.Axiom,
   metis intruder_deduct.Compose,
   metis intruder_deduct.Decompose)

lemma deduct_num_if_deduct: "M ⊢ t ⟹ ∃ k. ⟨M; k⟩ ⊢n t"
proof (induction t rule: intruder_deduct_induct)
  case (Compose T f)
  then obtain steps where *: "∀ t ∈ set T. ⟨M; steps t⟩ ⊢n t" by moura
  then obtain n where "∀ t ∈ set T. steps t ≤ n"
    using finite_nat_set_iff_bounded_le[of "steps ' set T"]
    by auto
  thus ?case using ComposeN[OF Compose.hyps(1,2), of M steps] * by force
next
  case (Decompose t K T ti)
  hence "⋀ u. u ∈ insert t (set K) ⟹ ∃ k. ⟨M; k⟩ ⊢n u" by auto
  then obtain steps where *: "⟨M; steps t⟩ ⊢n t" "∀ t ∈ set K. ⟨M; steps t⟩ ⊢n t" by moura
  then obtain n where "steps t ≤ n" "∀ t ∈ set K. steps t ≤ n"
    using finite_nat_set_iff_bounded_le[of "steps ' insert t (set K)"]
    by auto
  thus ?case using DecomposeN[OF _ Decompose.hyps(2) _ Decompose.hyps(4), of M _ steps] * by force
qed (metis AxiomN)

lemma deduct_normalize:
  assumes M: "∀ m ∈ M. ∀ f T. Fun f T ⊆ m ⟶ P f T"
  and t: "⟨M; k⟩ ⊢n t" "Fun f T ⊆ t" "¬ P f T"
  shows "∃ l ≤ k. (⟨M; l⟩ ⊢n Fun f T) ∧ (∀ t ∈ set T. ∃ j < l. ⟨M; j⟩ ⊢n t)"
using t
proof (induction t rule: intruder_deduct_num_induct)
  case (AxiomN t) thus ?case using M by auto
next
  case (ComposeN T' f' steps) thus ?case
  proof (cases "Fun f' T' = Fun f T")
    case True
    hence "⟨M; Suc (Max (insert 0 (steps ' set T')))) ⊢n Fun f T" "T = T'"
      using intruder_deduct_num.ComposeN[OF ComposeN.hyps] by auto
    moreover have "⋀ t. t ∈ set T ⟹ ⟨M; steps t⟩ ⊢n t"
      using True ComposeN.hyps(3) by auto
  end
end

```

```

    moreover have " $\bigwedge t. t \in \text{set } T \implies \text{steps } t < \text{Suc } (\text{Max } (\text{insert } 0 (\text{steps ' set } T)))$ "
      using Max_less_iff[of "insert 0 (steps ' set T)" "Suc (Max (insert 0 (steps ' set T)))"]
      by auto
    ultimately show ?thesis by auto
  next
    case False
    then obtain t' where t': " $t' \in \text{set } T'$ " "Fun f T  $\sqsubseteq$  t'" using ComposeN by auto
    hence " $\exists l \leq \text{steps } t'. (\langle M; l \rangle \vdash_n \text{Fun } f T) \wedge (\forall t \in \text{set } T. \exists j < l. \langle M; j \rangle \vdash_n t)$ "
      using ComposeN.IH[OF _ _ ComposeN.prem(2)] by auto
    moreover have " $\text{steps } t' < \text{Suc } (\text{Max } (\text{insert } 0 (\text{steps ' set } T')))$ "
      using Max_less_iff[of "insert 0 (steps ' set T')" "Suc (Max (insert 0 (steps ' set T')))]"
      using t'(1) by auto
    ultimately show ?thesis using ComposeN.hyps(3)[OF t'(1)]
      by (meson Suc_le_eq le_Suc_eq le_trans)
  qed
next
  case (DecomposeN t K T' t_i steps n)
  hence *: "Fun f T  $\sqsubseteq$  t"
    using term.order_trans[of "Fun f T" t_i t] Ana_subterm[of t K T']
    by blast
  have " $\exists l \leq n. (\langle M; l \rangle \vdash_n \text{Fun } f T) \wedge (\forall t' \in \text{set } T. \exists j < l. \langle M; j \rangle \vdash_n t')$ "
    using DecomposeN.IH(1)[OF * DecomposeN.prem(2)] by auto
  moreover have " $n < \text{Suc } (\text{Max } (\text{insert } n (\text{steps ' set } K)))$ "
    using Max_less_iff[of "insert n (steps ' set K)" "Suc (Max (insert n (steps ' set K)))]"
    by auto
  ultimately show ?case using DecomposeN.hyps(4) by (meson Suc_le_eq le_Suc_eq le_trans)
qed

lemma deduct_inv:
  assumes " $\langle M; n \rangle \vdash_n t$ "
  shows " $t \in M \vee$ 
    ( $\exists f T. t = \text{Fun } f T \wedge \text{public } f \wedge \text{length } T = \text{arity } f \wedge (\forall t \in \text{set } T. \exists l < n. \langle M; l \rangle \vdash_n t)$ )
 $\vee$ 
    ( $\exists m \in \text{subterms}_{\text{set } M}. (\exists l < n. \langle M; l \rangle \vdash_n m) \wedge (\forall k \in \text{set } (\text{fst } (\text{Ana } m)). \exists l < n. \langle M; l \rangle \vdash_n k) \wedge$ 
       $t \in \text{set } (\text{snd } (\text{Ana } m)))$ "
    (is "?P t n  $\vee$  ?Q t n  $\vee$  ?R t n")
using assms
proof (induction n arbitrary: t rule: nat_less_induct)
  case (1 n t) thus ?case
  proof (cases n)
    case 0
    hence "t  $\in M$ " using deduct_zero_in_ik "1.prem(1)" by metis
    thus ?thesis by auto
  next
    case (Suc n')
    hence " $\langle M; \text{Suc } n' \rangle \vdash_n t$ "
      " $\forall m < \text{Suc } n'. \forall x. (\langle M; m \rangle \vdash_n x) \longrightarrow ?P x m \vee ?Q x m \vee ?R x m$ "
      using "1.prem" "1.IH" by blast+
    hence "?P t (Suc n')  $\vee$  ?Q t (Suc n')  $\vee$  ?R t (Suc n')"
    proof (induction t rule: intruder_deduct_num_induct)
      case (AxiomN t) thus ?case by simp
    next
      case (ComposeN T f steps)
      have " $\bigwedge t. t \in \text{set } T \implies \text{steps } t < \text{Suc } (\text{Max } (\text{insert } 0 (\text{steps ' set } T)))$ "
        using Max_less_iff[of "insert 0 (steps ' set T)" "Suc (Max (insert 0 (steps ' set T)))]"
        by auto
      thus ?case using ComposeN.hyps by metis
    next
      case (DecomposeN t K T t_i steps n)
      have 0: " $n < \text{Suc } (\text{Max } (\text{insert } n (\text{steps ' set } K)))$ "
        " $\bigwedge k. k \in \text{set } K \implies \text{steps } k < \text{Suc } (\text{Max } (\text{insert } n (\text{steps ' set } K)))$ "
        using Max_less_iff[of "insert n (steps ' set K)" "Suc (Max (insert n (steps ' set K)))]"

```



```

by auto

have IH1: "?P t j ∨ ?Q t j ∨ ?R t j" when jt: "j < n" "<M; j> ⊢n t" for j t
  using jt DecomposeN.prem1 0(1)
  by simp

have IH2: "?P t n ∨ ?Q t n ∨ ?R t n"
  using DecomposeN.IH(1) IH1
  by simp

have 1: "∀k ∈ set (fst (Ana t)). ∃l < Suc (Max (insert n (steps ' set K))). <M; l> ⊢n k"
  using DecomposeN.hyps(1,2,3) 0(2)
  by auto

have 2: "ti ∈ set (snd (Ana t))"
  using DecomposeN.hyps(2,4)
  by fastforce

have 3: "t ∈ subtermsset M" when "t ∈ set (snd (Ana m))" "m ⊆set M" for m
  using that(1) Ana_subterm[of m _ "snd (Ana m)"] in_subterms_subset_Union[OF that(2)]
  by (metis (no_types, lifting) prod.collapse psubsetD subsetCE subsetD)

have 4: "?R ti (Suc (Max (insert n (steps ' set K))))" when "?R t n"
  using that 0(1) 1 2 3 DecomposeN.hyps(1)
  by (metis (no_types, lifting))

have 5: "?R ti (Suc (Max (insert n (steps ' set K))))" when "?P t n"
  using that 0(1) 1 2 DecomposeN.hyps(1)
  by blast

have 6: ?case when *: "?Q t n"
proof -
  obtain g S where g:
    "t = Fun g S" "public g" "length S = arity g" "∀t ∈ set S. ∃l < n. <M; l> ⊢n t"
  using * by mouna
  then obtain l where l: "l < n" "<M; l> ⊢n ti"
  using 0(1) DecomposeN.hyps(2,4) Ana_fun_subterm[of g S K T] by blast

  have **: "l < Suc (Max (insert n (steps ' set K)))" using l(1) 0(1) by simp

  show ?thesis using IH1[OF l] less_trans[OF _ **] by fastforce
qed

show ?case using IH2 4 5 6 by argo
qed
thus ?thesis using Suc by fast
qed
qed

lemma restricted_deduct_if_deduct:
  assumes M: "∀m ∈ M. ∀f T. Fun f T ⊆ m ⟶ P (Fun f T)"
  and P_subterm: "∀f T t. M ⊢ Fun f T ⟶ P (Fun f T) ⟶ t ∈ set T ⟶ P t"
  and P_Ana_key: "∀t K T k. M ⊢ t ⟶ P t ⟶ Ana t = (K, T) ⟶ M ⊢ k ⟶ k ∈ set K ⟶ P k"
  and m: "M ⊢ m" "P m"
  shows "<M; P> ⊢r m"
proof -
  { fix k assume "<M; k> ⊢n m"
    hence ?thesis using m(2)
  }
  proof (induction k arbitrary: m rule: nat_less_induct)
    case (1 n m) thus ?case
    proof (cases n)
      case 0
      hence "m ∈ M" using deduct_zero_in_ik "1.prem1"(1) by metis
    end
  end
end

```

```

thus ?thesis by auto
next
case (Suc n')
hence " $\langle M; \text{Suc } n' \rangle \vdash_n m$ "
      " $\forall m < \text{Suc } n'. \forall x. (\langle M; m \rangle \vdash_n x) \longrightarrow P x \longrightarrow \langle M; P \rangle \vdash_r x$ "
      using "1.prem" "1.IH" by blast+
thus ?thesis using "1.prem"(2)
proof (induction m rule: intruder_deduct_num_induct)
  case (ComposeN T f steps)
  have *: "steps t < Suc (Max (insert 0 (steps ' set T)))" when "t ∈ set T" for t
    using Max_less_iff[of "insert 0 (steps ' set T)"] that
    by blast

  have **: "P t" when "t ∈ set T" for t
    using P_subterm ComposeN.prem(2) that
      Fun_param_is_subterm[OF that]
      intruder_deduct.Compose[OF ComposeN.hyps(1,2)]
      deduct_if_deduct_num[OF ComposeN.hyps(3)]
    by blast

  have " $\langle M; P \rangle \vdash_r t$ " when "t ∈ set T" for t
    using ComposeN.prem(1) ComposeN.hyps(3)[OF that] *[OF that] **[OF that]
    by blast
  thus ?case
    by (metis intruder_deduct_restricted.ComposeR[OF ComposeN.hyps(1,2)] ComposeN.prem(2))
next
case (DecomposeN t K T ti steps l)
show ?case
proof (cases "P t")
  case True
  hence " $\bigwedge k. k \in \text{set } K \implies P k$ "
    using P_Ana_key DecomposeN.hyps(1,2,3) deduct_if_deduct_num
    by blast
  moreover have
    " $\bigwedge k m x. k \in \text{set } K \implies m < \text{steps } k \implies \langle M; m \rangle \vdash_n x \implies P x \implies \langle M; P \rangle \vdash_r x$ "
  proof -
    fix k m x assume *: "k ∈ set K" "m < steps k" " $\langle M; m \rangle \vdash_n x$ " "P x"
    have "steps k ∈ insert 1 (steps ' set K)" using *(1) by simp
    hence "m < Suc (Max (insert 1 (steps ' set K)))"
      using less_trans[OF *(2), of "Suc (Max (insert 1 (steps ' set K)))"]
      Max_less_iff[of "insert 1 (steps ' set K)"]
      "Suc (Max (insert 1 (steps ' set K)))"
    by auto
    thus " $\langle M; P \rangle \vdash_r x$ " using DecomposeN.prem(1) *(3,4) by simp
  qed
  ultimately have " $\bigwedge k. k \in \text{set } K \implies \langle M; P \rangle \vdash_r k$ "
    using DecomposeN.IH(2) by auto
  moreover have " $\langle M; P \rangle \vdash_r t$ "
    using True DecomposeN.prem(1) DecomposeN.hyps(1) le_imp_less_Suc
      Max_less_iff[of "insert 1 (steps ' set K)"] "Suc (Max (insert 1 (steps ' set K)))"
    by blast
  ultimately show ?thesis
    using intruder_deduct_restricted.DecomposeR[OF _ DecomposeN.hyps(2)
      _ DecomposeN.hyps(4)]
    by metis
next
case False
obtain g S where gS: "t = Fun g S" using DecomposeN.hyps(2,4) by (cases t) moura+
hence *: "Fun g S ⊆ t" "¬P (Fun g S)" using False by force+
have "∃ j < l.  $\langle M; j \rangle \vdash_n t_i$ "
  using gS DecomposeN.hyps(2,4) Ana_fun_subterm[of g S K T]
    deduct_normalize[of M "λf T. P (Fun f T)", OF M DecomposeN.hyps(1) *]
  by force

```

```

    hence "∃ j < Suc (Max (insert 1 (steps ' set K))). ⟨M; j⟩ ⊢n ti"
      using Max_less_iff[of "insert 1 (steps ' set K)"
        "Suc (Max (insert 1 (steps ' set K)))"]
        less_trans[of _ 1 "Suc (Max (insert 1 (steps ' set K)))"]
      by blast
    thus ?thesis using DecomposeN.prem1(1,2) by meson
  qed
qed auto
qed
qed
} thus ?thesis using deduct_num_if_deduct m(1) by metis
qed

lemma restricted_deduct_if_deduct':
  assumes "∀ m ∈ M. P m"
    and "∀ t t'. P t ⟶ t' ⊆ t ⟶ P t'"
    and "∀ t K T k. P t ⟶ Ana t = (K, T) ⟶ k ∈ set K ⟶ P k"
    and "M ⊢ m" "P m"
  shows "⟨M; P⟩ ⊢r m"
using restricted_deduct_if_deduct[of M P m] assms
by blast

lemma private_const_deduct:
  assumes c: "¬public c" "M ⊢ (Fun c []::('fun,'var) term)"
  shows "Fun c [] ∈ M ∨
    (∃ m ∈ subtermsset M. M ⊢ m ∧ (∀ k ∈ set (fst (Ana m)). M ⊢ m) ∧
      Fun c [] ∈ set (snd (Ana m)))"

proof -
  obtain n where "⟨M; n⟩ ⊢n Fun c []"
    using c(2) deduct_num_if_deduct by mouna
  hence "Fun c [] ∈ M ∨
    (∃ m ∈ subtermsset M.
      (∃ l < n. ⟨M; l⟩ ⊢n m) ∧
      (∀ k ∈ set (fst (Ana m)). ∃ l < n. ⟨M; l⟩ ⊢n k) ∧ Fun c [] ∈ set (snd (Ana m)))"
    using deduct_inv[of M n "Fun c []"] c(1) by fast
  thus ?thesis using deduct_if_deduct_num[of M] by blast
qed

lemma private_fun_deduct_in_ik'':
  assumes t: "M ⊢ Fun f T" "Fun c [] ∈ set T" "∀ m ∈ subtermsset M. Fun f T ∉ set (snd (Ana m))"
    and c: "¬public c" "Fun c [] ∉ M" "∀ m ∈ subtermsset M. Fun c [] ∉ set (snd (Ana m))"
  shows "Fun f T ∈ M"

proof -
  have *: "∄ n. ⟨M; n⟩ ⊢n Fun c []"
    using private_const_deduct[OF c(1)] c(2,3) deduct_if_deduct_num
    by blast

  obtain n where n: "⟨M; n⟩ ⊢n Fun f T"
    using t(1) deduct_num_if_deduct
    by blast

  show ?thesis
    using deduct_inv[OF n] t(2,3) *
    by blast
qed

end

```

### 2.4.10 Executable Definitions for Code Generation

```

fun intruder_synth' where
  "intruder_synth' pu ar M (Var x) = (Var x ∈ M)"
| "intruder_synth' pu ar M (Fun f T) = (

```

```

Fun f T ∈ M ∨ (pu f ∧ length T = ar f ∧ list_all (intruder_synth' pu ar M) T))"

definition "wftrm' ar t ≡ (∀ s ∈ subterms t. is_Fun s → ar (the_Fun s) = length (args s))"

definition "wftrms' ar M ≡ (∀ t ∈ M. wftrm' ar t)"

definition "analyzed_in' An pu ar t M ≡ (case An t of
  (K,T) ⇒ (∀ k ∈ set K. intruder_synth' pu ar M k) → (∀ s ∈ set T. intruder_synth' pu ar M s))"

lemma (in intruder_model) intruder_synth'_induct[consumes 1, case_names Var Fun]:
  assumes "intruder_synth' public arity M t"
    "∧x. intruder_synth' public arity M (Var x) ⇒ P (Var x)"
    "∧f T. (∧z. z ∈ set T ⇒ intruder_synth' public arity M z ⇒ P z) ⇒
      intruder_synth' public arity M (Fun f T) ⇒ P (Fun f T) "
  shows "P t"
using assms by (induct public arity M t rule: intruder_synth'.induct) auto

lemma (in intruder_model) wftrm_code[code_unfold]:
  "wftrm t = wftrm' arity t"
unfolding wftrm_def wftrm'_def
by auto

lemma (in intruder_model) wftrms_code[code_unfold]:
  "wftrms M = wftrms' arity M"
using wftrm_code
unfolding wftrms'_def
by auto

lemma (in intruder_model) intruder_synth_code[code_unfold]:
  "intruder_synth M t = intruder_synth' public arity M t"
(is "?A ↔ ?B")
proof
  show "?A ⇒ ?B"
  proof (induction t rule: intruder_synth_induct)
    case (AxiomC t) thus ?case by (cases t) auto
  qed (fastforce simp add: list_all_iff)

  show "?B ⇒ ?A"
  proof (induction t rule: intruder_synth'_induct)
    case (Fun f T) thus ?case
    proof (cases "Fun f T ∈ M")
      case False
      hence "public f" "length T = arity f" "list_all (intruder_synth' public arity M) T"
      using Fun.hyps by fastforce+
      thus ?thesis
      using Fun.IH intruder_synth.ComposeC[of T f M] Ball_set[of T]
      by blast
    qed simp
  qed simp
qed

lemma (in intruder_model) analyzed_in_code[code_unfold]:
  "analyzed_in t M = analyzed_in' Ana public arity t M"
using intruder_synth_code[of M]
unfolding analyzed_in_def analyzed_in'_def
by fastforce

end

```

## 3 The Typing Result for Non-Stateful Protocols

In this chapter, we formalize and prove a typing result for “stateless” security protocols. This work is described in more detail in [2] and [1, chapter 3].

### 3.1 Strands and Symbolic Intruder Constraints (Strands\_and\_Constraints)

```
theory Strands_and_Constraints
imports Messages More_Unification Intruder_Deduction
begin
```

#### 3.1.1 Constraints, Strands and Related Definitions

```
datatype poscheckvariant = Assign ("assign") | Check ("check")
```

A strand (or constraint) step is either a message transmission (either a message being sent *Send* or being received *Receive*) or a check on messages (a positive check *Equality*—which can be either an “assignment” or just a check—or a negative check *Inequality*)

```
datatype (funstp: 'a, varstp: 'b) strand_step =
  Send      "('a,'b) term" ("send⟨_⟩st" 80)
| Receive   "('a,'b) term" ("receive⟨_⟩st" 80)
| Equality   poscheckvariant "('a,'b) term" "('a,'b) term" ("⟨_: _ ≐ _⟩st" [80,80])
| Inequality (bvarstp: "'b list") ("⟨('a,'b) term × ('a,'b) term⟩ list" ("∀_⟨_≠: _⟩st" [80,80]))
where
  "bvarstp (Send _) = []"
| "bvarstp (Receive _) = []"
| "bvarstp (Equality _ _ _) = []"
```

A strand is a finite sequence of strand steps (constraints and strands share the same datatype)

```
type_synonym ('a,'b) strand = "('a,'b) strand_step list"
```

```
type_synonym ('a,'b) strands = "('a,'b) strand set"
```

```
abbreviation "trmspairs F ≡ ⋃ (t,t') ∈ set F. {t,t'}"
```

```
fun trmsstp:: "('a,'b) strand_step ⇒ ('a,'b) terms" where
  "trmsstp (Send t) = {t}"
| "trmsstp (Receive t) = {t}"
| "trmsstp (Equality _ t t') = {t,t'}"
| "trmsstp (Inequality _ F) = trmspairs F"
```

```
lemma varsstp_unfold[simp]: "varsstp x = fvset (trmsstp x) ∪ set (bvarstp x)"
by (cases x) auto
```

The set of terms occurring in a strand

```
definition trmsst where "trmsst S ≡ ⋃ (trmsstp ' set S)"
```

```
fun trms_liststp:: "('a,'b) strand_step ⇒ ('a,'b) term list" where
  "trms_liststp (Send t) = [t]"
| "trms_liststp (Receive t) = [t]"
| "trms_liststp (Equality _ t t') = [t,t']"
| "trms_liststp (Inequality _ F) = concat (map (λ(t,t'). [t,t']) F)"
```

The set of terms occurring in a strand (list variant)

```
definition trms_listst where "trms_listst S ≡ remdups (concat (map trms_liststp S))"
```

The set of variables occurring in a sent message

**definition**  $fv_{snd}::('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{snd} \ x \equiv \text{case } x \text{ of Send } t \Rightarrow fv \ t \mid \_ \Rightarrow \{\}$

The set of variables occurring in a received message

**definition**  $fv_{rcv}::('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{rcv} \ x \equiv \text{case } x \text{ of Receive } t \Rightarrow fv \ t \mid \_ \Rightarrow \{\}$

The set of variables occurring in an equality constraint

**definition**  $fv_{eq}::\text{poscheckvariant} \Rightarrow ('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{eq} \ ac \ x \equiv \text{case } x \text{ of Equality } ac' \ s \ t \Rightarrow \text{if } ac = ac' \text{ then } fv \ s \cup fv \ t \text{ else } \{\} \mid \_ \Rightarrow \{\}$

The set of variables occurring at the left-hand side of an equality constraint

**definition**  $fv_{leq}::\text{poscheckvariant} \Rightarrow ('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{leq} \ ac \ x \equiv \text{case } x \text{ of Equality } ac' \ s \ t \Rightarrow \text{if } ac = ac' \text{ then } fv \ s \text{ else } \{\} \mid \_ \Rightarrow \{\}$

The set of variables occurring at the right-hand side of an equality constraint

**definition**  $fv_{req}::\text{poscheckvariant} \Rightarrow ('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{req} \ ac \ x \equiv \text{case } x \text{ of Equality } ac' \ s \ t \Rightarrow \text{if } ac = ac' \text{ then } fv \ t \text{ else } \{\} \mid \_ \Rightarrow \{\}$

The free variables of inequality constraints

**definition**  $fv_{ineq}::('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{ineq} \ x \equiv \text{case } x \text{ of Inequality } X \ F \Rightarrow fv_{pairs} \ F - \text{set } X \mid \_ \Rightarrow \{\}$

**fun**  $fv_{stp}::('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{stp} \ (\text{Send } t) = fv \ t$   
 $\mid fv_{stp} \ (\text{Receive } t) = fv \ t$   
 $\mid fv_{stp} \ (\text{Equality } \_ \ t \ t') = fv \ t \cup fv \ t'$   
 $\mid fv_{stp} \ (\text{Inequality } X \ F) = (\bigcup (t, t') \in \text{set } F. fv \ t \cup fv \ t') - \text{set } X$

The set of free variables of a strand

**definition**  $fv_{st}::('a, 'b) \text{strand} \Rightarrow 'b \text{ set}$  **where**  
 $fv_{st} \ S \equiv \bigcup (\text{set } (\text{map } fv_{stp} \ S))$

The set of bound variables of a strand

**definition**  $bvars_{st}::('a, 'b) \text{strand} \Rightarrow 'b \text{ set}$  **where**  
 $bvars_{st} \ S \equiv \bigcup (\text{set } (\text{map } (\text{set} \circ bvars_{stp}) \ S))$

The set of all variables occurring in a strand

**definition**  $vars_{st}::('a, 'b) \text{strand} \Rightarrow 'b \text{ set}$  **where**  
 $vars_{st} \ S \equiv \bigcup (\text{set } (\text{map } vars_{stp} \ S))$

**abbreviation**  $wfrestrictedvars_{stp}::('a, 'b) \text{strand\_step} \Rightarrow 'b \text{ set}$  **where**  
 $wfrestrictedvars_{stp} \ x \equiv$   
 $\text{case } x \text{ of Inequality } \_ \_ \Rightarrow \{\} \mid \text{Equality Check } \_ \_ \Rightarrow \{\} \mid \_ \Rightarrow vars_{stp} \ x$

The variables of a strand whose occurrences might be restricted by well-formedness constraints

**definition**  $wfrestrictedvars_{st}::('a, 'b) \text{strand} \Rightarrow 'b \text{ set}$  **where**  
 $wfrestrictedvars_{st} \ S \equiv \bigcup (\text{set } (\text{map } wfrestrictedvars_{stp} \ S))$

**abbreviation**  $wfvarsoccs_{stp}$  **where**  
 $wfvarsoccs_{stp} \ x \equiv \text{case } x \text{ of Send } t \Rightarrow fv \ t \mid \text{Equality Assign } s \ t \Rightarrow fv \ s \mid \_ \Rightarrow \{\}$

The variables of a strand that occur in sent messages or in assignments

**definition**  $wfvarsoccs_{st}$  **where**  
 $wfvarsoccs_{st} \ S \equiv \bigcup (\text{set } (\text{map } wfvarsoccs_{stp} \ S))$

The variables occurring at the right-hand side of assignment steps

**fun**  $\text{assignment\_rhs}_{st}$  **where**  
 $\text{assignment\_rhs}_{st} \ [] = \{\}$   
 $\mid \text{assignment\_rhs}_{st} \ (\text{Equality Assign } t \ t' \# S) = \text{insert } t' \ (\text{assignment\_rhs}_{st} \ S)$

/ "assignment\_rhs<sub>st</sub> (x#S) = assignment\_rhs<sub>st</sub> S"

The set of function symbols occurring in a strand

**definition** funs<sub>st</sub>:: "('a,'b) strand  $\Rightarrow$  'a set" where  
 "funs<sub>st</sub> S  $\equiv \bigcup (\text{set } (\text{map } \text{funs}_{stp} S))$ "

**fun** subst\_apply\_strand\_step:: "('a,'b) strand\_step  $\Rightarrow$  ('a,'b) subst  $\Rightarrow$  ('a,'b) strand\_step"  
 (infix ".<sub>stp</sub>" 51) where  
 "Send t .<sub>stp</sub>  $\vartheta$  = Send (t .  $\vartheta$ )"  
 / "Receive t .<sub>stp</sub>  $\vartheta$  = Receive (t .  $\vartheta$ )"  
 / "Equality a t t' .<sub>stp</sub>  $\vartheta$  = Equality a (t .  $\vartheta$ ) (t' .  $\vartheta$ )"  
 / "Inequality X F .<sub>stp</sub>  $\vartheta$  = Inequality X (F .<sub>pairs</sub> rm\_vars (set X)  $\vartheta$ )"

Substitution application for strands

**definition** subst\_apply\_strand:: "('a,'b) strand  $\Rightarrow$  ('a,'b) subst  $\Rightarrow$  ('a,'b) strand"  
 (infix ".<sub>st</sub>" 51) where  
 "S .<sub>st</sub>  $\vartheta \equiv \text{map } (\lambda x. x ._{stp} \vartheta) S$ "

The semantics of inequality constraints

**definition**  
 "ineq\_model ( $\mathcal{I}$ ::('a,'b) subst) X F  $\equiv$   
 ( $\forall \delta. \text{subst\_domain } \delta = \text{set } X \wedge \text{ground } (\text{subst\_range } \delta) \longrightarrow$   
 $\text{list\_ex } (\lambda f. \text{fst } f \cdot (\delta \circ_s \mathcal{I}) \neq \text{snd } f \cdot (\delta \circ_s \mathcal{I})) F$ )"

**fun** simple<sub>stp</sub> where  
 "simple<sub>stp</sub> (Receive t) = True"  
 / "simple<sub>stp</sub> (Send (Var v)) = True"  
 / "simple<sub>stp</sub> (Inequality X F) = ( $\exists \mathcal{I}. \text{ineq\_model } \mathcal{I} X F$ )"  
 / "simple<sub>stp</sub> \_ = False"

Simple constraints

**definition** simple where "simple S  $\equiv \text{list\_all simple}_{stp} S$ "

The intruder knowledge of a constraint

**fun** ik<sub>st</sub>:: "('a,'b) strand  $\Rightarrow$  ('a,'b) terms" where  
 "ik<sub>st</sub> [] = {}"  
 / "ik<sub>st</sub> (Receive t#S) = insert t (ik<sub>st</sub> S)"  
 / "ik<sub>st</sub> (\_#S) = ik<sub>st</sub> S"

Strand well-formedness

**fun** wf<sub>st</sub>:: "'b set  $\Rightarrow$  ('a,'b) strand  $\Rightarrow$  bool" where  
 "wf<sub>st</sub> V [] = True"  
 / "wf<sub>st</sub> V (Receive t#S) = (fv t  $\subseteq$  V  $\wedge$  wf<sub>st</sub> V S)"  
 / "wf<sub>st</sub> V (Send t#S) = wf<sub>st</sub> (V  $\cup$  fv t) S"  
 / "wf<sub>st</sub> V (Equality Assign s t#S) = (fv t  $\subseteq$  V  $\wedge$  wf<sub>st</sub> (V  $\cup$  fv s) S)"  
 / "wf<sub>st</sub> V (Equality Check s t#S) = wf<sub>st</sub> V S"  
 / "wf<sub>st</sub> V (Inequality \_ \_#S) = wf<sub>st</sub> V S"

Well-formedness of constraint states

**definition** wf<sub>constr</sub>:: "('a,'b) strand  $\Rightarrow$  ('a,'b) subst  $\Rightarrow$  bool" where  
 "wf<sub>constr</sub> S  $\vartheta \equiv (\text{wf}_{subst} \vartheta \wedge \text{wf}_{st} \{ \} S \wedge \text{subst\_domain } \vartheta \cap \text{vars}_{st} S = \{ \} \wedge$   
 $\text{range\_vars } \vartheta \cap \text{bvars}_{st} S = \{ \} \wedge \text{fv}_{st} S \cap \text{bvars}_{st} S = \{ \})$ "

**declare** trms<sub>st\_def</sub>[simp]  
**declare** fv<sub>snd\_def</sub>[simp]  
**declare** fv<sub>rcv\_def</sub>[simp]  
**declare** fv<sub>eq\_def</sub>[simp]  
**declare** fv<sub>l\_eq\_def</sub>[simp]  
**declare** fv<sub>r\_eq\_def</sub>[simp]  
**declare** fv<sub>ineq\_def</sub>[simp]  
**declare** fv<sub>st\_def</sub>[simp]  
**declare** vars<sub>st\_def</sub>[simp]

```

declare bvarsst_def[simp]
declare wfrestrictedvarsst_def[simp]
declare wfvarsoccsst_def[simp]

lemmas wfst_induct = wfst.induct[case_names Nil ConsRcv ConsSnd ConsEq ConsEq2 ConsIneq]
lemmas ikst_induct = ikst.induct[case_names Nil ConsRcv ConsSnd ConsEq ConsIneq]
lemmas assignment_rhsst_induct = assignment_rhsst.induct[case_names Nil ConsEq2 ConsSnd ConsRcv ConsEq ConsIneq]

```

### Lexicographical measure on strands

```

definition sizest::("a,'b) strand  $\Rightarrow$  nat" where
  "sizest S  $\equiv$  size_list ( $\lambda$ x. Max (insert 0 (size ' trmsstp x))) S"

definition measurest::("a,'b) strand  $\times$  ('a,'b) subst  $\times$  ('a,'b) strand  $\times$  ('a,'b) subst) set"
where
  "measurest  $\equiv$  measures [ $\lambda$ (S, $\vartheta$ ). card (fvst S),  $\lambda$ (S, $\vartheta$ ). sizest S]"

lemma measurest_alt_def:
  "((s,x),(t,y))  $\in$  measurest =
    (card (fvst s) < card (fvst t)  $\vee$  (card (fvst s) = card (fvst t)  $\wedge$  sizest s < sizest t))"
by (simp add: measurest_def sizest_def)

lemma measurest_trans: "trans measurest"
by (simp add: trans_def measurest_def sizest_def)

```

### Some lemmata

```

lemma trms_listst_is_trmsst: "trmsst S = set (trms_listst S)"
unfolding trmsst_def trms_listst_def
proof (induction S)
  case (Cons x S) thus ?case by (cases x) auto
qed simp

lemma subst_apply_strand_step_def:
  "s ·stp  $\vartheta$  = (case s of
    Send t  $\Rightarrow$  Send (t ·  $\vartheta$ )
  | Receive t  $\Rightarrow$  Receive (t ·  $\vartheta$ )
  | Equality a t t'  $\Rightarrow$  Equality a (t ·  $\vartheta$ ) (t' ·  $\vartheta$ )
  | Inequality X F  $\Rightarrow$  Inequality X (F ·pairs rm_vars (set X)  $\vartheta$ ))"
by (cases s) simp_all

lemma subst_apply_strand_nil[simp]: "[ ] ·st  $\delta$  = [ ]"
unfolding subst_apply_strand_def by simp

lemma finite_funsstp[simp]: "finite (funsstp x)" by (cases x) auto
lemma finite_funsst[simp]: "finite (funsst S)" unfolding funsst_def by simp
lemma finite_trmspairs[simp]: "finite (trmspairs x)" by (induct x) auto
lemma finite_trmsstp[simp]: "finite (trmsstp x)" by (cases x) auto
lemma finite_varsstp[simp]: "finite (varsstp x)" by auto
lemma finite_bvarsstp[simp]: "finite (set (bvarsstp x))" by rule
lemma finite_fvsnd[simp]: "finite (fvsnd x)" by (cases x) auto
lemma finite_fvrcv[simp]: "finite (fvrcv x)" by (cases x) auto
lemma finite_fvstp[simp]: "finite (fvstp x)" by (cases x) auto
lemma finite_varsst[simp]: "finite (varsst S)" by simp
lemma finite_bvarsst[simp]: "finite (bvarsst S)" by simp
lemma finite_fvst[simp]: "finite (fvst S)" by simp

lemma finite_wfrestrictedvarsstp[simp]: "finite (wfrestrictedvarsstp x)"
by (cases x) (auto split: poscheckvariant.splits)

lemma finite_wfrestrictedvarsst[simp]: "finite (wfrestrictedvarsst S)"
using finite_wfrestrictedvarsstp by auto

```



```

lemma finite_wfvarsoccsstp[simp]: "finite (wfvarsoccsstp x)"
by (cases x) (auto split: poscheckvariant.splits)

lemma finite_wfvarsoccsst[simp]: "finite (wfvarsoccsst S)"
using finite_wfvarsoccsstp by auto

lemma finite_ikst[simp]: "finite (ikst S)"
by (induct S rule: ikst.induct) simp_all

lemma finite_assignment_rhsst[simp]: "finite (assignment_rhsst S)"
by (induct S rule: assignment_rhsst.induct) simp_all

lemma ikst_is_rcv_set: "ikst A = {t. Receive t ∈ set A}"
by (induct A rule: ikst.induct) auto

lemma ikstD[dest]: "t ∈ ikst S ⟹ Receive t ∈ set S"
by (induct S rule: ikst.induct) auto

lemma ikstD'[dest]: "t ∈ ikst S ⟹ t ∈ trmsst S"
by (induct S rule: ikst.induct) auto

lemma ikstD''[dest]: "t ∈ subtermsset (ikst S) ⟹ t ∈ subtermsset (trmsst S)"
by (induct S rule: ikst.induct) auto

lemma ikst_subterm_exD:
  assumes "t ∈ ikst S"
  shows "∃ x ∈ set S. t ∈ subtermsset (trmsstp x)"
using assms ikstD by force

lemma assignment_rhsstD[dest]: "t ∈ assignment_rhsst S ⟹ ∃ t'. Equality Assign t' t ∈ set S"
by (induct S rule: assignment_rhsst.induct) auto

lemma assignment_rhsstD'[dest]: "t ∈ subtermsset (assignment_rhsst S) ⟹ t ∈ subtermsset (trmsst S)"
by (induct S rule: assignment_rhsst.induct) auto

lemma bvarsst_split: "bvarsst (S@S') = bvarsst S ∪ bvarsst S'"
unfolding bvarsst_def by auto

lemma bvarsst_singleton: "bvarsst [x] = set (bvarsstp x)"
unfolding bvarsst_def by auto

lemma strand_fv_bvars_disjointD:
  assumes "fvst S ∩ bvarsst S = {}" "Inequality X F ∈ set S"
  shows "set X ⊆ bvarsst S" "fvpairs F - set X ⊆ fvst S"
using assms by (induct S) fastforce+

lemma strand_fv_bvars_disjoint_unfold:
  assumes "fvst S ∩ bvarsst S = {}" "Inequality X F ∈ set S" "Inequality Y G ∈ set S"
  shows "set Y ∩ (fvpairs F - set X) = {}"
proof -
  have "set X ⊆ bvarsst S" "set Y ⊆ bvarsst S"
    "fvpairs F - set X ⊆ fvst S" "fvpairs G - set Y ⊆ fvst S"
  using strand_fv_bvars_disjointD[OF assms(1)] assms(2,3) by auto
  thus ?thesis using assms(1) by fastforce
qed

lemma strand_subst_hom[iff]:
  "(S@S') ·st ∅ = (S ·st ∅)@(S' ·st ∅)" "(x#S) ·st ∅ = (x ·stp ∅)#(S ·st ∅)"
unfolding subst_apply_strand_def by auto

lemma strand_subst_comp: "range_vars δ ∩ bvarsst S = {} ⟹ S ·st δ ∘s ∅ = ((S ·st δ) ·st ∅)"
proof (induction S)

```

```

case (Cons x S)
have *: "range_vars  $\delta \cap \text{bvars}_{st} S = \{\}$ " "range_vars  $\delta \cap (\text{set } (\text{bvars}_{stp} x)) = \{\}$ "
  using Cons_bvars_st_split[of "[x]" S] append_Cons inf_sup_absorb
  by (metis (no_types, lifting) Int_iff Un_commute disjoint_iff_not_equal self_append_conv2,
      metis append_self_conv2 bvars_st_singleton inf_bot_right inf_left_commute)
hence IH: " $S \cdot_{st} \delta \circ_s \vartheta = (S \cdot_{st} \delta) \cdot_{st} \vartheta$ " using Cons.IH by auto
have "(x#S  $\cdot_{st} \delta \circ_s \vartheta) = (x \cdot_{stp} \delta \circ_s \vartheta) \# (S \cdot_{st} \delta \circ_s \vartheta)$ " by (metis strand_subst_hom(2))
hence "... = (x  $\cdot_{stp} \delta \circ_s \vartheta) \# ((S \cdot_{st} \delta) \cdot_{st} \vartheta)$ " by (metis IH)
hence "... = ((x  $\cdot_{stp} \delta) \cdot_{stp} \vartheta) \# ((S \cdot_{st} \delta) \cdot_{st} \vartheta)$ " using rm_vars_comp[OF *(2)]
proof (induction x)
  case (Inequality X F) thus ?case
    by (induct F) (auto simp add: subst_apply_pairs_def subst_apply_strand_step_def)
qed (simp_all add: subst_apply_strand_step_def)
thus ?case using IH by auto
qed (simp add: subst_apply_strand_def)

lemma strand_substI[intro]:
  "subst_domain  $\vartheta \cap \text{fv}_{st} S = \{\}$   $\implies S \cdot_{st} \vartheta = S$ "
  "subst_domain  $\vartheta \cap \text{vars}_{st} S = \{\}$   $\implies S \cdot_{st} \vartheta = S$ "
proof -
  show "subst_domain  $\vartheta \cap \text{vars}_{st} S = \{\}$   $\implies S \cdot_{st} \vartheta = S$ "
  proof (induction S)
    case (Cons x S)
    hence " $S \cdot_{st} \vartheta = S$ " by auto
    moreover have " $\text{vars}_{stp} x \cap \text{subst\_domain } \vartheta = \{\}$ " using Cons.prem by auto
    hence " $x \cdot_{stp} \vartheta = x$ "
    proof (induction x)
      case (Inequality X F) thus ?case
        by (induct F) (force simp add: subst_apply_pairs_def)
    qed auto
    ultimately show ?case by simp
  qed (simp add: subst_apply_strand_def)

  show "subst_domain  $\vartheta \cap \text{fv}_{st} S = \{\}$   $\implies S \cdot_{st} \vartheta = S$ "
  proof (induction S)
    case (Cons x S)
    hence " $S \cdot_{st} \vartheta = S$ " by auto
    moreover have " $\text{fv}_{stp} x \cap \text{subst\_domain } \vartheta = \{\}$ "
      using Cons.prem by auto
    hence " $x \cdot_{stp} \vartheta = x$ "
    proof (induction x)
      case (Inequality X F) thus ?case
        by (induct F) (force simp add: subst_apply_pairs_def)
    qed auto
    ultimately show ?case by simp
  qed (simp add: subst_apply_strand_def)
qed

lemma strand_substI':
  " $\text{fv}_{st} S = \{\}$   $\implies S \cdot_{st} \vartheta = S$ "
  " $\text{vars}_{st} S = \{\}$   $\implies S \cdot_{st} \vartheta = S$ "
by (metis inf_bot_right strand_substI(1),
    metis inf_bot_right strand_substI(2))

lemma strand_subst_set: "(set (S  $\cdot_{st} \vartheta)) = ((\lambda x. x \cdot_{stp} \vartheta) ` (set S))"$ "
by (auto simp add: subst_apply_strand_def)

lemma strand_map_inv_set_snd_rcv_subst:
  assumes "finite (M::('a,'b) terms)"
  shows "set ((map Send (inv set M))  $\cdot_{st} \vartheta) = \text{Send } ` (M \cdot_{set} \vartheta)$ " (is ?A)
    "set ((map Receive (inv set M))  $\cdot_{st} \vartheta) = \text{Receive } ` (M \cdot_{set} \vartheta)$ " (is ?B)
proof -
  { fix f::('a,'b) term  $\Rightarrow$  ('a,'b) strand_step" assume f: "f = Send  $\vee$  f = Receive"

```

```

from assms have "set ((map f (inv set M)) ·st ∅) = f ‘ (M ·set ∅)"
proof (induction rule: finite_induct)
  case empty thus ?case unfolding inv_def by auto
next
  case (insert m M)
  have "set (map f (inv set (insert m M)) ·st ∅) =
    insert (f m ·stp ∅) (set (map f (inv set M) ·st ∅))"
    by (simp add: insert.hyps(1) inv_set_fset subst_apply_strand_def)
  thus ?case using f insert.IH by auto
qed
}
thus "?A" "?B" by auto
qed

lemma strand_ground_subst_vars_subset:
  assumes "ground (subst_range ∅)" shows "varsst (S ·st ∅) ⊆ varsst S"
proof (induction S)
  case (Cons x S)
  have "varsstp (x ·stp ∅) ⊆ varsstp x" using ground_subst_fv_subset[OF assms]
  proof (cases x)
    case (Inequality X F)
    let ?∅ = "rm_vars (set X) ∅"
    have "fvpairs (F ·pairs ?∅) ⊆ fvpairs F"
    proof (induction F)
      case (Cons f F)
      obtain t t' where f: "f = (t, t')" by (metis surj_pair)
      hence "fvpairs (f#F ·pairs ?∅) = fv (t · ?∅) ∪ fv (t' · ?∅) ∪ fvpairs (F ·pairs ?∅)"
        "fvpairs (f#F) = fv t ∪ fv t' ∪ fvpairs F"
      by (auto simp add: subst_apply_pairs_def)
      thus ?case
        using ground_subst_fv_subset[OF ground_subset[OF rm_vars_img_subset assms, of "set X"]]
        Cons.IH
        by (metis (no_types, lifting) Un_mono)
    qed (simp add: subst_apply_pairs_def)
    moreover have
      "varsstp (x ·stp ∅) = fvpairs (F ·pairs rm_vars (set X) ∅) ∪ set X"
      "varsstp x = fvpairs F ∪ set X"
    using Inequality
    by (auto simp add: subst_apply_pairs_def)
    ultimately show ?thesis by auto
  qed auto
  thus ?case using Cons.IH by auto
qed (simp add: subst_apply_strand_def)

lemma ik_union_subset: "⋃ (P ‘ ikst S) ⊆ (⋃ x ∈ (set S). ⋃ (P ‘ trmsstp x))"
by (induct S rule: ikst.induct) auto

lemma ik_snd_empty[simp]: "ikst (map Send X) = {}"
by (induct "map Send X" arbitrary: X rule: ikst.induct) auto

lemma ik_snd_empty'[simp]: "ikst [Send t] = {}" by simp

lemma ik_append[iff]: "ikst (S@S') = ikst S ∪ ikst S'" by (induct S rule: ikst.induct) auto

lemma ik_cons: "ikst (x#S) = ikst [x] ∪ ikst S" using ik_append[of "[x]" S] by simp

lemma assignment_rhs_append[iff]: "assignment_rhsst (S@S') = assignment_rhsst S ∪ assignment_rhsst S'"
by (induct S rule: assignment_rhsst.induct) auto

lemma eqs_rcv_map_empty: "assignment_rhsst (map Receive M) = {}"
by auto

```

```

lemma ik_rcv_map: assumes "t ∈ set L" shows "t ∈ ikst (map Receive L)"
proof -
  { fix L L'
    have "t ∈ ikst [Receive t]" by auto
    hence "t ∈ ikst (map Receive L@Receive t#map Receive L')" using ik_append by auto
    hence "t ∈ ikst (map Receive (L@t#L'))" by auto
  }
  thus ?thesis using assms split_list_last by force
qed

lemma ik_subst: "ikst (S ·st δ) = ikst S ·set δ"
by (induct rule: ikst.induct) auto

lemma ik_rcv_map': assumes "t ∈ ikst (map Receive L)" shows "t ∈ set L"
using assms by force

lemma ik_append_subset[simp]: "ikst S ⊆ ikst (S@S'" "ikst S' ⊆ ikst (S@S')"
by (induct S rule: ikst.induct) auto

lemma assignment_rhs_append_subset[simp]:
  "assignment_rhsst S ⊆ assignment_rhsst (S@S'"
  "assignment_rhsst S' ⊆ assignment_rhsst (S@S'"
by (induct S rule: assignment_rhsst.induct) auto

lemma trmsst_cons: "trmsst (x#S) = trmsstp x ∪ trmsst S" by simp

lemma trm_strand_subst_cong:
  "t ∈ trmsst S ⟹ t · δ ∈ trmsst (S ·st δ)
  ∨ (∃X F. Inequality X F ∈ set S ∧ t · rm_vars (set X) δ ∈ trmsst (S ·st δ))"
  (is "t ∈ trmsst S ⟹ ?P t δ S")
  "t ∈ trmsst (S ·st δ) ⟹ (∃t'. t = t' · δ ∧ t' ∈ trmsst S)
  ∨ (∃X F. Inequality X F ∈ set S ∧ (∃t' ∈ trmspairs F. t = t' · rm_vars (set X) δ))"
  (is "t ∈ trmsst (S ·st δ) ⟹ ?Q t δ S")
proof -
  show "t ∈ trmsst S ⟹ ?P t δ S"
  proof (induction S)
    case (Cons x S) show ?case
    proof (cases "t ∈ trmsst S")
    case True
    hence "?P t δ S" using Cons by simp
    thus ?thesis
    by (cases x)
    (metis (no_types, lifting) Un_iff list.set_intros(2) strand_subst_hom(2) trmsst.cons)+
  next
    case False
    hence "t ∈ trmsstp x" using Cons.prem by auto
    thus ?thesis
    proof (induction x)
    case (Inequality X F)
    hence "t · rm_vars (set X) δ ∈ trmsstp (Inequality X F ·stp δ)"
    by (induct F) (auto simp add: subst_apply_pairs_def subst_apply_strand_step_def)
    thus ?case by fastforce
    qed (auto simp add: subst_apply_strand_step_def)
  qed
  qed simp

show "t ∈ trmsst (S ·st δ) ⟹ ?Q t δ S"
proof (induction S)
  case (Cons x S) show ?case
  proof (cases "t ∈ trmsst (S ·st δ)")
  case True
  hence "?Q t δ S" using Cons by simp
  thus ?thesis by (cases x) force+

```

```

next
  case False
  hence "t ∈ trmsstp (x ·stp δ)" using Cons.premis by auto
  thus ?thesis
  proof (induction x)
    case (Inequality X F)
    hence "t ∈ trmsstp (Inequality X F) ·set rm_vars (set X) δ"
      by (induct F) (force simp add: subst_apply_pairs_def)+
    thus ?case by fastforce
  qed (auto simp add: subst_apply_strand_step_def)
qed
qed simp
qed

```

### 3.1.2 Lemmata: Free Variables of Strands

lemma *fv\_trm\_snd\_rcv[simp]*: "fv<sub>set</sub> (trms<sub>stp</sub> (Send t)) = fv t" "fv<sub>set</sub> (trms<sub>stp</sub> (Receive t)) = fv t"  
by *simp\_all*

lemma *in\_strand\_fv\_subset*: "x ∈ set S ⇒ vars<sub>stp</sub> x ⊆ vars<sub>st</sub> S" by *fastforce*

lemma *in\_strand\_fv\_subset\_snd*: "Send t ∈ set S ⇒ fv t ⊆ ⋃ (set (map fv<sub>snd</sub> S))" by *auto*

lemma *in\_strand\_fv\_subset\_rcv*: "Receive t ∈ set S ⇒ fv t ⊆ ⋃ (set (map fv<sub>rcv</sub> S))" by *auto*

lemma *fv<sub>snd</sub>E*:

assumes "v ∈ ⋃ (set (map fv<sub>snd</sub> S))"  
obtains t where "send⟨t⟩<sub>st</sub> ∈ set S" "v ∈ fv t"

proof -

have "∃ t. send⟨t⟩<sub>st</sub> ∈ set S ∧ v ∈ fv t"  
by (metis (no\_types, lifting) assms UN\_E empty\_iff set\_map strand\_step.case\_eq\_if  
fv<sub>snd</sub>\_def strand\_step.collapse(1))  
thus ?thesis by (metis that)

qed

lemma *fv<sub>rcv</sub>E*:

assumes "v ∈ ⋃ (set (map fv<sub>rcv</sub> S))"  
obtains t where "receive⟨t⟩<sub>st</sub> ∈ set S" "v ∈ fv t"

proof -

have "∃ t. receive⟨t⟩<sub>st</sub> ∈ set S ∧ v ∈ fv t"  
by (metis (no\_types, lifting) assms UN\_E empty\_iff set\_map strand\_step.case\_eq\_if  
fv<sub>rcv</sub>\_def strand\_step.collapse(2))  
thus ?thesis by (metis that)

qed

lemma *vars<sub>stp</sub>I[intro]*: "x ∈ fv<sub>stp</sub> s ⇒ x ∈ vars<sub>stp</sub> s"

by (induct s rule: fv<sub>stp</sub>.induct) auto

lemma *vars<sub>st</sub>I[intro]*: "x ∈ fv<sub>st</sub> S ⇒ x ∈ vars<sub>st</sub> S" using *vars<sub>stp</sub>I* by *fastforce*

lemma *fv<sub>st</sub>\_subset\_vars<sub>st</sub>[simp]*: "fv<sub>st</sub> S ⊆ vars<sub>st</sub> S" using *vars<sub>st</sub>I* by *force*

lemma *vars<sub>st</sub>\_is\_fv<sub>st</sub>\_bvars<sub>st</sub>*: "vars<sub>st</sub> S = fv<sub>st</sub> S ∪ bvars<sub>st</sub> S"

proof (induction S)

case (Cons x S) thus ?case  
proof (induction x)  
case (Inequality X F) thus ?case by (induct F) auto  
qed auto

qed *simp*

lemma *fv<sub>stp</sub>\_is\_subterm\_trms<sub>stp</sub>*: "x ∈ fv<sub>stp</sub> a ⇒ Var x ∈ subterms<sub>set</sub> (trms<sub>stp</sub> a)"  
using *var\_is\_subterm* by (cases a) *force+*

lemma *fv<sub>st</sub>\_is\_subterm\_trms<sub>st</sub>*: "x ∈ fv<sub>st</sub> A ⇒ Var x ∈ subterms<sub>set</sub> (trms<sub>st</sub> A)"

proof (induction A)

### 3 The Typing Result for Non-Stateful Protocols

```

case (Cons a A) thus ?case using fv_stp_is_subterm_trms_stp by (cases "x ∈ fv_st A") auto
qed simp

lemma vars_st_snd_map: "vars_st (map Send X) = fv (Fun f X)" by auto

lemma vars_st_rcv_map: "vars_st (map Receive X) = fv (Fun f X)" by auto

lemma vars_snd_rcv_union:
  "vars_stp x = fv_snd x ∪ fv_rcv x ∪ fv_eq assign x ∪ fv_eq check x ∪ fv_ineq x ∪ set (bvars_stp x)"
proof (cases x)
  case (Equality ac t t') thus ?thesis by (cases ac) auto
qed auto

lemma fv_snd_rcv_union:
  "fv_stp x = fv_snd x ∪ fv_rcv x ∪ fv_eq assign x ∪ fv_eq check x ∪ fv_ineq x"
proof (cases x)
  case (Equality ac t t') thus ?thesis by (cases ac) auto
qed auto

lemma fv_snd_rcv_empty[simp]: "fv_snd x = {} ∨ fv_rcv x = {}" by (cases x) simp_all

lemma vars_snd_rcv_strand[iff]:
  "vars_st (S::('a,'b) strand) =
    (⋃(set (map fv_snd S))) ∪ (⋃(set (map fv_rcv S))) ∪ (⋃(set (map (fv_eq assign) S)))
    ∪ (⋃(set (map (fv_eq check) S))) ∪ (⋃(set (map fv_ineq S))) ∪ bvars_st S"
unfolding bvars_st_def
proof (induction S)
  case (Cons x S)
  have "⋂s V. vars_stp (s::('a,'b) strand_step) ∪ V =
    fv_snd s ∪ fv_rcv s ∪ fv_eq assign s ∪ fv_eq check s ∪ fv_ineq s ∪ set (bvars_stp s) ∪ V"
  by (metis vars_snd_rcv_union)
  thus ?case using Cons.IH by (auto simp add: sup_assoc sup_left_commute)
qed simp

lemma fv_snd_rcv_strand[iff]:
  "fv_st (S::('a,'b) strand) =
    (⋃(set (map fv_snd S))) ∪ (⋃(set (map fv_rcv S))) ∪ (⋃(set (map (fv_eq assign) S)))
    ∪ (⋃(set (map (fv_eq check) S))) ∪ (⋃(set (map fv_ineq S)))"
unfolding bvars_st_def
proof (induction S)
  case (Cons x S)
  have "⋂s V. fv_stp (s::('a,'b) strand_step) ∪ V =
    fv_snd s ∪ fv_rcv s ∪ fv_eq assign s ∪ fv_eq check s ∪ fv_ineq s ∪ V"
  by (metis fv_snd_rcv_union)
  thus ?case using Cons.IH by (auto simp add: sup_assoc sup_left_commute)
qed simp

lemma vars_snd_rcv_strand2[iff]:
  "wfrestrictedvars_st (S::('a,'b) strand) =
    (⋃(set (map fv_snd S))) ∪ (⋃(set (map fv_rcv S))) ∪ (⋃(set (map (fv_eq assign) S)))"
by (induct S) (auto simp add: split: strand_step.split poscheckvariant.split)

lemma fv_snd_rcv_strand_subset[simp]:
  "⋃(set (map fv_snd S)) ⊆ fv_st S" "⋃(set (map fv_rcv S)) ⊆ fv_st S"
  "⋃(set (map (fv_eq ac) S)) ⊆ fv_st S" "⋃(set (map fv_ineq S)) ⊆ fv_st S"
  "wfvarsoccs_st S ⊆ fv_st S"
proof -
  show "⋃(set (map fv_snd S)) ⊆ fv_st S" "⋃(set (map fv_rcv S)) ⊆ fv_st S" "⋃(set (map fv_ineq S)) ⊆
fv_st S"
  using fv_snd_rcv_strand[of S] by auto

  show "⋃(set (map (fv_eq ac) S)) ⊆ fv_st S"
  by (induct S) (auto split: strand_step.split poscheckvariant.split)

```

```

show "wfvarsoccsst S ⊆ fvst S"
  by (induct S) (auto split: strand_step.split poscheckvariant.split)
qed

lemma vars_snd_rcv_strand_subset2[simp]:
  "⋃(set (map fvsnd S)) ⊆ wfrestrictedvarsst S" "⋃(set (map fvrcv S)) ⊆ wfrestrictedvarsst S"
  "⋃(set (map (fveq assign) S)) ⊆ wfrestrictedvarsst S" "wfvarsoccsst S ⊆ wfrestrictedvarsst S"
  by (induction S) (auto split: strand_step.split poscheckvariant.split)

lemma wfrestrictedvarsst_subset_varsst: "wfrestrictedvarsst S ⊆ varsst S"
  by (induction S) (auto split: strand_step.split poscheckvariant.split)

lemma subst_sends_strand_step_fv_to_img: "fvstp (x ·stp δ) ⊆ fvstp x ∪ range_vars δ"
  using subst_sends_fv_to_img[of _ δ]
  proof (cases x)
    case (Inequality X F)
    let ?ϑ = "rm_vars (set X) δ"
    have "fvpairs (F ·pairs ?ϑ) ⊆ fvpairs F ∪ range_vars ?ϑ"
    proof (induction F)
      case (Cons f F) thus ?case
        using subst_sends_fv_to_img[of _ ?ϑ]
        by (auto simp add: subst_apply_pairs_def)
    qed (auto simp add: subst_apply_pairs_def)
    hence "fvpairs (F ·pairs ?ϑ) ⊆ fvpairs F ∪ range_vars δ"
    using rm_vars_img_subset[of "set X" δ] fv_set_mono
    unfolding range_vars_alt_def by blast+
    thus ?thesis using Inequality by (auto simp add: subst_apply_strand_step_def)
  qed (auto simp add: subst_apply_strand_step_def)

lemma subst_sends_strand_fv_to_img: "fvst (S ·st δ) ⊆ fvst S ∪ range_vars δ"
  proof (induction S)
    case (Cons x S)
    have *: "fvst (x#S ·st δ) = fvstp (x ·stp δ) ∪ fvst (S ·st δ)"
      "fvst (x#S) ∪ range_vars δ = fvstp x ∪ fvst S ∪ range_vars δ"
    by auto
    thus ?case using Cons.IH subst_sends_strand_step_fv_to_img[of x δ] by auto
  qed simp

lemma ineq_apply_subst:
  assumes "subst_domain δ ∩ set X = {}"
  shows "(Inequality X F) ·stp δ = Inequality X (F ·pairs δ)"
  using rm_vars_apply'[OF assms] by (simp add: subst_apply_strand_step_def)

lemma fv_strand_step_subst:
  assumes "P = fvstp ∨ P = fvrcv ∨ P = fvsnd ∨ P = fveq ac ∨ P = fvineq"
  and "set (bvarsstp x) ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "fvset (δ ' (P x)) = P (x ·stp δ)"
  proof (cases x)
    case (Send t)
    hence "varsstp x = fv t" "fvsnd x = fv t" by auto
    thus ?thesis using assms Send subst_apply_fv_unfold[of _ δ] by auto
  next
    case (Receive t)
    hence "varsstp x = fv t" "fvrcv x = fv t" by auto
    thus ?thesis using assms Receive subst_apply_fv_unfold[of _ δ] by auto
  next
    case (Equality ac' t t') show ?thesis
    proof (cases "ac = ac'")
      case True
      hence "varsstp x = fv t ∪ fv t'" "fveq ac x = fv t ∪ fv t'"
      using Equality
      by auto
    case False
    thus ?thesis by auto
  qed

```

```

    thus ?thesis
      using assms Equality subst_apply_fv_unfold[of _  $\delta$ ] True
      by auto
  next
    case False
    hence "varsstp x = fv t  $\cup$  fv t'" "fveq ac x = {}"
      using Equality
      by auto
    thus ?thesis
      using assms Equality subst_apply_fv_unfold[of _  $\delta$ ] False
      by auto
  qed
next
  case (Inequality X F)
  hence 1: "set X  $\cap$  (subst_domain  $\delta \cup$  range_vars  $\delta$ ) = {}"
    "x .stp  $\delta$  = Inequality X (F .pairs  $\delta$ )"
    "rm_vars (set X)  $\delta$  =  $\delta$ "
    using assms ineq_apply_subst[of  $\delta$  X F] rm_vars_apply' [of  $\delta$  "set X"]
    unfolding range_vars_alt_def by force+

  have 2: "fvineq x = fvpairs F - set X" using Inequality by auto
  hence "fvset ( $\delta$  ' fvineq x) = fvset ( $\delta$  ' fvpairs F) - set X"
    using fvset-subst_img_eq[OF 1(1), of "fvpairs F"] by simp
  hence 3: "fvset ( $\delta$  ' fvineq x) = fvpairs (F .pairs  $\delta$ ) - set X" by (metis fvpairs-step_subst)

  have 4: "fvineq (x .stp  $\delta$ ) = fvpairs (F .pairs  $\delta$ ) - set X" using 1(2) by auto

  show ?thesis
    using assms(1) Inequality subst_apply_fv_unfold[of _  $\delta$ ] 1(2) 2 3 4
    unfolding fveq-def fvrcv-def fvsnd-def
    by (metis (no_types) Sup_empty image_empty fvpairs.simps fvset.simps
        fvstp.simps(4) strand_step.simps(20))
  qed

lemma fv_strand_subst:
  assumes "P = fvstp  $\vee$  P = fvrcv  $\vee$  P = fvsnd  $\vee$  P = fveq ac  $\vee$  P = fvineq"
  and "bvarsst S  $\cap$  (subst_domain  $\delta \cup$  range_vars  $\delta$ ) = {}"
  shows "fvset ( $\delta$  ' ( $\bigcup$  (set (map P S)))) =  $\bigcup$  (set (map P (S .st  $\delta$ )))"
using assms(2)
proof (induction S)
  case (Cons x S)
  hence *: "bvarsst S  $\cap$  (subst_domain  $\delta \cup$  range_vars  $\delta$ ) = {}"
    "set (bvarsstp x)  $\cap$  (subst_domain  $\delta \cup$  range_vars  $\delta$ ) = {}"
    unfolding bvarsst-def by force+
  hence **: "fvset ( $\delta$  ' P x) = P (x .stp  $\delta$ )" using fv_strand_step_subst[OF assms(1), of x  $\delta$ ] by auto
  have "fvset ( $\delta$  ' ( $\bigcup$  (set (map P (x#S)))) = fvset ( $\delta$  ' P x)  $\cup$  ( $\bigcup$  (set (map P ((S .st  $\delta$ )))))"
    using Cons unfolding range_vars_alt_def bvarsst-def by force
  hence "fvset ( $\delta$  ' ( $\bigcup$  (set (map P (x#S)))) = P (x .stp  $\delta$ )  $\cup$  fvset ( $\delta$  ' ( $\bigcup$  (set (map P S))))"
    using ** by simp
  thus ?case using Cons.IH[OF *(1)] unfolding bvarsst-def by simp
qed simp

lemma fv_strand_subst2:
  assumes "bvarsst S  $\cap$  (subst_domain  $\delta \cup$  range_vars  $\delta$ ) = {}"
  shows "fvset ( $\delta$  ' (wfrestrictedvarsst S)) = wfrestrictedvarsst (S .st  $\delta$ )"
by (metis (no_types, lifting) assms fvset.simps vars_snd_rcv_strand2 fv_strand_subst UN_Un image_Un)

lemma fv_strand_subst':
  assumes "bvarsst S  $\cap$  (subst_domain  $\delta \cup$  range_vars  $\delta$ ) = {}"
  shows "fvset ( $\delta$  ' (fvst S)) = fvst (S .st  $\delta$ )"
by (metis assms fv_strand_subst fvst-def)

lemma fv_trmspairs-is_fvpairs:

```



```

"fv_set (trms_pairs F) = fv_pairs F"
by auto

lemma fv_pairs_in_fv_trms_pairs: "x ∈ fv_pairs F ⇒ x ∈ fv_set (trms_pairs F)"
using fv_trms_pairs_is_fv_pairs[of F] by blast

lemma trms_st_append: "trms_st (A@B) = trms_st A ∪ trms_st B"
by auto

lemma trms_pairs_subst: "trms_pairs (a ·_pairs ∅) = trms_pairs a ·_set ∅"
by (auto simp add: subst_apply_pairs_def)

lemma trms_pairs_fv_subst_subset:
  "t ∈ trms_pairs F ⇒ fv (t · ∅) ⊆ fv_pairs (F ·_pairs ∅)"
by (force simp add: subst_apply_pairs_def)

lemma trms_pairs_fv_subst_subset':
  fixes t::('a,'b) term and ∅::('a,'b) subst
  assumes "t ∈ subterms_set (trms_pairs F)"
  shows "fv (t · ∅) ⊆ fv_pairs (F ·_pairs ∅)"
proof -
  { fix x assume "x ∈ fv t"
    hence "x ∈ fv_pairs F"
      using fv_subset[OF assms] fv_subterms_set[of "trms_pairs F"] fv_trms_pairs_is_fv_pairs[of F]
      by blast
    hence "fv (∅ x) ⊆ fv_pairs (F ·_pairs ∅)" using fv_pairs_subst_fv_subset by fast
  } thus ?thesis by (meson fv_subst_obtain_var subset_iff)
qed

lemma trms_pairs_funs_term_cases:
  assumes "t ∈ trms_pairs (F ·_pairs ∅)" "f ∈ funs_term t"
  shows "(∃ u ∈ trms_pairs F. f ∈ funs_term u) ∨ (∃ x ∈ fv_pairs F. f ∈ funs_term (∅ x))"
using assms(1)
proof (induction F)
  case (Cons g F)
  obtain s u where g: "g = (s,u)" by (metis surj_pair)
  show ?case
  proof (cases "t ∈ trms_pairs (F ·_pairs ∅)")
    case False
    thus ?thesis
      using assms(2) Cons.prem1 g funs_term_subst[of _ ∅]
      by (auto simp add: subst_apply_pairs_def)
  qed (use Cons.IH in fastforce)
qed simp

lemma trm_stp_subst:
  assumes "subst_domain ∅ ∩ set (bvars_stp a) = {}"
  shows "trms_stp (a ·_stp ∅) = trms_stp a ·_set ∅"
proof -
  have "rm_vars (set (bvars_stp a)) ∅ = ∅" using assms by force
  thus ?thesis
    using assms
    by (auto simp add: subst_apply_pairs_def subst_apply_strand_step_def
        split: strand_step.splits)
qed

lemma trms_st_subst:
  assumes "subst_domain ∅ ∩ bvars_st A = {}"
  shows "trms_st (A ·_st ∅) = trms_st A ·_set ∅"
using assms
proof (induction A)
  case (Cons a A)
  have 1: "subst_domain ∅ ∩ bvars_st A = {}" "subst_domain ∅ ∩ set (bvars_stp a) = {}"

```

```

using Cons.premis by auto
hence IH: "trmsst A ·set ∅ = trmsst (A ·st ∅)" using Cons.IH by simp

have "trmsst (a#A) = trmsstp a ∪ trmsst A" by auto
hence 2: "trmsst (a#A) ·set ∅ = (trmsstp a ·set ∅) ∪ (trmsst A ·set ∅)" by (metis image_Un)

have "trmsst (a#A ·st ∅) = (trmsstp (a ·stp ∅)) ∪ trmsst (A ·st ∅)"
  by (auto simp add: subst_apply_strand_def)
hence 3: "trmsst (a#A ·st ∅) = (trmsstp a ·set ∅) ∪ trmsst (A ·st ∅)"
  using trmstp-subst[OF 1(2)] by auto

show ?case using IH 2 3 by metis
qed (simp add: subst_apply_strand_def)

lemma strand_map_set_subst:
  assumes δ: "bvarsst S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "⋃(set (map trmsstp (S ·st δ))) = (⋃(set (map trmsstp S))) ·set δ"
using assms
proof (induction S)
  case (Cons x S)
  hence "bvarsst [x] ∩ subst_domain δ = {}" "bvarsst S ∩ (subst_domain δ ∪ range_vars δ) = {}"
    unfolding bvarsst-def by force+
  hence *: "subst_domain δ ∩ set (bvarsstp x) = {}"
    "⋃(set (map trmsstp (S ·st δ))) = ⋃(set (map trmsstp S)) ·set δ"
    using Cons.IH(1) bvarsst-singleton[of x] by auto
  hence "trmsstp (x ·stp δ) = (trmsstp x) ·set δ"
  proof (cases x)
    case (Inequality X F)
    thus ?thesis
      using rm_vars_apply'[of δ "set X"] *
      by (metis (no_types, lifting) image_cong trmstp-subst)
  qed simp_all
  thus ?case using * subst_all_insert by auto
qed simp

lemma subst_apply_fv_subset_strand_trm:
  assumes P: "P = fvstp ∨ P = fvrcv ∨ P = fvsnd ∨ P = fveq ac ∨ P = fvineq"
  and fv_sub: "fv t ⊆ ⋃(set (map P S)) ∪ V"
  and δ: "bvarsst S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "fv (t · δ) ⊆ ⋃(set (map P (S ·st δ))) ∪ fvset (δ ' V)"
using fv_strand_subst[OF P δ] subst_apply_fv_subset[OF fv_sub, of δ] by force

lemma subst_apply_fv_subset_strand_trm2:
  assumes fv_sub: "fv t ⊆ wfrestrictedvarsst S ∪ V"
  and δ: "bvarsst S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "fv (t · δ) ⊆ wfrestrictedvarsst (S ·st δ) ∪ fvset (δ ' V)"
using fv_strand_subst2[OF δ] subst_apply_fv_subset[OF fv_sub, of δ] by force

lemma subst_apply_fv_subset_strand:
  assumes P: "P = fvstp ∨ P = fvrcv ∨ P = fvsnd ∨ P = fveq ac ∨ P = fvineq"
  and P_subset: "P x ⊆ ⋃(set (map P S)) ∪ V"
  and δ: "bvarsst S ∩ (subst_domain δ ∪ range_vars δ) = {}"
    "set (bvarsstp x) ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "P (x ·stp δ) ⊆ ⋃(set (map P (S ·st δ))) ∪ fvset (δ ' V)"
proof (cases x)
  case (Send t)
  hence *: "fvstp x = fv t" "fvstp (x ·stp δ) = fv (t · δ)"
    "fvrcv x = {}" "fvrcv (x ·stp δ) = {}"
    "fvsnd x = fv t" "fvsnd (x ·stp δ) = fv (t · δ)"
    "fveq ac x = {}" "fveq ac (x ·stp δ) = {}"
    "fvineq x = {}" "fvineq (x ·stp δ) = {}"
    by auto
  hence **: "(P x = fv t ∧ P (x ·stp δ) = fv (t · δ)) ∨ (P x = {} ∧ P (x ·stp δ) = {})" by (metis P)

```

```

moreover
{ assume "P x = {}" "P (x ·stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fv t" "P (x ·stp δ) = fv (t · δ)"
  hence "fv t ⊆ ⋃ (set (map P S)) ∪ V" using P_subset by auto
  hence "fv (t · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' V)"
    unfolding varsst_def using P subst_apply_fv_subset_strand_trm assms by blast
  hence ?thesis using ⟨P (x ·stp δ) = fv (t · δ)⟩ by force
}
ultimately show ?thesis by metis
next
case (Receive t)
hence *: "fvstp x = fv t" "fvstp (x ·stp δ) = fv (t · δ)"
      "fvrcv x = fv t" "fvrcv (x ·stp δ) = fv (t · δ)"
      "fvsnd x = {}" "fvsnd (x ·stp δ) = {}"
      "fveq ac x = {}" "fveq ac (x ·stp δ) = {}"
      "fvineq x = {}" "fvineq (x ·stp δ) = {}"
  by auto
hence **: "(P x = fv t ∧ P (x ·stp δ) = fv (t · δ)) ∨ (P x = {} ∧ P (x ·stp δ) = {})" by (metis P)
moreover
{ assume "P x = {}" "P (x ·stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fv t" "P (x ·stp δ) = fv (t · δ)"
  hence "fv t ⊆ ⋃ (set (map P S)) ∪ V" using P_subset by auto
  hence "fv (t · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' V)"
    unfolding varsst_def using P subst_apply_fv_subset_strand_trm assms by blast
  hence ?thesis using ⟨P (x ·stp δ) = fv (t · δ)⟩ by blast
}
ultimately show ?thesis by metis
next
case (Equality ac' t t') show ?thesis
proof (cases "ac' = ac")
  case True
  hence *: "fvstp x = fv t ∪ fv t'" "fvstp (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ)"
        "fvrcv x = {}" "fvrcv (x ·stp δ) = {}"
        "fvsnd x = {}" "fvsnd (x ·stp δ) = {}"
        "fveq ac x = fv t ∪ fv t'" "fveq ac (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ)"
        "fvineq x = {}" "fvineq (x ·stp δ) = {}"
    using Equality by auto
  hence **: "(P x = fv t ∪ fv t' ∧ P (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ))
    ∨ (P x = {} ∧ P (x ·stp δ) = {})"
    by (metis P)
  moreover
  { assume "P x = {}" "P (x ·stp δ) = {}" hence ?thesis by simp }
  moreover
  { assume "P x = fv t ∪ fv t'" "P (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ)"
    hence "fv t ⊆ ⋃ (set (map P S)) ∪ V" "fv t' ⊆ ⋃ (set (map P S)) ∪ V" using P_subset by auto
    hence "fv (t · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' V)"
      "fv (t' · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' V)"
      unfolding varsst_def using P subst_apply_fv_subset_strand_trm assms by metis+
    hence ?thesis using ⟨P (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ)⟩ by blast
  }
  ultimately show ?thesis by metis
next
case False
hence *: "fvstp x = fv t ∪ fv t'" "fvstp (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ)"
      "fvrcv x = {}" "fvrcv (x ·stp δ) = {}"
      "fvsnd x = {}" "fvsnd (x ·stp δ) = {}"
      "fveq ac x = {}" "fveq ac (x ·stp δ) = {}"
      "fvineq x = {}" "fvineq (x ·stp δ) = {}"
  using Equality by auto
hence **: "(P x = fv t ∪ fv t' ∧ P (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ))
  ∨ (P x = {} ∧ P (x ·stp δ) = {})"

```

```

    by (metis P)
  moreover
  { assume "P x = {}" "P (x ·stp δ) = {}" hence ?thesis by simp }
  moreover
  { assume "P x = fv t ∪ fv t'" "P (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ)"
    hence "fv t ⊆ ⋃ (set (map P S)) ∪ V" "fv t' ⊆ ⋃ (set (map P S)) ∪ V" using P_subset by auto
    hence "fv (t · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' V)"
      "fv (t' · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' V)"
    unfolding varsst_def using P subst_apply_fv_subset_strand_trm assms by metis+
    hence ?thesis using ⟨P (x ·stp δ) = fv (t · δ) ∪ fv (t' · δ)⟩ by blast
  }
  ultimately show ?thesis by metis
qed
next
case (Inequality X F)
hence *: "fvstp x = fvpairs F - set X" "fvstp (x ·stp δ) = fvpairs (F ·pairs δ) - set X"
  "fvrcv x = {}" "fvrcv (x ·stp δ) = {}"
  "fvsnd x = {}" "fvsnd (x ·stp δ) = {}"
  "fveq ac x = {}" "fveq ac (x ·stp δ) = {}"
  "fvineq x = fvpairs F - set X"
  "fvineq (x ·stp δ) = fvpairs (F ·pairs δ) - set X"
  using δ(2) ineq_apply_subst[of δ X F] by force+
hence **: "(P x = fvpairs F - set X ∧ P (x ·stp δ) = fvpairs (F ·pairs δ) - set X)
  ∨ (P x = {} ∧ P (x ·stp δ) = {})"
  by (metis P)
moreover
{ assume "P x = {}" "P (x ·stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fvpairs F - set X" "P (x ·stp δ) = fvpairs (F ·pairs δ) - set X"
  hence "fvpairs F - set X ⊆ ⋃ (set (map P S)) ∪ V"
    using P_subset by auto
  hence "fvpairs (F ·pairs δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' (V ∪ set X))"
  proof (induction F)
    case (Cons f G)
    hence IH: "fvpairs (G ·pairs δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' (V ∪ set X))"
      by (metis (no_types, lifting) Diff_subset_conv UN_insert le_sup_iff
        list.simps(15) fvpairs.simps)
    obtain t t' where f: "f = (t, t')" by (metis surj_pair)
    hence "fv t ⊆ ⋃ (set (map P S)) ∪ (V ∪ set X)" "fv t' ⊆ ⋃ (set (map P S)) ∪ (V ∪ set X)"
      using Cons.prem by auto
    hence "fv (t · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' (V ∪ set X))"
      "fv (t' · δ) ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' (V ∪ set X))"
      using subst_apply_fv_subset_strand_trm[OF P _ assms(3)]
      by blast+
    thus ?case using f IH by (auto simp add: subst_apply_pairs_def)
  qed (simp add: subst_apply_pairs_def)
  moreover have "fvset (δ ' set X) = set X" using assms(4) Inequality by force
  ultimately have "fvpairs (F ·pairs δ) - set X ⊆ ⋃ (set (map P (S ·st δ))) ∪ fvset (δ ' V)"
    by auto
  hence ?thesis using ⟨P (x ·stp δ) = fvpairs (F ·pairs δ) - set X⟩ by blast
}
ultimately show ?thesis by metis
qed

lemma subst_apply_fv_subset_strand2:
  assumes P: "P = fvstp ∨ P = fvrcv ∨ P = fvsnd ∨ P = fveq ac ∨ P = fvineq ∨ P = fvreq ac"
  and P_subset: "P x ⊆ wfrestrictedvarsst S ∪ V"
  and δ: "bvarsst S ∩ (subst_domain δ ∪ range_vars δ) = {}"
    "set (bvarsstp x) ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "P (x ·stp δ) ⊆ wfrestrictedvarsst (S ·st δ) ∪ fvset (δ ' V)"
proof (cases x)
  case (Send t)
  hence *: "fvstp x = fv t" "fvstp (x ·stp δ) = fv (t · δ)"

```

```

"fvrcv x = {}" "fvrcv (x .stp δ) = {}"
"fvsnd x = fv t" "fvsnd (x .stp δ) = fv (t . δ)"
"fveq ac x = {}" "fveq ac (x .stp δ) = {}"
"fvineq x = {}" "fvineq (x .stp δ) = {}"
"fvreq ac x = {}" "fvreq ac (x .stp δ) = {}"

by auto
hence **: "(P x = fv t ∧ P (x .stp δ) = fv (t . δ)) ∨ (P x = {} ∧ P (x .stp δ) = {})" by (metis P)
moreover
{ assume "P x = {}" "P (x .stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fv t" "P (x .stp δ) = fv (t . δ)"
  hence "fv t ⊆ wfrestrictedvarsst S ∪ V" using P_subset by auto
  hence "fv (t . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
    using P subst_apply_fv_subset_strand_trm2 assms by blast
  hence ?thesis using ⟨P (x .stp δ) = fv (t . δ)⟩ by blast
}
ultimately show ?thesis by metis
next
case (Receive t)
hence *: "fvstp x = fv t" "fvstp (x .stp δ) = fv (t . δ)"
"fvrcv x = fv t" "fvrcv (x .stp δ) = fv (t . δ)"
"fvsnd x = {}" "fvsnd (x .stp δ) = {}"
"fveq ac x = {}" "fveq ac (x .stp δ) = {}"
"fvineq x = {}" "fvineq (x .stp δ) = {}"
"fvreq ac x = {}" "fvreq ac (x .stp δ) = {}"

by auto
hence **: "(P x = fv t ∧ P (x .stp δ) = fv (t . δ)) ∨ (P x = {} ∧ P (x .stp δ) = {})" by (metis P)
moreover
{ assume "P x = {}" "P (x .stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fv t" "P (x .stp δ) = fv (t . δ)"
  hence "fv t ⊆ wfrestrictedvarsst S ∪ V" using P_subset by auto
  hence "fv (t . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
    using P subst_apply_fv_subset_strand_trm2 assms by blast
  hence ?thesis using ⟨P (x .stp δ) = fv (t . δ)⟩ by blast
}
ultimately show ?thesis by metis
next
case (Equality ac' t t') show ?thesis
proof (cases "ac' = ac")
case True
hence *: "fvstp x = fv t ∪ fv t'" "fvstp (x .stp δ) = fv (t . δ) ∪ fv (t' . δ)"
"fvrcv x = {}" "fvrcv (x .stp δ) = {}"
"fvsnd x = {}" "fvsnd (x .stp δ) = {}"
"fveq ac x = fv t ∪ fv t'" "fveq ac (x .stp δ) = fv (t . δ) ∪ fv (t' . δ)"
"fvineq x = {}" "fvineq (x .stp δ) = {}"
"fvreq ac x = fv t'" "fvreq ac (x .stp δ) = fv (t' . δ)"

using Equality by auto
hence **: "(P x = fv t ∪ fv t' ∧ P (x .stp δ) = fv (t . δ) ∪ fv (t' . δ))
  ∨ (P x = {} ∧ P (x .stp δ) = {})
  ∨ (P x = fv t' ∧ P (x .stp δ) = fv (t' . δ))"

by (metis P)
moreover
{ assume "P x = {}" "P (x .stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fv t ∪ fv t'" "P (x .stp δ) = fv (t . δ) ∪ fv (t' . δ)"
  hence "fv t ⊆ wfrestrictedvarsst S ∪ V" "fv t' ⊆ wfrestrictedvarsst S ∪ V" using P_subset by
auto
hence "fv (t . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
"fv (t' . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
  using P subst_apply_fv_subset_strand_trm2 assms by blast+
hence ?thesis using ⟨P (x .stp δ) = fv (t . δ) ∪ fv (t' . δ)⟩ by blast
}

```

```

moreover
{ assume "P x = fv t'" "P (x .stp δ) = fv (t' . δ)"
  hence "fv t' ⊆ wfrestrictedvarsst S ∪ V" using P_subset by auto
  hence "fv (t' . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
    using P subst_apply_fv_subset_strand_trm2 assms by blast+
  hence ?thesis using ⟨P (x .stp δ) = fv (t' . δ)⟩ by blast
}
ultimately show ?thesis by metis
next
case False
hence *: "fvstp x = fv t ∪ fv t'" "fvstp (x .stp δ) = fv (t . δ) ∪ fv (t' . δ)"
  "fvrcv x = {}" "fvrcv (x .stp δ) = {}"
  "fvsnd x = {}" "fvsnd (x .stp δ) = {}"
  "fveq ac x = {}" "fveq ac (x .stp δ) = {}"
  "fvineq x = {}" "fvineq (x .stp δ) = {}"
  "fvreq ac x = {}" "fvreq ac (x .stp δ) = {}"
  using Equality by auto
hence **: "(P x = fv t ∪ fv t' ∧ P (x .stp δ) = fv (t . δ) ∪ fv (t' . δ))
  ∨ (P x = {} ∧ P (x .stp δ) = {})"
  ∨ (P x = fv t' ∧ P (x .stp δ) = fv (t' . δ))"
  by (metis P)
moreover
{ assume "P x = {}" "P (x .stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fv t ∪ fv t'" "P (x .stp δ) = fv (t . δ) ∪ fv (t' . δ)"
  hence "fv t ⊆ wfrestrictedvarsst S ∪ V" "fv t' ⊆ wfrestrictedvarsst S ∪ V"
    using P_subset by auto
  hence "fv (t . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
    "fv (t' . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
    using P subst_apply_fv_subset_strand_trm2 assms by blast+
  hence ?thesis using ⟨P (x .stp δ) = fv (t . δ) ∪ fv (t' . δ)⟩ by blast
}
moreover
{ assume "P x = fv t'" "P (x .stp δ) = fv (t' . δ)"
  hence "fv t' ⊆ wfrestrictedvarsst S ∪ V" using P_subset by auto
  hence "fv (t' . δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' V)"
    using P subst_apply_fv_subset_strand_trm2 assms by blast+
  hence ?thesis using ⟨P (x .stp δ) = fv (t' . δ)⟩ by blast
}
ultimately show ?thesis by metis
qed
next
case (Inequality X F)
hence *: "fvstp x = fvpairs F - set X" "fvstp (x .stp δ) = fvpairs (F .pairs δ) - set X"
  "fvrcv x = {}" "fvrcv (x .stp δ) = {}"
  "fvsnd x = {}" "fvsnd (x .stp δ) = {}"
  "fveq ac x = {}" "fveq ac (x .stp δ) = {}"
  "fvineq x = fvpairs F - set X" "fvineq (x .stp δ) = fvpairs (F .pairs δ) - set X"
  "fvreq ac x = {}" "fvreq ac (x .stp δ) = {}"
  using δ(2) ineq_apply_subst[of δ X F] by force+
hence **: "(P x = fvpairs F - set X ∧ P (x .stp δ) = fvpairs (F .pairs δ) - set X)
  ∨ (P x = {} ∧ P (x .stp δ) = {})"
  by (metis P)
moreover
{ assume "P x = {}" "P (x .stp δ) = {}" hence ?thesis by simp }
moreover
{ assume "P x = fvpairs F - set X" "P (x .stp δ) = fvpairs (F .pairs δ) - set X"
  hence "fvpairs F - set X ⊆ wfrestrictedvarsst S ∪ V" using P_subset by auto
  hence "fvpairs (F .pairs δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' (V ∪ set X))"
  proof (induction F)
    case (Cons f G)
    hence IH: "fvpairs (G .pairs δ) ⊆ wfrestrictedvarsst (S .st δ) ∪ fvset (δ ' (V ∪ set X))"
      by (metis (no_types, lifting) Diff_subset_conv UN_insert le_sup_iff)

```

```

      list.simps(15) fv_pairs.simps)
obtain t t' where f: "f = (t,t')" by (metis surj_pair)
hence "fv t  $\subseteq$  wfrestrictedvarsst S  $\cup$  (V  $\cup$  set X)" "fv t'  $\subseteq$  wfrestrictedvarsst S  $\cup$  (V  $\cup$  set
X)"
  using Cons.prem by auto
hence "fv (t  $\cdot$   $\delta$ )  $\subseteq$  wfrestrictedvarsst (S  $\cdot$ st  $\delta$ )  $\cup$  fvset ( $\delta$  ' (V  $\cup$  set X))"
  "fv (t'  $\cdot$   $\delta$ )  $\subseteq$  wfrestrictedvarsst (S  $\cdot$ st  $\delta$ )  $\cup$  fvset ( $\delta$  ' (V  $\cup$  set X))"
  using subst_apply_fv_subset_strand_trm2[OF _ assms(3)] P
  by blast+
  thus ?case using f IH by (auto simp add: subst_apply_pairs_def)
qed (simp add: subst_apply_pairs_def)
moreover have "fvset ( $\delta$  ' set X) = set X" using assms(4) Inequality by force
ultimately have "fvpairs (F  $\cdot$ pairs  $\delta$ ) - set X  $\subseteq$  wfrestrictedvarsst (S  $\cdot$ st  $\delta$ )  $\cup$  fvset ( $\delta$  ' V)"
  by fastforce
hence ?thesis using (P (x  $\cdot$ stp  $\delta$ ) = fvpairs (F  $\cdot$ pairs  $\delta$ ) - set X) by blast
}
ultimately show ?thesis by metis
qed

lemma strand_subst_fv_bounded_if_img_bounded:
  assumes "range_vars  $\delta \subseteq$  fvst S"
  shows "fvst (S  $\cdot$ st  $\delta$ )  $\subseteq$  fvst S"
using subst_sends_strand_fv_to_img[OF S  $\delta$ ] assms by blast

lemma strand_fv_subst_subset_if_subst_elim:
  assumes "subst_elim  $\delta$  v" and "v  $\in$  fvst S  $\vee$  bvvarsst S  $\cap$  (subst_domain  $\delta \cup$  range_vars  $\delta$ ) = {}"
  shows "v  $\notin$  fvst (S  $\cdot$ st  $\delta$ )"
proof (cases "v  $\in$  fvst S")
  case True thus ?thesis
  proof (induction S)
    case (Cons x S)
    have *: "v  $\notin$  fvstp (x  $\cdot$ stp  $\delta$ )"
    using assms(1)
    proof (cases x)
      case (Inequality X F)
      hence "subst_elim (rm_vars (set X)  $\delta$ ) v  $\vee$  v  $\in$  set X" using assms(1) by blast
      moreover have "fvstp (Inequality X F  $\cdot$ stp  $\delta$ ) = fvpairs (F  $\cdot$ pairs rm_vars (set X)  $\delta$ ) - set X"
      using Inequality by auto
      ultimately have "v  $\notin$  fvstp (Inequality X F  $\cdot$ stp  $\delta$ )"
      by (induct F) (auto simp add: subst_elim_def subst_apply_pairs_def)
      thus ?thesis using Inequality by simp
    qed (simp_all add: subst_elim_def)
    moreover have "v  $\notin$  fvst (S  $\cdot$ st  $\delta$ )" using Cons.IH
    proof (cases "v  $\in$  fvst S")
      case False
      moreover have "v  $\notin$  range_vars  $\delta$ "
      by (simp add: subst_elimD''[OF assms(1)] range_vars_alt_def)
      ultimately show ?thesis by (meson UnE subsetCE subst_sends_strand_fv_to_img)
    qed simp
    ultimately show ?case by auto
  qed simp
next
  case False
  thus ?thesis
  using assms fv_strand_subst'
  unfolding subst_elim_def
  by (metis (mono_tags, hide_lams) fvset.simps imageE mem_simps(8) subst_apply_term.simps(1))
qed

lemma strand_fv_subst_subset_if_subst_elim':
  assumes "subst_elim  $\delta$  v" "v  $\in$  fvst S" "range_vars  $\delta \subseteq$  fvst S"
  shows "fvst (S  $\cdot$ st  $\delta$ )  $\subset$  fvst S"
using strand_fv_subst_subset_if_subst_elim[OF assms(1)] assms(2)

```

```

strand_subst_fv_bounded_if_img_bounded[OF assms(3)]
by blast

lemma fv_ik_is_fv_rcv: "fvset (ikst S) =  $\bigcup$  (set (map fvrcv S))"
by (induct S rule: ikst.induct) auto

lemma fv_ik_subset_fv_st[simp]: "fvset (ikst S)  $\subseteq$  wfrestrictedvarsst S"
by (induct S rule: ikst.induct) auto

lemma fv_assignment_rhs_subset_fv_st[simp]: "fvset (assignment_rhsst S)  $\subseteq$  wfrestrictedvarsst S"
by (induct S rule: assignment_rhsst.induct) force+

lemma fv_ik_subset_fv_st'[simp]: "fvset (ikst S)  $\subseteq$  fvst S"
by (induct S rule: ikst.induct) auto

lemma ikst_var_is_fv: "Var x  $\in$  subtermsset (ikst A)  $\implies$  x  $\in$  fvst A"
by (meson fv_ik_subset_fv_st'[of A] fv_subset_subterms subsetCE term.set_intros(3))

lemma fv_assignment_rhs_subset_fv_st'[simp]: "fvset (assignment_rhsst S)  $\subseteq$  fvst S"
by (induct S rule: assignment_rhsst.induct) auto

lemma ikst_assignment_rhsst_wfrestrictedvars_subset:
  "fvset (ikst A  $\cup$  assignment_rhsst A)  $\subseteq$  wfrestrictedvarsst A"
using fv_ik_subset_fv_st[of A] fv_assignment_rhs_subset_fv_st[of A]
by simp+

lemma strand_step_id_subst[iff]: "x ·stp Var = x" by (cases x) auto

lemma strand_id_subst[iff]: "S ·st Var = S" using strand_step_id_subst by (induct S) auto

lemma strand_subst_vars_union_bound[simp]: "varsst (S ·st  $\delta$ )  $\subseteq$  varsst S  $\cup$  range_vars  $\delta$ "
proof (induction S)
  case (Cons x S)
  moreover have "varsstp (x ·stp  $\delta$ )  $\subseteq$  varsstp x  $\cup$  range_vars  $\delta$ " using subst_sends_fv_to_img[of _  $\delta$ ]
  proof (cases x)
    case (Inequality X F)
    define  $\delta'$  where " $\delta' \equiv \text{rm\_vars (set X) } \delta$ "
    have 0: "range_vars  $\delta' \subseteq$  range_vars  $\delta$ "
      using rm_vars_img[of "set X"  $\delta$ ]
      by (auto simp add:  $\delta'$ _def subst_domain_def range_vars_alt_def)

    have "varsstp (x ·stp  $\delta$ ) = fvpairs (F ·pairs  $\delta'$ )  $\cup$  set X" "varsstp x = fvpairs F  $\cup$  set X"
      using Inequality by (auto simp add:  $\delta'$ _def)
    moreover have "fvpairs (F ·pairs  $\delta'$ )  $\subseteq$  fvpairs F  $\cup$  range_vars  $\delta$ "
    proof (induction F)
      case (Cons f G)
      obtain t t' where f: "f = (t, t')" by moura
      hence "fvpairs (f#G ·pairs  $\delta'$ ) = fv (t ·  $\delta'$ )  $\cup$  fv (t' ·  $\delta'$ )  $\cup$  fvpairs (G ·pairs  $\delta'$ )"
        "fvpairs (f#G) = fv t  $\cup$  fv t'  $\cup$  fvpairs G"
        by (auto simp add: subst_apply_pairs_def)
      thus ?case
        using 0 Cons.IH subst_sends_fv_to_img[of t  $\delta'$ ] subst_sends_fv_to_img[of t'  $\delta'$ ]
        unfolding f by auto
    qed (simp add: subst_apply_pairs_def)
    ultimately show ?thesis by auto
  qed auto
  ultimately show ?case by auto
qed simp

lemma strand_vars_split:
  "varsst (S@S') = varsst S  $\cup$  varsst S'"
  "wfrestrictedvarsst (S@S') = wfrestrictedvarsst S  $\cup$  wfrestrictedvarsst S'"
  "fvst (S@S') = fvst S  $\cup$  fvst S'"

```



by auto

lemma bvars\_subst\_ident: "bvars<sub>st</sub> S = bvars<sub>st</sub> (S ·<sub>st</sub> δ)"

unfolding bvars<sub>st</sub>\_def

by (induct S) (simp\_all add: subst\_apply\_strand\_step\_def split: strand\_step.splits)

lemma strand\_subst\_subst\_idem:

assumes "subst\_idem δ" "subst\_domain δ ∪ range\_vars δ ⊆ fv<sub>st</sub> S" "subst\_domain ∅ ∩ fv<sub>st</sub> S = {}"  
 "range\_vars δ ∩ bvars<sub>st</sub> S = {}" "range\_vars ∅ ∩ bvars<sub>st</sub> S = {}"

shows "(S ·<sub>st</sub> δ) ·<sub>st</sub> ∅ = (S ·<sub>st</sub> δ)"

and "(S ·<sub>st</sub> δ) ·<sub>st</sub> (∅ ∘<sub>s</sub> δ) = (S ·<sub>st</sub> δ)"

proof -

from assms(2,3) have "fv<sub>st</sub> (S ·<sub>st</sub> δ) ∩ subst\_domain ∅ = {}"

using subst\_sends\_strand\_fv\_to\_img[of S δ] by blast

thus "(S ·<sub>st</sub> δ) ·<sub>st</sub> ∅ = (S ·<sub>st</sub> δ)" by blast

thus "(S ·<sub>st</sub> δ) ·<sub>st</sub> (∅ ∘<sub>s</sub> δ) = (S ·<sub>st</sub> δ)"

by (metis assms(1,4,5) bvars\_subst\_ident strand\_subst\_comp subst\_idem\_def)

qed

lemma strand\_subst\_img\_bound:

assumes "subst\_domain δ ∪ range\_vars δ ⊆ fv<sub>st</sub> S"

and "(subst\_domain δ ∪ range\_vars δ) ∩ bvars<sub>st</sub> S = {}"

shows "range\_vars δ ⊆ fv<sub>st</sub> (S ·<sub>st</sub> δ)"

proof -

have "subst\_domain δ ⊆ ⋃(set (map fv<sub>stp</sub> S))" by (metis (no\_types) fv<sub>st</sub>\_def Un\_subset\_iff assms(1))  
 thus ?thesis

unfolding range\_vars\_alt\_def fv<sub>st</sub>\_def

by (metis subst\_range.simps fv\_set\_mono fv\_strand\_subst Int\_commute assms(2) image\_Un  
 le\_iff\_sup)

qed

lemma strand\_subst\_img\_bound':

assumes "subst\_domain δ ∪ range\_vars δ ⊆ vars<sub>st</sub> S"

and "(subst\_domain δ ∪ range\_vars δ) ∩ bvars<sub>st</sub> S = {}"

shows "range\_vars δ ⊆ vars<sub>st</sub> (S ·<sub>st</sub> δ)"

proof -

have "(subst\_domain δ ∪ fv<sub>set</sub> (δ ' subst\_domain δ)) ∩ vars<sub>st</sub> S =  
 subst\_domain δ ∪ fv<sub>set</sub> (δ ' subst\_domain δ)"

using assms(1) by (metis inf.absorb\_iff1 range\_vars\_alt\_def subst\_range.simps)

hence "range\_vars δ ⊆ fv<sub>st</sub> (S ·<sub>st</sub> δ)"

using vars\_snd\_rcv\_strand fv\_snd\_rcv\_strand assms(2) strand\_subst\_img\_bound

unfolding range\_vars\_alt\_def

by (metis (no\_types) inf\_le2 inf\_sup\_distrib1 subst\_range.simps sup\_bot.right\_neutral)

thus "range\_vars δ ⊆ vars<sub>st</sub> (S ·<sub>st</sub> δ)"

by (metis fv\_snd\_rcv\_strand le\_supI1 vars\_snd\_rcv\_strand)

qed

lemma strand\_subst\_all\_fv\_subset:

assumes "fv t ⊆ fv<sub>st</sub> S" "(subst\_domain δ ∪ range\_vars δ) ∩ bvars<sub>st</sub> S = {}"

shows "fv (t · δ) ⊆ fv<sub>st</sub> (S ·<sub>st</sub> δ)"

using assms by (metis fv\_strand\_subst' Int\_commute subst\_apply\_fv\_subset)

lemma strand\_subst\_not\_dom\_fixed:

assumes "v ∈ fv<sub>st</sub> S" and "v ∉ subst\_domain δ"

shows "v ∈ fv<sub>st</sub> (S ·<sub>st</sub> δ)"

using assms

proof (induction S)

case (Cons x S')

have 1: "⋀X. v ∉ subst\_domain (rm\_vars (set X) δ)"

using Cons.prem(2) rm\_vars\_dom\_subset by force

show ?case

proof (cases "v ∈ fv<sub>st</sub> S'")

```

    case True thus ?thesis using Cons.IH[OF _ Cons.prem2] by auto
  next
    case False
    hence 2: "v ∈ fvstp x" using Cons.prem1 by simp
    hence "v ∈ fvstp (x ·stp δ)" using Cons.prem2 subst_not_dom_fixed
    proof (cases x)
      case (Inequality X F)
      hence "v ∈ fvpairs F - set X" using 2 by simp
      hence "v ∈ fvpairs (F ·pairs rm_vars (set X) δ)"
        using subst_not_dom_fixed[OF _ 1]
        by (induct F) (auto simp add: subst_apply_pairs_def)
      thus ?thesis using Inequality 2 by auto
    qed (force simp add: subst_domain_def)+
    thus ?thesis by auto
  qed
qed simp

lemma strand_vars_unfold: "v ∈ varsst S ⇒ ∃ S' x S''. S = S'@x#S'' ∧ v ∈ varsstp x"
proof (induction S)
  case (Cons x S) thus ?case
  proof (cases "v ∈ varsstp x")
    case True thus ?thesis by blast
  next
    case False
    hence "v ∈ varsst S" using Cons.prem by auto
    thus ?thesis using Cons.IH by (metis append_Cons)
  qed
qed simp

lemma strand_fv_unfold: "v ∈ fvst S ⇒ ∃ S' x S''. S = S'@x#S'' ∧ v ∈ fvstp x"
proof (induction S)
  case (Cons x S) thus ?case
  proof (cases "v ∈ fvstp x")
    case True thus ?thesis by blast
  next
    case False
    hence "v ∈ fvst S" using Cons.prem by auto
    thus ?thesis using Cons.IH by (metis append_Cons)
  qed
qed simp

lemma subterm_if_in_strand_ik:
  "t ∈ ikst S ⇒ ∃ t'. Receive t' ∈ set S ∧ t ⊆ t'"
by (induct S rule: ikst_induct) auto

lemma fv_subset_if_in_strand_ik:
  "t ∈ ikst S ⇒ fv t ⊆ ⋃ (set (map fvrcv S))"
proof -
  assume "t ∈ ikst S"
  then obtain t' where "Receive t' ∈ set S" "t ⊆ t'" by (metis subterm_if_in_strand_ik)
  hence "fv t ⊆ fv t'" by (simp add: subtermeq_vars_subset)
  thus ?thesis using in_strand_fv_subset_rcv[OF (Receive t' ∈ set S)] by auto
qed

lemma fv_subset_if_in_strand_ik':
  "t ∈ ikst S ⇒ fv t ⊆ fvst S"
using fv_subset_if_in_strand_ik[of t S] fv_snd_rcv_strand_subset(2)[of S] by blast

lemma vars_subset_if_in_strand_ik2:
  "t ∈ ikst S ⇒ fv t ⊆ wfrestrictedvarsst S"
using fv_subset_if_in_strand_ik[of t S] vars_snd_rcv_strand_subset2(2)[of S] by blast

```

## 3.1.3 Lemmata: Simple Strands

```

lemma simple_Cons[dest]: "simple (s#S)  $\implies$  simple S"
unfolding simple_def by auto

lemma simple_split[dest]:
  assumes "simple (S@S'"
  shows "simple S" "simple S'"
using assms unfolding simple_def by auto

lemma simple_append[intro]: "[simple S; simple S']  $\implies$  simple (S@S'"
unfolding simple_def by auto

lemma simple_append_sym[sym]: "simple (S@S')  $\implies$  simple (S'@S)" by auto

lemma not_simple_if_snd_fun: "( $\exists$  S' S'' f X. S = S'@Send (Fun f X)#S'')  $\implies$   $\neg$ simple S"
unfolding simple_def by auto

lemma not_list_all_elim: " $\neg$ list_all P A  $\implies$   $\exists$  B x C. A = B@x#C  $\wedge$   $\neg$ P x  $\wedge$  list_all P B"
proof (induction A rule: List.rev_induct)
  case (snoc a A)
  show ?case
  proof (cases "list_all P A")
    case True
    thus ?thesis using snoc.prem by auto
  next
    case False
    then obtain B x C where "A = B@x#C" " $\neg$ P x" "list_all P B" using snoc.IH[OF False] by auto
    thus ?thesis by auto
  qed
qed simp

lemma not_simple_stp_elim:
  assumes " $\neg$ simplestp x"
  shows "( $\exists$  f T. x = Send (Fun f T))  $\vee$ 
    ( $\exists$  a t t'. x = Equality a t t')  $\vee$ 
    ( $\exists$  X F. x = Inequality X F  $\wedge$   $\neg$ ( $\exists$  I. ineq_model I X F))"
using assms by (cases x) (fastforce elim: simplestp.elim)+

lemma not_simple_elim:
  assumes " $\neg$ simple S"
  shows "( $\exists$  A B f T. S = A@Send (Fun f T)#B  $\wedge$  simple A)  $\vee$ 
    ( $\exists$  A B a t t'. S = A@Equality a t t'#B  $\wedge$  simple A)  $\vee$ 
    ( $\exists$  A B X F. S = A@Inequality X F#B  $\wedge$   $\neg$ ( $\exists$  I. ineq_model I X F))"
by (metis assms not_list_all_elim not_simple_stp_elim simple_def)

lemma simple_fun_prefix_unique:
  assumes "A = S@Send (Fun f X)#S'" "simple S"
  shows " $\forall$  T g Y T'. A = T@Send (Fun g Y)#T'  $\wedge$  simple T  $\longrightarrow$  S = T  $\wedge$  f = g  $\wedge$  X = Y  $\wedge$  S' = T'"
proof -
  { fix T g Y T' assume *: "A = T@Send (Fun g Y)#T'" "simple T"
    { assume "length S < length T" hence False using assms *
      by (metis id_take_nth_drop not_simple_if_snd_fun nth_append nth_append_length)
    }
    moreover
    { assume "length S > length T" hence False using assms *
      by (metis id_take_nth_drop not_simple_if_snd_fun nth_append nth_append_length)
    }
    ultimately have "S = T" using assms * by (meson List.append_eq_append_conv linorder_neqE_nat)
  }
  thus ?thesis using assms(1) by blast
qed

```

```

lemma simple_snd_is_var: "[[Send t ∈ set S; simple S]] ⇒ ∃ v. t = Var v"
unfolding simple_def
by (metis list_all_append list_all_simps(1) simple_step.elims(2) split_list_first
    strand_step.distinct(1) strand_step.distinct(5) strand_step.inject(1))

```

### 3.1.4 Lemmata: Strand Measure

```

lemma measure_st_wellfounded: "wf measure_st" unfolding measure_st_def by simp

```

```

lemma strand_size_append[iff]: "size_st (S@S') = size_st S + size_st S'"
by (induct S) (auto simp add: size_st_def)

```

```

lemma strand_size_map_fun_lt[simp]:
  "size_st (map Send X) < size (Fun f X)"
  "size_st (map Send X) < size_st [Send (Fun f X)]"
  "size_st (map Send X) < size_st [Receive (Fun f X)]"
by (induct X) (auto simp add: size_st_def)

```

```

lemma strand_size_rm_fun_lt[simp]:
  "size_st (S@S') < size_st (S@Send (Fun f X)#S'"
  "size_st (S@S') < size_st (S@Receive (Fun f X)#S'"
by (induct S) (auto simp add: size_st_def)

```

```

lemma strand_fv_card_map_fun_eq:
  "card (fv_st (S@Send (Fun f X)#S')) = card (fv_st (S@(map Send X)@S'))"
proof -
  have "fv_st (S@Send (Fun f X)#S') = fv_st (S@(map Send X)@S'" by auto
  thus ?thesis by simp
qed

```

```

lemma strand_fv_card_rm_fun_le[simp]: "card (fv_st (S@S')) ≤ card (fv_st (S@Send (Fun f X)#S'))"
by (force intro: card_mono)

```

```

lemma strand_fv_card_rm_eq_le[simp]: "card (fv_st (S@S')) ≤ card (fv_st (S@Equality a t t'#S'))"
by (force intro: card_mono)

```

### 3.1.5 Lemmata: Well-formed Strands

```

lemma wf_prefix[dest]: "wf_st V (S@S') ⇒ wf_st V S"
by (induct S rule: wf_st.induct) auto

```

```

lemma wf_vars_mono[simp]: "wf_st V S ⇒ wf_st (V ∪ W) S"
proof (induction S arbitrary: V)
  case (Cons x S) thus ?case
  proof (cases x)
    case (Send t)
    hence "wf_st (V ∪ fv t ∪ W) S" using Cons.prem(1) Cons.IH by simp
    thus ?thesis using Send by (simp add: sup_commute sup_left_commute)
  next
    case (Equality a t t')
    show ?thesis
    proof (cases a)
      case Assign
      hence "wf_st (V ∪ fv t ∪ W) S" "fv t' ⊆ V ∪ W" using Equality Cons.prem(1) Cons.IH by auto
      thus ?thesis using Equality Assign by (simp add: sup_commute sup_left_commute)
    next
      case Check
      thus ?thesis using Equality Cons by auto
    qed
  qed auto
qed simp

```

```

lemma wf_stI[intro]: "wfrestrictedvars_st S ⊆ V ⇒ wf_st V S"
proof (induction S)

```

```

case (Cons x S) thus ?case
proof (cases x)
  case (Send t)
  hence "wfst V S" "V ∪ fv t = V" using Cons by auto
  thus ?thesis using Send by simp
next
  case (Equality a t t')
  show ?thesis
  proof (cases a)
    case Assign
    hence "wfst V S" "fv t' ⊆ V" using Equality Cons by auto
    thus ?thesis using wf_vars_mono Equality Assign by simp
  next
    case Check thus ?thesis using Equality Cons by auto
  qed
qed simp_all
qed simp

lemma wfstI'[intro]: "⋃(fvrcv ' set S) ∪ ⋃(fvreq assign ' set S) ⊆ V ⇒ wfst V S"
proof (induction S)
  case (Cons x S) thus ?case
  proof (cases x)
    case (Equality a t t') thus ?thesis using Cons by (cases a) auto
  qed simp_all
qed simp

lemma wf_append_exec: "wfst V (S@S') ⇒ wfst (V ∪ wfvarsoccsst S) S'"
proof (induction S arbitrary: V)
  case (Cons x S V) thus ?case
  proof (cases x)
    case (Send t)
    hence "wfst (V ∪ fv t ∪ wfvarsoccsst S) S'" using Cons.prem Cons.IH by simp
    thus ?thesis using Send by (auto simp add: sup_assoc)
  next
    case (Equality a t t') show ?thesis
    proof (cases a)
      case Assign
      hence "wfst (V ∪ fv t ∪ wfvarsoccsst S) S'" using Equality Cons.prem Cons.IH by auto
      thus ?thesis using Equality Assign by (auto simp add: sup_assoc)
    next
      case Check
      hence "wfst (V ∪ wfvarsoccsst S) S'" using Equality Cons.prem Cons.IH by auto
      thus ?thesis using Equality Check by (auto simp add: sup_assoc)
    qed
  qed auto
qed simp

lemma wf_append_suffix:
  "wfst V S ⇒ wfrestrictedvarsst S' ⊆ wfrestrictedvarsst S ∪ V ⇒ wfst V (S@S')"
proof (induction V S rule: wfst_induct)
  case (ConsSnd V t S)
  hence *: "wfst (V ∪ fv t) S" by simp_all
  hence "wfrestrictedvarsst S' ⊆ wfrestrictedvarsst S ∪ (V ∪ fv t)"
  using ConsSnd.prem(2) by fastforce
  thus ?case using ConsSnd.IH * by simp
next
  case (ConsRcv V t S)
  hence *: "fv t ⊆ V" "wfst V S" by simp_all
  hence "wfrestrictedvarsst S' ⊆ wfrestrictedvarsst S ∪ V"
  using ConsRcv.prem(2) by fastforce
  thus ?case using ConsRcv.IH * by simp
next
  case (ConsEq V t t' S)

```

```

hence *: "fv t'  $\subseteq$  V" "wfst (V  $\cup$  fv t) S" by simp_all
moreover have "varsstp (Equality Assign t t') = fv t  $\cup$  fv t'"
  by simp
moreover have "wfrestrictedvarsst (Equality Assign t t'#S) = fv t  $\cup$  fv t'  $\cup$  wfrestrictedvarsst S"
  by auto
ultimately have "wfrestrictedvarsst S'  $\subseteq$  wfrestrictedvarsst S  $\cup$  (V  $\cup$  fv t)"
  using ConsEq.prem(2) by blast
thus ?case using ConsEq.IH * by simp
qed (simp_all add: wfstI)

```

```

lemma wf_append_suffix':
  assumes "wfst V S"
  and " $\bigcup$  (fvrcv ' set S')  $\cup$   $\bigcup$  (fvreq assign ' set S')  $\subseteq$  wfvarsoccsst S  $\cup$  V"
  shows "wfst V (S@S')"
using assms
proof (induction V S rule: wfst-induct)
  case (ConsSnd V t S)
  hence *: "wfst (V  $\cup$  fv t) S" by simp_all
  have "wfvarsoccsst (send⟨t⟩st#S) = fv t  $\cup$  wfvarsoccsst S"
    unfolding wfvarsoccsst-def by simp
  hence " $(\bigcup a \in \text{set } S'. \text{fv}_{rcv} a) \cup (\bigcup a \in \text{set } S'. \text{fv}_{req} \text{ assign } a) \subseteq \text{wfvarsoccs}_{st} S \cup (V \cup \text{fv } t)$ "
    using ConsSnd.prem(2) unfolding wfvarsoccsst-def by auto
  thus ?case using ConsSnd.IH[OF *] by auto
next
  case (ConsEq V t t' S)
  hence *: "fv t'  $\subseteq$  V" "wfst (V  $\cup$  fv t) S" by simp_all
  have "wfvarsoccsst (<assign: t  $\doteq$  t'>st#S) = fv t  $\cup$  wfvarsoccsst S"
    unfolding wfvarsoccsst-def by simp
  hence " $(\bigcup a \in \text{set } S'. \text{fv}_{rcv} a) \cup (\bigcup a \in \text{set } S'. \text{fv}_{req} \text{ assign } a) \subseteq \text{wfvarsoccs}_{st} S \cup (V \cup \text{fv } t)$ "
    using ConsEq.prem(2) unfolding wfvarsoccsst-def by auto
  thus ?case using ConsEq.IH[OF *(2)] *(1) by auto
qed (auto simp add: wfstI')

```

```

lemma wf_send_compose: "wfst V (S@⟨map Send X⟩@S') = wfst V (S@Send (Fun f X)#S')"
proof (induction S arbitrary: V)
  case Nil thus ?case
  proof (induction X arbitrary: V)
    case (Cons y Y) thus ?case by (simp add: sup_assoc)
  qed simp
next
  case (Cons s S) thus ?case
  proof (cases s)
    case (Equality ac t t') thus ?thesis using Cons by (cases ac) auto
  qed auto
qed

```

```

lemma wf_snd_append[iff]: "wfst V (S@[Send t]) = wfst V S"
by (induct S rule: wfst.induct) simp_all

```

```

lemma wf_snd_append': "wfst V S  $\implies$  wfst V (Send t#S)"
by simp

```

```

lemma wf_rcv_append[dest]: "wfst V (S@Receive t#S')  $\implies$  wfst V (S@S')"
by (induct S rule: wfst.induct) simp_all

```

```

lemma wf_rcv_append'[intro]:
  " $\llbracket \text{wf}_{st} V (S@S'); \text{fv } t \subseteq \text{wfrestrictedvars}_{st} S \cup V \rrbracket \implies \text{wf}_{st} V (S@Receive t\#S')$ "
proof (induction S rule: wfst-induct)
  case (ConsRcv V t' S)
  hence "wfst V (S@S'" "fv t  $\subseteq$  wfrestrictedvarsst S  $\cup$  V"
    by auto+
  thus ?case using ConsRcv by auto
next

```

```

case (ConsEq V t' t'' S)
hence "fv t''  $\subseteq$  V" by simp
moreover have
  "wfrestrictedvarsst (Equality Assign t' t''#S) = fv t'  $\cup$  fv t''  $\cup$  wfrestrictedvarsst S"
  by auto
ultimately have "fv t  $\subseteq$  wfrestrictedvarsst S  $\cup$  (V  $\cup$  fv t')"
  using ConsEq.prem2 by blast
thus ?case using ConsEq by auto
qed auto

lemma wf_rcv_append''[intro]: "[wfst V S; fv t  $\subseteq$   $\bigcup$  (set (map fvsnd S))]  $\implies$  wfst V (S@[Receive t])"
by (induct S)
  (simp, metis vars_snd_rcv_strand_subset2(1) append_Nil2 le_supI1 order_trans wf_rcv_append')

lemma wf_rcv_append'''[intro]: "[wfst V S; fv t  $\subseteq$  wfrestrictedvarsst S  $\cup$  V]  $\implies$  wfst V (S@[Receive t])"
by (simp add: wf_rcv_append'[of _ _ "[]"])

lemma wf_eq_append[dest]: "wfst V (S@Equality a t t'#S')  $\implies$  fv t  $\subseteq$  wfrestrictedvarsst S  $\cup$  V  $\implies$ 
wfst V (S@S)"
proof (induction S rule: wfst-induct)
  case (Nil V)
  hence "wfst (V  $\cup$  fv t) S'" by (cases a) auto
  moreover have "V  $\cup$  fv t = V" using Nil by auto
  ultimately show ?case by simp
next
  case (ConsRcv V u S)
  hence "wfst V (S @ Equality a t t' # S'" "fv t  $\subseteq$  wfrestrictedvarsst S  $\cup$  V" "fv u  $\subseteq$  V"
    by fastforce+
  hence "wfst V (S@S'" using ConsRcv.IH by auto
  thus ?case using (fv u  $\subseteq$  V) by simp
next
  case (ConsEq V u u' S)
  hence "wfst (V  $\cup$  fv u) (S@Equality a t t'#S'" "fv t  $\subseteq$  wfrestrictedvarsst S  $\cup$  (V  $\cup$  fv u)" "fv u'
 $\subseteq$  V"
    by auto
  hence "wfst (V  $\cup$  fv u) (S@S'" using ConsEq.IH by auto
  thus ?case using (fv u'  $\subseteq$  V) by simp
qed auto

lemma wf_eq_append'[intro]:
  "[wfst V (S@S'); fv t'  $\subseteq$  wfrestrictedvarsst S  $\cup$  V]  $\implies$  wfst V (S@Equality a t t'#S'"
proof (induction S rule: wfst-induct)
  case Nil thus ?case by (cases a) auto
next
  case (ConsEq V u u' S)
  hence "wfst (V  $\cup$  fv u) (S@S'" "fv t'  $\subseteq$  wfrestrictedvarsst S  $\cup$  V  $\cup$  fv u"
    by fastforce+
  thus ?case using ConsEq by auto
next
  case (ConsEq2 V u u' S)
  hence "wfst V (S@S'" by auto
  thus ?case using ConsEq2 by auto
next
  case (ConsRcv V u S)
  hence "wfst V (S@S'" "fv t'  $\subseteq$  wfrestrictedvarsst S  $\cup$  V"
    by fastforce+
  thus ?case using ConsRcv by auto
next
  case (ConsSnd V u S)
  hence "wfst (V  $\cup$  fv u) (S@S'" "fv t'  $\subseteq$  wfrestrictedvarsst S  $\cup$  (V  $\cup$  fv u)"
    by fastforce+
  thus ?case using ConsSnd by auto

```

qed auto

lemma wf\_eq\_append''[intro]:

" $\llbracket wf_{st} V (S@S'); fv\ t' \subseteq wfvarsoccs_{st}\ S \cup V \rrbracket \implies wf_{st} V (S@[Equality\ a\ t\ t']@S')$ "

proof (induction S rule: wf\_st\_induct)

case Nil thus ?case by (cases a) auto

next

case (ConsEq V u u' S)

hence " $wf_{st} (V \cup fv\ u) (S@S')$ " " $fv\ t' \subseteq wfvarsoccs_{st}\ S \cup V \cup fv\ u$ " by fastforce+

thus ?case using ConsEq by auto

next

case (ConsEq2 V u u' S)

hence " $wf_{st} (V \cup fv\ u) (S@S')$ " " $fv\ t' \subseteq wfvarsoccs_{st}\ S \cup V \cup fv\ u$ " by fastforce+

thus ?case using ConsEq2 by auto

next

case (ConsRcv V u S)

hence " $wf_{st} V (S@S')$ " " $fv\ t' \subseteq wfvarsoccs_{st}\ S \cup V$ " by fastforce+

thus ?case using ConsRcv by auto

next

case (ConsSnd V u S)

hence " $wf_{st} (V \cup fv\ u) (S@S')$ " " $fv\ t' \subseteq wfvarsoccs_{st}\ S \cup (V \cup fv\ u)$ " by auto

thus ?case using ConsSnd by auto

qed auto

lemma wf\_eq\_append'''[intro]:

" $\llbracket wf_{st} V S; fv\ t' \subseteq wfrestrictedvars_{st}\ S \cup V \rrbracket \implies wf_{st} V (S@[Equality\ a\ t\ t'])$ "

by (simp add: wf\_eq\_append'[of \_ \_ "[]"])

lemma wf\_eq\_check\_append[dest]: " $wf_{st} V (S@[Equality\ Check\ t\ t'\#S']) \implies wf_{st} V (S@S')$ "

by (induct S rule: wf\_st\_induct) simp\_all

lemma wf\_eq\_check\_append'[intro]: " $wf_{st} V (S@S') \implies wf_{st} V (S@[Equality\ Check\ t\ t'\#S'])$ "

by (induct S rule: wf\_st\_induct) auto

lemma wf\_eq\_check\_append''[intro]: " $wf_{st} V S \implies wf_{st} V (S@[Equality\ Check\ t\ t'])$ "

by (induct S rule: wf\_st\_induct) auto

lemma wf\_ineq\_append[dest]: " $wf_{st} V (S@[Inequality\ X\ F\#S']) \implies wf_{st} V (S@S')$ "

by (induct S rule: wf\_st\_induct) simp\_all

lemma wf\_ineq\_append'[intro]: " $wf_{st} V (S@S') \implies wf_{st} V (S@[Inequality\ X\ F\#S'])$ "

by (induct S rule: wf\_st\_induct) auto

lemma wf\_ineq\_append''[intro]: " $wf_{st} V S \implies wf_{st} V (S@[Inequality\ X\ F])$ "

by (induct S rule: wf\_st\_induct) auto

lemma wf\_rcv\_fv\_single[elim]: " $wf_{st} V (Receive\ t\#S') \implies fv\ t \subseteq V$ "

by simp

lemma wf\_rcv\_fv: " $wf_{st} V (S@Receive\ t\#S') \implies fv\ t \subseteq wfvarsoccs_{st}\ S \cup V$ "

by (induct S arbitrary: V) (auto split!: strand\_step.split poscheckvariant.split)

lemma wf\_eq\_fv: " $wf_{st} V (S@[Equality\ Assign\ t\ t'\#S']) \implies fv\ t' \subseteq wfvarsoccs_{st}\ S \cup V$ "

by (induct S arbitrary: V) (auto split!: strand\_step.split poscheckvariant.split)

lemma wf\_simple\_fv\_occurrence:

assumes " $wf_{st}\ \{\}\ S$ " "simple S" " $v \in wfrestrictedvars_{st}\ S$ "

shows " $\exists S_{pre}\ S_{suf}. S = S_{pre}@Send\ (Var\ v)\#S_{suf} \wedge v \notin wfrestrictedvars_{st}\ S_{pre}$ "

using assms

proof (induction S rule: List.rev\_induct)

case (snoc x S)

from  $\langle wf_{st}\ \{\}\ (S@[x]) \rangle$  have " $wf_{st}\ \{\}\ S$ " " $wf_{st}\ (wfrestrictedvars_{st}\ S)\ [x]$ "

using wf\_append\_exec[THEN wf\_vars\_mono, of "{}" S "[x]" " $wfrestrictedvars_{st}\ S - wfvarsoccs_{st}\ S$ "]



```

vars_snd_rcv_strand_subset2(4) [of S]
Diff_partition [of "wfvarsoccsst S" "wfrestrictedvarsst S"]
by auto
from ⟨simple (S@[x])⟩ have "simple S" "simplestp x" unfolding simple_def by auto

show ?case
proof (cases "v ∈ wfrestrictedvarsst S")
  case False
  show ?thesis
  proof (cases x)
    case (Receive t)
    hence "fv t ⊆ wfrestrictedvarsst S" using ⟨wfst (wfrestrictedvarsst S) [x]⟩ by simp
    hence "v ∈ wfrestrictedvarsst S"
    using ⟨v ∈ wfrestrictedvarsst (S@[x])⟩ ⟨x = Receive t⟩
    by auto
    thus ?thesis using ⟨x = Receive t⟩ snoc.IH[OF ⟨wfst {} S⟩ ⟨simple S⟩] by fastforce
  next
    case (Send t)
    hence "v ∈ varsstp x" using ⟨v ∈ wfrestrictedvarsst (S@[x])⟩ False by auto
    from Send obtain w where "t = Var w" using ⟨simplestp x⟩ by (cases t) simp_all
    hence "v = w" using ⟨x = Send t⟩ ⟨v ∈ varsstp x⟩ by simp
    thus ?thesis using ⟨x = Send t⟩ ⟨v ∉ wfrestrictedvarsst S⟩ ⟨t = Var w⟩ by auto
  next
    case (Equality ac t t') thus ?thesis using snoc.prems(2) unfolding simple_def by auto
  next
    case (Inequality t t') thus ?thesis using False snoc.prems(3) by auto
  qed
qed (use snoc.IH[OF ⟨wfst {} S⟩ ⟨simple S⟩] in fastforce)
qed simp

lemma Unifier_strand_fv_subset:
  assumes g_in_ik: "t ∈ ikst S"
  and δ: "Unifier δ (Fun f X) t"
  and disj: "bvarsst S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "fv (Fun f X · δ) ⊆ ⋃ (set (map fvrcv (S ·st δ)))"
by (metis (no_types) fv_subset_if_in_strand_ik[OF g_in_ik]
    disj δ fv_strand_subst subst_apply_fv_subset)

lemma wfst_induct' [consumes 1, case_names Nil ConsSnd ConsRcv ConsEq ConsEq2 ConsIneq]:
  fixes S: "('a, 'b) strand"
  assumes "wfst V S"
  "P []"
  "⋀t S. [wfst V S; P S] ⟹ P (S@[Send t])"
  "⋀t S. [wfst V S; P S; fv t ⊆ V ∪ wfvarsoccsst S] ⟹ P (S@[Receive t])"
  "⋀t t' S. [wfst V S; P S; fv t' ⊆ V ∪ wfvarsoccsst S] ⟹ P (S@[Equality Assign t t'])"
  "⋀t t' S. [wfst V S; P S] ⟹ P (S@[Equality Check t t'])"
  "⋀X F S. [wfst V S; P S] ⟹ P (S@[Inequality X F])"
  shows "P S"
using assms
proof (induction S rule: List.rev_induct)
  case (snoc x S)
  hence *: "wfst V S" "wfst (V ∪ wfvarsoccsst S) [x]" by (metis wf_prefix, metis wf_append_exec)
  have IH: "P S" using snoc.IH[OF *(1)] snoc.prems by auto
  note ** = snoc.prems(3,4,5,6,7)[OF *(1) IH] *(2)
  show ?case using **(1,2,4,5,6)
  proof (cases x)
    case (Equality ac t t')
    then show ?thesis using **(3,4,6) by (cases ac) auto
  qed auto
qed simp

lemma wf_subst_apply:
  "wfst V S ⟹ wfst (fvset (δ ' V)) (S ·st δ)"

```

```

proof (induction S arbitrary: V rule: wfst-induct)
  case (ConsRcv V t S)
  hence "wfst V S" "fv t ⊆ V" by simp_all
  hence "wfst (fvset (δ ' V)) (S ·st δ)" "fv (t · δ) ⊆ fvset (δ ' V)"
    using ConsRcv.IH subst_apply_fv_subset by simp_all
  thus ?case by simp
next
  case (ConsSnd V t S)
  hence "wfst (V ∪ fv t) S" by simp
  hence "wfst (fvset (δ ' (V ∪ fv t))) (S ·st δ)" using ConsSnd.IH by metis
  hence "wfst (fvset (δ ' V) ∪ fv (t · δ)) (S ·st δ)" using subst_apply_fv_union by metis
  thus ?case by simp
next
  case (ConsEq V t t' S)
  hence "wfst (V ∪ fv t) S" "fv t' ⊆ V" by auto
  hence "wfst (fvset (δ ' (V ∪ fv t))) (S ·st δ)" and *: "fv (t' · δ) ⊆ fvset (δ ' V)"
    using ConsEq.IH subst_apply_fv_subset by force+
  hence "wfst (fvset (δ ' V) ∪ fv (t · δ)) (S ·st δ)" using subst_apply_fv_union by metis
  thus ?case using * by simp
qed simp_all

lemma wf_unify:
  assumes wf: "wfst V (S@Send (Fun f X)#S')"
  and g_in_ik: "t ∈ ikst S"
  and δ: "Unifier δ (Fun f X) t"
  and disj: "bvarsst (S@Send (Fun f X)#S') ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "wfst (fvset (δ ' V)) ((S@S') ·st δ)"
using assms
proof (induction S' arbitrary: V rule: List.rev_induct)
  case (snoc x S' V)
  have fun_fv_bound: "fv (Fun f X · δ) ⊆ ⋃ (set (map fvrcv (S ·st δ)))"
    using snoc.prem(4) bvarsst-split Unifier_strand_fv_subset[OF g_in_ik δ] by auto
  hence "fv (Fun f X · δ) ⊆ fvset (ikst (S ·st δ))" using fv_ik_is_fv_rcv by metis
  hence "fv (Fun f X · δ) ⊆ wfrestrictedvarsst (S ·st δ)" using fv_ik_subset_fv_st[of "S ·st δ"] by blast
  hence *: "fv ((Fun f X) · δ) ⊆ wfrestrictedvarsst ((S@S') ·st δ)" by fastforce

  from snoc.prem(1) have "wfst V (S@Send (Fun f X)#S')"
    using wf_prefix[of V "S@Send (Fun f X)#S'" "[x]"] by simp
  hence **: "wfst (fvset (δ ' V)) ((S@S') ·st δ)"
    using snoc.IH[OF _ snoc.prem(2,3)] snoc.prem(4) by auto

  from snoc.prem(1) have ***: "wfst (V ∪ wfvarsoccsst (S@Send (Fun f X)#S')) [x]"
    using wf_append_exec[of V "(S@Send (Fun f X)#S')" "[x]"] by simp

  from snoc.prem(4) have disj':
    "bvarsst (S@S') ∩ (subst_domain δ ∪ range_vars δ) = {}"
    "set (bvarsstp x) ∩ (subst_domain δ ∪ range_vars δ) = {}"
  by auto

  show ?case
  proof (cases x)
    case (Send t)
    thus ?thesis using wf_snd_append[of "fvset (δ ' V)" "(S@S') ·st δ"] ** by auto
  next
    case (Receive t)
    hence "fvstp x ⊆ V ∪ wfvarsoccsst (S@Send (Fun f X)#S')" using *** by auto
    hence "fvstp x ⊆ V ∪ wfrestrictedvarsst (S@Send (Fun f X)#S')"
      using vars_snd_rcv_strand_subset2(4)[of "S@Send (Fun f X)#S'"] by blast
    hence "fvstp x ⊆ V ∪ fv (Fun f X) ∪ wfrestrictedvarsst (S@S')" by auto
    hence "fvstp (x ·stp δ) ⊆ fvset (δ ' V) ∪ fv ((Fun f X) · δ) ∪ wfrestrictedvarsst ((S@S') ·st δ)"
      by (metis (no_types) inf_sup_aci(5) subst_apply_fv_subset_strand2 subst_apply_fv_union disj')
    hence "fvstp (x ·stp δ) ⊆ fvset (δ ' V) ∪ wfrestrictedvarsst ((S@S') ·st δ)" using * by blast
  end
end

```

```

hence "fv (t · δ) ⊆ wfrestrictedvarsst ((S@S') ·st δ) ∪ fvset (δ ' V) " using ⟨x = Receive t⟩ by
auto
hence "wfst (fvset (δ ' V)) (((S@S') ·st δ)@[Receive (t · δ)])"
  using wf_rcv_append'''[OF **, of "t · δ"] by metis
thus ?thesis using ⟨x = Receive t⟩ by auto
next
case (Equality ac s s') show ?thesis
proof (cases ac)
  case Assign
  hence "fv s' ⊆ V ∪ wfvarsoccsst (S@Send (Fun f X)#S') " using Equality *** by auto
  hence "fv s' ⊆ V ∪ wfrestrictedvarsst (S@Send (Fun f X)#S') "
    using vars_snd_rcv_strand_subset2(4)[of "S@Send (Fun f X)#S'"] by blast
  hence "fv s' ⊆ V ∪ fv (Fun f X) ∪ wfrestrictedvarsst (S@S') " by auto
  moreover have "fv s' = fvreq ac x" "fv (s' · δ) = fvreq ac (x ·stp δ) "
    using Equality by simp_all
  ultimately have "fv (s' · δ) ⊆ fvset (δ ' V) ∪ fv (Fun f X · δ) ∪ wfrestrictedvarsst ((S@S') ·st
δ)"
    using subst_apply_fv_subset_strand2[of "fveq ac" ac x]
    by (metis disj'(1) subst_apply_fv_subset_strand_trm2 subst_apply_fv_union sup_commute)
  hence "fv (s' · δ) ⊆ fvset (δ ' V) ∪ wfrestrictedvarsst ((S@S') ·st δ) " using * by blast
  hence "fv (s' · δ) ⊆ wfrestrictedvarsst ((S@S') ·st δ) ∪ fvset (δ ' V) "
    using ⟨x = Equality ac s s'⟩ by auto
  hence "wfst (fvset (δ ' V)) (((S@S') ·st δ)@[Equality ac (s · δ) (s' · δ)])"
    using wf_eq_append'''[OF **] by metis
  thus ?thesis using ⟨x = Equality ac s s'⟩ by auto
next
case Check thus ?thesis using wf_eq_check_append'''[OF **] Equality by simp
qed
next
case (Inequality t t') thus ?thesis using wf_ineq_append'''[OF **] by simp
qed
qed (auto dest: wf_subst_apply)

lemma wf_equality:
  assumes wf: "wfst V (S@Equality ac t t'#S') "
  and δ: "mgu t t' = Some δ"
  and disj: "bvarsst (S@Equality ac t t'#S') ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "wfst (fvset (δ ' V)) ((S@S') ·st δ) "
using assms
proof (induction S' arbitrary: V rule: List.rev_induct)
  case Nil thus ?case using wf_prefix[of V S "[Equality ac t t']"] wf_subst_apply[of V S δ] by auto
next
  case (snoc x S' V) show ?case
  proof (cases ac)
    case Assign
    hence "fv t' ⊆ V ∪ wfvarsoccsst S"
      using wf_eq_fv[of V, of S t t' "S'@[x]"] snoc by auto
    hence "fv t' ⊆ V ∪ wfrestrictedvarsst S"
      using vars_snd_rcv_strand_subset2(4)[of S] by blast
    hence "fv t' ⊆ V ∪ wfrestrictedvarsst (S@S') " by force
    moreover have disj':
      "bvarsst (S@S') ∩ (subst_domain δ ∪ range_vars δ) = {}"
      "set (bvarsstp x) ∩ (subst_domain δ ∪ range_vars δ) = {}"
      "bvarsst (S@Equality ac t t'#S') ∩ (subst_domain δ ∪ range_vars δ) = {}"
      using snoc.prem(3) by auto
    ultimately have
      "fv (t' · δ) ⊆ fvset (δ ' V) ∪ wfrestrictedvarsst ((S@S') ·st δ) "
      by (metis inf_sup_aci(5) subst_apply_fv_subset_strand_trm2)
    moreover have "fv (t · δ) = fv (t' · δ) "
      by (metis MGU_is_Unifier[OF mgu_gives_MGU[OF δ]])
    ultimately have *:
      "fv (t · δ) ∪ fv (t' · δ) ⊆ fvset (δ ' V) ∪ wfrestrictedvarsst ((S@S') ·st δ) "
      by simp
  qed

```

```

from snoc.premis(1) have "wfst V (S@Equality ac t t'#S')"
```

$$\text{using wf\_prefix[of V "S@Equality ac t t'\#S'"] by simp}$$

```

hence **: "wfst (fvset (δ ' V)) ((S@S') .st δ)" by (metis snoc.IH δ disj'(3))

from snoc.premis(1) have ***: "wfst (V ∪ wfvarsoccsst (S@Equality ac t t'#S')) [x]"
  using wf_append_exec[of V "S@Equality ac t t'\#S'" "[x]"] by simp

show ?thesis
proof (cases x)
  case (Send t)
  thus ?thesis using wf_snd_append[of "fvset (δ ' V)" "(S@S') .st δ"] ** by auto
next
  case (Receive s)
  hence "fvstp x ⊆ V ∪ wfvarsoccsst (S@Equality ac t t'#S')" using *** by auto
  hence "fvstp x ⊆ V ∪ wfrestrictedvarsst (S@Equality ac t t'#S')"
```

$$\text{using vars\_snd\_rcv\_strand\_subset2(4)[of "S@Equality ac t t'\#S'"] by blast}$$

```

  hence "fvstp x ⊆ V ∪ fv t ∪ fv t' ∪ wfrestrictedvarsst (S@S')"
```

$$\text{by (cases ac) auto}$$

```

  hence "fvstp (x .stp δ) ⊆ fvset (δ ' V) ∪ fv (t . δ) ∪ fv (t' . δ) ∪ wfrestrictedvarsst ((S@S') .st δ)"
    using subst_apply_fv_subset_strand2[of fvstp]
    by (metis (no_types) inf_sup_aci(5) subst_apply_fv_union disj'(1,2))
  hence "fvstp (x .stp δ) ⊆ fvset (δ ' V) ∪ wfrestrictedvarsst ((S@S') .st δ)"
    when "ac = Assign"
    using * that by blast
  hence "fv (s . δ) ⊆ wfrestrictedvarsst ((S@S') .st δ) ∪ (fvset (δ ' V))"
```

$$\text{when "ac = Assign"}$$

```

  using ⟨x = Receive s⟩ that by auto
  hence "wfst (fvset (δ ' V)) (((S@S') .st δ)@[Receive (s . δ)])"
```

$$\text{when "ac = Assign"}$$

```

  using wf_rcv_append'''[OF **, of "s . δ"] that by metis
  thus ?thesis using ⟨x = Receive s⟩ Assign by auto
next
  case (Equality ac' s s') show ?thesis
  proof (cases ac')
    case Assign
    hence "fv s' ⊆ V ∪ wfvarsoccsst (S@Equality ac t t'#S')"
```

$$\text{using *** Equality by auto}$$

```

    hence "fv s' ⊆ V ∪ wfrestrictedvarsst (S@Equality ac t t'#S')"
```

$$\text{using vars\_snd\_rcv\_strand\_subset2(4)[of "S@Equality ac t t'\#S'"] by blast}$$

```

    hence "fv s' ⊆ V ∪ fv t ∪ fv t' ∪ wfrestrictedvarsst (S@S')"
```

$$\text{by (cases ac) auto}$$

```

    moreover have "fv s' = fvreq ac' x" "fv (s' . δ) = fvreq ac' (x .stp δ)"
      using Equality by simp_all
    ultimately have
      "fv (s' . δ) ⊆ fvset (δ ' V) ∪ fv (t . δ) ∪ fv (t' . δ) ∪ wfrestrictedvarsst ((S@S') .st δ)"
      using subst_apply_fv_subset_strand2[of "fvreq ac'" ac' x]
      by (metis disj'(1) subst_apply_fv_subset_strand_trm2 subst_apply_fv_union sup_commute)
    hence "fv (s' . δ) ⊆ fvset (δ ' V) ∪ wfrestrictedvarsst ((S@S') .st δ)"
      using * ⟨ac = Assign⟩ by blast
    hence ****:
      "fv (s' . δ) ⊆ wfrestrictedvarsst ((S@S') .st δ) ∪ fvset (δ ' V)"
      using ⟨x = Equality ac' s s'⟩ ⟨ac = Assign⟩ by auto
    thus ?thesis
      using ⟨x = Equality ac' s s'⟩ ** **** wf_eq_append' ⟨ac = Assign⟩
      by (metis (no_types, lifting) append.assoc append_Nil2 strand_step.case(3)
          strand_subst_hom subst_apply_strand_step_def)
  next
    case Check thus ?thesis using wf_eq_check_append'''[OF **] Equality by simp
  qed
next
  case (Inequality s s') thus ?thesis using wf_ineq_append'''[OF **] by simp

```

```

qed
qed (metis snoc.premis(1) wf_eq_check_append wf_subst_apply)
qed

lemma wf_rcv_prefix_ground:
  "wfst {} ((map Receive M)@S)  $\implies$  varsst (map Receive M) = {}"
by (induct M) auto

lemma simple_wfvarsoccsst_is_fvsnd:
  assumes "simple S"
  shows "wfvarsoccsst S =  $\bigcup$  (set (map fvsnd S))"
using assms unfolding simple_def
proof (induction S)
  case (Cons x S) thus ?case by (cases x) auto
qed simp

lemma wfst_simple_induct[consumes 2, case_names Nil ConsSnd ConsRcv ConsIneq]:
  fixes S::('a,'b) strand
  assumes "wfst V S" "simple S"
  "P []"
  " $\bigwedge v S. \llbracket wf_{st} V S; simple S; P S \rrbracket \implies P (S@[Send (Var v)])$ "
  " $\bigwedge t S. \llbracket wf_{st} V S; simple S; P S; fv\ t \subseteq V \cup \bigcup (set (map fv_{snd} S)) \rrbracket \implies P (S@[Receive\ t])$ "
  " $\bigwedge X F S. \llbracket wf_{st} V S; simple S; P S \rrbracket \implies P (S@[Inequality\ X\ F])$ "
  shows "P S"
using assms
proof (induction S rule: wfst_induct')
  case (ConsSnd t S)
  hence "P S" by auto
  obtain v where "t = Var v" using simple_snd_is_var[OF _ (simple (S@[Send t]))] by auto
  thus ?case using ConsSnd.premis(3)[OF (wfst V S) _ (P S)] (simple (S@[Send t])) by auto
next
  case (ConsRcv t S) thus ?case using simple_wfvarsoccsst_is_fvsnd[of "S@[Receive t]"] by auto
qed (auto simp add: simple_def)

lemma wf_trm_stp_dom_fv_disjoint:
  " $\llbracket wf_{constr} S\ \vartheta; t \in trms_{st}\ S \rrbracket \implies subst\_domain\ \vartheta \cap fv\ t = \{\}$ "
unfolding wfconstr_def by force

lemma wf_constr_bvars_disj: "wfconstr S  $\vartheta \implies (subst\_domain\ \vartheta \cup range\_vars\ \vartheta) \cap bvars_{st}\ S = \{\}$ "
unfolding range_vars_alt_def wfconstr_def by fastforce

lemma wf_constr_bvars_disj':
  assumes "wfconstr S  $\vartheta$ " "subst_domain  $\delta \cup range\_vars\ \delta \subseteq fv_{st}\ S$ "
  shows "(subst_domain  $\delta \cup range\_vars\ \delta) \cap bvars_{st}\ S = \{\}$ " (is ?A)
  and "(subst_domain  $\vartheta \cup range\_vars\ \vartheta) \cap bvars_{st}\ (S \cdot_{st} \delta) = \{\}$ " (is ?B)
proof -
  have "(subst_domain  $\vartheta \cup range\_vars\ \vartheta) \cap bvars_{st}\ S = \{\}$ " "fvst S  $\cap bvars_{st}\ S = \{\}$ "
  using assms(1) unfolding range_vars_alt_def wfconstr_def by fastforce+
  thus ?A and ?B using assms(2) bvars_subst_ident[of S  $\delta$ ] by blast+
qed

lemma (in intruder_model) wf_simple_strand_first_Send_var_split:
  assumes "wfst {} S" "simple S" " $\exists v \in wfrestrictedvars_{st}\ S. t \cdot \mathcal{I} = \mathcal{I}\ v$ "
  shows " $\exists v\ S_{pre}\ S_{suf}. S = S_{pre}@Send (Var v)\#S_{suf} \wedge t \cdot \mathcal{I} = \mathcal{I}\ v$ "
  " $\wedge \neg(\exists w \in wfrestrictedvars_{st}\ S_{pre}. t \cdot \mathcal{I} = \mathcal{I}\ w)$ "
  (is "?P S")
using assms
proof (induction S rule: wfst_simple_induct)
  case (ConsSnd v S) show ?case
  proof (cases " $\exists w \in wfrestrictedvars_{st}\ S. t \cdot \mathcal{I} = \mathcal{I}\ w$ ")
    case True thus ?thesis using ConsSnd.IH by fastforce
  next
    case False thus ?thesis using ConsSnd.premis by auto
  end
end

```

```

qed
next
  case (ConsRcv t' S)
  have "fv t'  $\subseteq$  wfrestrictedvarsst S" using ConsRcv.hyps(3) vars_snd_rcv_strand_subset2(1) by force
  hence " $\exists v \in \text{wfrestrictedvars}_{st} S. t \cdot \mathcal{I} = \mathcal{I} v$ "
    using ConsRcv.prem(1) by fastforce
  hence "?P S" by (metis ConsRcv.IH)
  thus ?case by fastforce
next
  case (ConsIneq X F S)
  moreover have "wfrestrictedvarsst (S @ [Inequality X F]) = wfrestrictedvarsst S" by auto
  ultimately have "?P S" by blast
  thus ?case by fastforce
qed simp

lemma (in intruder_model) wf_strand_first_Send_var_split:
  assumes "wfst {} S" " $\exists v \in \text{wfrestrictedvars}_{st} S. t \cdot \mathcal{I} \sqsubseteq \mathcal{I} v$ "
  shows " $\exists S_{pre} S_{suf}. \neg(\exists w \in \text{wfrestrictedvars}_{st} S_{pre}. t \cdot \mathcal{I} \sqsubseteq \mathcal{I} w)$ "
     $\wedge ((\exists t'. S = S_{pre} @ \text{Send } t' \# S_{suf} \wedge t \cdot \mathcal{I} \sqsubseteq t' \cdot \mathcal{I})$ 
       $\vee (\exists t' t''. S = S_{pre} @ \text{Equality Assign } t' t'' \# S_{suf} \wedge t \cdot \mathcal{I} \sqsubseteq t' \cdot \mathcal{I}))$ "
    (is " $\exists S_{pre} S_{suf}. ?P S_{pre} \wedge ?Q S_{pre} S_{suf}$ ")
using assms
proof (induction S rule: wfst-induct')
  case (ConsSnd t' S) show ?case
  proof (cases " $\exists w \in \text{wfrestrictedvars}_{st} S. t \cdot \mathcal{I} \sqsubseteq \mathcal{I} w$ ")
    case True
    then obtain Spre Ssuf where "?P Spre" "?Q Spre Ssuf"
      using ConsSnd.IH by maura
    thus ?thesis by fastforce
  next
    case False
    then obtain v where v: "v  $\in$  fv t'" "t  $\cdot \mathcal{I} \sqsubseteq \mathcal{I} v$ "
      using ConsSnd.prem by auto
    hence "t  $\cdot \mathcal{I} \sqsubseteq t' \cdot \mathcal{I}$ "
      using subst_mono[of "Var v" t'  $\mathcal{I}$ ] vars_iff_subtermeq[of v t'] term.order_trans
      by auto
    thus ?thesis using False v by auto
  qed
next
  case (ConsRcv t' S)
  have "fv t'  $\subseteq$  wfrestrictedvarsst S"
    using ConsRcv.hyps vars_snd_rcv_strand_subset2(4)[of S] by blast
  hence " $\exists v \in \text{wfrestrictedvars}_{st} S. t \cdot \mathcal{I} \sqsubseteq \mathcal{I} v$ "
    using ConsRcv.prem by fastforce
  then obtain Spre Ssuf where "?P Spre" "?Q Spre Ssuf"
    using ConsRcv.IH by maura
  thus ?case by fastforce
next
  case (ConsEq s s' S)
  have *: "fv s'  $\subseteq$  wfrestrictedvarsst S"
    using ConsEq.hyps vars_snd_rcv_strand_subset2(4)[of S]
    by blast
  show ?case
  proof (cases " $\exists v \in \text{wfrestrictedvars}_{st} S. t \cdot \mathcal{I} \sqsubseteq \mathcal{I} v$ ")
    case True
    then obtain Spre Ssuf where "?P Spre" "?Q Spre Ssuf"
      using ConsEq.IH by maura
    thus ?thesis by fastforce
  next
    case False
    then obtain v where "v  $\in$  fv s" "t  $\cdot \mathcal{I} \sqsubseteq \mathcal{I} v$ " using ConsEq.prem * by auto
    hence "t  $\cdot \mathcal{I} \sqsubseteq s \cdot \mathcal{I}$ "
      using vars_iff_subtermeq[of v s] subst_mono[of "Var v" s  $\mathcal{I}$ ] term.order_trans

```

```

    by auto
  thus ?thesis using False by fastforce
qed
next
case (ConsEq2 s s' S)
have "wfrestrictedvarsst (S@[Equality Check s s']) = wfrestrictedvarsst S" by auto
hence "∃ v ∈ wfrestrictedvarsst S. t · I ⊆ I v" using ConsEq2.prem by metis
then obtain Spre Ssuf where "?P Spre" "?Q S Spre Ssuf"
  using ConsEq2.IH by moura
thus ?case by fastforce
next
case (ConsIneq X F S)
hence "∃ v ∈ wfrestrictedvarsst S. t · I ⊆ I v" by fastforce
then obtain Spre Ssuf where "?P Spre" "?Q S Spre Ssuf"
  using ConsIneq.IH by moura
thus ?case by fastforce
qed simp

```

### 3.1.6 Constraint Semantics

```

context intruder_model
begin

```

#### Definitions

The constraint semantics in which the intruder is limited to composition only

```

fun strand_sem_c::('fun,'var) terms ⇒ ('fun,'var) strand ⇒ ('fun,'var) subst ⇒ bool ("⊢c" " ")
where
  "[M; []]c = (λI. True)"
  | "[M; Send t#S]c = (λI. M ⊢c t · I ∧ [M; S]c I)"
  | "[M; Receive t#S]c = (λI. [insert (t · I) M; S]c I)"
  | "[M; Equality _ t t'#S]c = (λI. t · I = t' · I ∧ [M; S]c I)"
  | "[M; Inequality X F#S]c = (λI. ineq_model I X F ∧ [M; S]c I)"

```

```

definition constr_sem_c ("_ ⊢c ⟨_,_⟩") where "I ⊢c ⟨S, ϑ⟩ ≡ (ϑ supports I ∧ [{}; S]c I)"
abbreviation constr_sem_c' ("_ ⊢c ⟨_⟩" 90) where "I ⊢c ⟨S⟩ ≡ I ⊢c ⟨S, Var⟩"

```

The full constraint semantics

```

fun strand_sem_d::('fun,'var) terms ⇒ ('fun,'var) strand ⇒ ('fun,'var) subst ⇒ bool ("⊢d" " ")
where
  "[M; []]d = (λI. True)"
  | "[M; Send t#S]d = (λI. M ⊢ t · I ∧ [M; S]d I)"
  | "[M; Receive t#S]d = (λI. [insert (t · I) M; S]d I)"
  | "[M; Equality _ t t'#S]d = (λI. t · I = t' · I ∧ [M; S]d I)"
  | "[M; Inequality X F#S]d = (λI. ineq_model I X F ∧ [M; S]d I)"

```

```

definition constr_sem_d ("_ ⊢ ⟨_,_⟩") where "I ⊢ ⟨S, ϑ⟩ ≡ (ϑ supports I ∧ [{}; S]d I)"
abbreviation constr_sem_d' ("_ ⊢ ⟨_⟩" 90) where "I ⊢ ⟨S⟩ ≡ I ⊢ ⟨S, Var⟩"

```

```

lemmas strand_sem_induct = strand_sem_c.induct[case_names Nil ConsSnd ConsRcv ConsEq ConsIneq]

```

#### Lemmata

```

lemma strand_sem_d_if_c: "I ⊢c ⟨S, ϑ⟩ ⇒ I ⊢ ⟨S, ϑ⟩"
proof -
  assume *: "I ⊢c ⟨S, ϑ⟩"
  { fix M have "[M; S]c I ⇒ [M; S]d I"
    proof (induction S rule: strand_sem_induct)
      case (ConsSnd M t S)
      hence "M ⊢c t · I" "[M; S]d I" by auto
      thus ?case using strand_sem_d.simps(2)[of M t S] by auto
    qed (auto simp add: ineq_model_def)
  }

```

```

thus ?thesis using * by (simp add: constr_sem_c_def constr_sem_d_def)
qed

lemma strand_sem_mono_ik:
  "[M ⊆ M'; [M; S]_c ∅] ⇒ [M'; S]_c ∅" (is "[?A'; ?A''] ⇒ ?A")
  "[M ⊆ M'; [M; S]_d ∅] ⇒ [M'; S]_d ∅" (is "[?B'; ?B''] ⇒ ?B")
proof -
  show "[?A'; ?A''] ⇒ ?A"
  proof (induction M S arbitrary: M M' rule: strand_sem_induct)
    case (ConsRcv M t S)
    thus ?case using ConsRcv.IH[of "insert (t · ∅) M" "insert (t · ∅) M'"] by auto
  next
    case (ConsSnd M t S)
    hence "M ⊢_c t · ∅" "[M'; S]_c ∅" by auto
    hence "M' ⊢_c t · ∅" using ideduct_synth_mono ⟨M ⊆ M'⟩ by metis
    thus ?case using ⟨[M'; S]_c ∅⟩ by simp
  qed auto

  show "[?B'; ?B''] ⇒ ?B"
  proof (induction M S arbitrary: M M' rule: strand_sem_induct)
    case (ConsRcv M t S)
    thus ?case using ConsRcv.IH[of "insert (t · ∅) M" "insert (t · ∅) M'"] by auto
  next
    case (ConsSnd M t S)
    hence "M ⊢_d t · ∅" "[M'; S]_d ∅" by auto
    hence "M' ⊢_d t · ∅" using ideduct_mono ⟨M ⊆ M'⟩ by metis
    thus ?case using ⟨[M'; S]_d ∅⟩ by simp
  qed auto
qed

context
begin
private lemma strand_sem_split_left:
  "[M; S@S']_c ∅ ⇒ [M; S]_c ∅"
  "[M; S@S']_d ∅ ⇒ [M; S]_d ∅"
proof (induct S arbitrary: M)
  case (Cons x S)
  { case 1 thus ?case using Cons by (cases x) simp_all }
  { case 2 thus ?case using Cons by (cases x) simp_all }
qed simp_all

private lemma strand_sem_split_right:
  "[M; S@S']_c ∅ ⇒ [M ∪ (ikst S ·set ∅); S']_c ∅"
  "[M; S@S']_d ∅ ⇒ [M ∪ (ikst S ·set ∅); S']_d ∅"
proof (induction S arbitrary: M rule: ikst-induct)
  case (ConsRcv t S)
  { case 1 thus ?case using ConsRcv.IH[of "insert (t · ∅) M"] by simp }
  { case 2 thus ?case using ConsRcv.IH[of "insert (t · ∅) M"] by simp }
qed simp_all

lemmas strand_sem_split[dest] =
  strand_sem_split_left(1) strand_sem_split_right(1)
  strand_sem_split_left(2) strand_sem_split_right(2)
end

lemma strand_sem_Send_split[dest]:
  "[[M; map Send T]_c ∅; t ∈ set T] ⇒ [M; [Send t]_c ∅] (is "[?A'; ?A''] ⇒ ?A")
  "[[M; map Send T]_d ∅; t ∈ set T] ⇒ [M; [Send t]_d ∅] (is "[?B'; ?B''] ⇒ ?B")
  "[[M; map Send T@S]_c ∅; t ∈ set T] ⇒ [M; Send t#S]_c ∅ (is "[?C'; ?C''] ⇒ ?C")
  "[[M; map Send T@S]_d ∅; t ∈ set T] ⇒ [M; Send t#S]_d ∅ (is "[?D'; ?D''] ⇒ ?D")
proof -
  show A: "[?A'; ?A''] ⇒ ?A" by (induct "map Send T" arbitrary: T rule: strand_sem_c.induct) auto
  show B: "[?B'; ?B''] ⇒ ?B" by (induct "map Send T" arbitrary: T rule: strand_sem_d.induct) auto

```



```

show "[[?C'; ?C'']] ==> ?C" "[[?D'; ?D'']] ==> ?D"
  using list.set_map list.simps(8) set_empty ik_snd_empty sup_bot.right_neutral
  by (metis (no_types, lifting) A strand_sem_split(1,2) strand_sem_c.simps(2),
      metis (no_types, lifting) B strand_sem_split(3,4) strand_sem_d.simps(2))
qed

lemma strand_sem_Send_map:
  "(( $\bigwedge t. t \in \text{set } T \implies \llbracket M; [\text{Send } t] \rrbracket_c \mathcal{I}$ )  $\implies \llbracket M; \text{map Send } T \rrbracket_c \mathcal{I}$ "
  "(( $\bigwedge t. t \in \text{set } T \implies \llbracket M; [\text{Send } t] \rrbracket_d \mathcal{I}$ )  $\implies \llbracket M; \text{map Send } T \rrbracket_d \mathcal{I}$ "
  by (induct T) auto

lemma strand_sem_Receive_map: " $\llbracket M; \text{map Receive } T \rrbracket_c \mathcal{I}$ " " $\llbracket M; \text{map Receive } T \rrbracket_d \mathcal{I}$ "
  by (induct T arbitrary: M) auto

lemma strand_sem_append[intro]:
  " $\llbracket \llbracket M; S \rrbracket_c \vartheta; \llbracket M \cup (\text{ik}_{st} S \cdot_{\text{set}} \vartheta); S' \rrbracket_c \vartheta \rrbracket \implies \llbracket M; S@S' \rrbracket_c \vartheta$ "
  " $\llbracket \llbracket M; S \rrbracket_d \vartheta; \llbracket M \cup (\text{ik}_{st} S \cdot_{\text{set}} \vartheta); S' \rrbracket_d \vartheta \rrbracket \implies \llbracket M; S@S' \rrbracket_d \vartheta$ "
proof (induction S arbitrary: M)
  case (Cons x S)
  { case 1 thus ?case using Cons by (cases x) auto }
  { case 2 thus ?case using Cons by (cases x) auto }
qed simp_all

lemma ineq_model_subst:
  fixes F::"('a,'b) term  $\times$  ('a,'b) term) list"
  assumes "(subst_domain  $\delta \cup \text{range\_vars } \delta) \cap \text{set } X = \{\}$ "
  and "ineq_model ( $\delta \circ_s \vartheta$ ) X F"
  shows "ineq_model  $\vartheta$  X (F  $\cdot_{\text{pairs}}$   $\delta$ )"
proof -
  { fix  $\sigma$ ::"('a,'b) subst" and t t'
    assume  $\sigma$ : "subst_domain  $\sigma = \text{set } X$ " "ground (subst_range  $\sigma$ )"
    and *: "list_ex ( $\lambda f. \text{fst } f \cdot (\sigma \circ_s (\delta \circ_s \vartheta)) \neq \text{snd } f \cdot (\sigma \circ_s (\delta \circ_s \vartheta))$ ) F"
    obtain f where f: "f  $\in \text{set } F$ " "fst f  $\cdot \sigma \circ_s (\delta \circ_s \vartheta) \neq \text{snd } f \cdot \sigma \circ_s (\delta \circ_s \vartheta)$ "
    using * by (induct F) auto
    have " $\sigma \circ_s (\delta \circ_s \vartheta) = \delta \circ_s (\sigma \circ_s \vartheta)$ "
    by (metis (no_types, lifting)  $\sigma$  subst_compose_assoc assms(1) inf_sup_aci(1)
        subst_comp_eq_if_disjoint_vars sup_inf_absorb_range_vars_alt_def)
    hence "(fst f  $\cdot \delta$ )  $\cdot \sigma \circ_s \vartheta \neq (\text{snd } f \cdot \delta) \cdot \sigma \circ_s \vartheta$ " using f by auto
    moreover have "(fst f  $\cdot \delta, \text{snd } f \cdot \delta) \in \text{set } (F \cdot_{\text{pairs}} \delta)$ "
    using f(1) by (auto simp add: subst_apply_pairs_def)
    ultimately have "list_ex ( $\lambda f. \text{fst } f \cdot (\sigma \circ_s \vartheta) \neq \text{snd } f \cdot (\sigma \circ_s \vartheta)$ ) (F  $\cdot_{\text{pairs}} \delta$ )"
    using f(1) Bex_set by fastforce
  }
  thus ?thesis using assms unfolding ineq_model_def by simp
qed

lemma ineq_model_subst':
  fixes F::"('a,'b) term  $\times$  ('a,'b) term) list"
  assumes "(subst_domain  $\delta \cup \text{range\_vars } \delta) \cap \text{set } X = \{\}$ "
  and "ineq_model  $\vartheta$  X (F  $\cdot_{\text{pairs}}$   $\delta$ )"
  shows "ineq_model ( $\delta \circ_s \vartheta$ ) X F"
proof -
  { fix  $\sigma$ ::"('a,'b) subst" and t t'
    assume  $\sigma$ : "subst_domain  $\sigma = \text{set } X$ " "ground (subst_range  $\sigma$ )"
    and *: "list_ex ( $\lambda f. \text{fst } f \cdot (\sigma \circ_s \vartheta) \neq \text{snd } f \cdot (\sigma \circ_s \vartheta)$ ) (F  $\cdot_{\text{pairs}} \delta$ )"
    obtain f where f: "f  $\in \text{set } (F \cdot_{\text{pairs}} \delta)$ " "fst f  $\cdot \sigma \circ_s \vartheta \neq \text{snd } f \cdot \sigma \circ_s \vartheta$ "
    using * by (induct F) (auto simp add: subst_apply_pairs_def)
    then obtain g where g: "g  $\in \text{set } F$ " "f = g  $\cdot_p \delta$ " by (auto simp add: subst_apply_pairs_def)
    have " $\sigma \circ_s (\delta \circ_s \vartheta) = \delta \circ_s (\sigma \circ_s \vartheta)$ "
    by (metis (no_types, lifting)  $\sigma$  subst_compose_assoc assms(1) inf_sup_aci(1)
        subst_comp_eq_if_disjoint_vars sup_inf_absorb_range_vars_alt_def)
    hence "fst g  $\cdot \sigma \circ_s (\delta \circ_s \vartheta) \neq \text{snd } g \cdot \sigma \circ_s (\delta \circ_s \vartheta)$ "
    using f(2) g by (simp add: prod.case_eq_if)
  }

```

```

    hence "list_ex (λf. fst f · (σ ∘s (δ ∘s ϑ))) ≠ snd f · (σ ∘s (δ ∘s ϑ))) F"
      using g Bex_set by fastforce
  }
  thus ?thesis using assms unfolding ineq_model_def by simp
qed

lemma ineq_model_ground_subst:
  fixes F::"('a,'b) term × ('a,'b) term) list"
  assumes "fv_pairs F - set X ⊆ subst_domain δ"
    and "ground (subst_range δ)"
    and "ineq_model δ X F"
  shows "ineq_model (δ ∘s ϑ) X F"
proof -
  { fix σ::"('a,'b) subst" and t t'
    assume σ: "subst_domain σ = set X" "ground (subst_range σ)"
      and *: "list_ex (λf. fst f · (σ ∘s δ)) ≠ snd f · (σ ∘s δ)) F"
    obtain f where f: "f ∈ set F" "fst f · σ ∘s δ ≠ snd f · σ ∘s δ"
      using * by (induct F) auto
    hence "fv (fst f) ⊆ fv_pairs F" "fv (snd f) ⊆ fv_pairs F" by auto
    hence "fv (fst f) - set X ⊆ subst_domain δ" "fv (snd f) - set X ⊆ subst_domain δ"
      using assms(1) by auto
    hence "fv (fst f · σ) ⊆ subst_domain δ" "fv (snd f · σ) ⊆ subst_domain δ"
      using σ by (simp_all add: range_vars_alt_def subst_fv_unfold_ground_img)
    hence "fv (fst f · σ ∘s δ) = {}" "fv (snd f · σ ∘s δ) = {}"
      using assms(2) by (simp_all add: subst_fv_dom_ground_if_ground_img)
    hence "fst f · σ ∘s (δ ∘s ϑ) ≠ snd f · σ ∘s (δ ∘s ϑ)" using f(2) subst_ground_ident by fastforce

    hence "list_ex (λf. fst f · (σ ∘s (δ ∘s ϑ))) ≠ snd f · (σ ∘s (δ ∘s ϑ))) F"
      using f(1) Bex_set by fastforce
  }
  thus ?thesis using assms unfolding ineq_model_def by simp
qed

context
begin
private lemma strand_sem_subst_c:
  assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
  shows "[M; S]c (δ ∘s ϑ) ⇒ [M; S ·st δ]c ϑ"
using assms
proof (induction S arbitrary: δ M rule: strand_sem_induct)
  case (ConsSnd M t S)
  hence "[M; S ·st δ]c ϑ" "M ⊢c t · (δ ∘s ϑ)" by auto
  hence "M ⊢c (t · δ) · ϑ"
    using subst_comp_all[of δ ϑ M] subst_subst_compose[of t δ ϑ] by simp
  thus ?case
    using ⟨[M; S ·st δ]c ϑ⟩
    unfolding subst_apply_strand_def
    by simp
next
  case (ConsRcv M t S)
  have *: "[insert (t · δ ∘s ϑ) M; S]c (δ ∘s ϑ)" using ConsRcv.prem1 by simp
  have "bvarsst (Receive t#S) = bvarsst S" by auto
  hence **: "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}" using ConsRcv.prem2 by blast
  have "[M; Receive (t · δ)#(S ·st δ)]c ϑ"
    using ConsRcv.IH[OF *] by (simp add: subst_all_insert)
  thus ?case by simp
next
  case (ConsIneq M X F S)
  hence *: "[M; S ·st δ]c ϑ" and
    ***: "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
    unfolding bvarsst_def ineq_model_def by auto
  have **: "ineq_model (δ ∘s ϑ) X F"
    using ConsIneq by (auto simp add: subst_compose_assoc ineq_model_def)

```

```

have "∀γ. subst_domain γ = set X ∧ ground (subst_range γ)
  → (subst_domain δ ∪ range_vars δ) ∩ (subst_domain γ ∪ range_vars γ) = {}"
  using * ** *** unfolding range_vars_alt_def by auto
hence "∀γ. subst_domain γ = set X ∧ ground (subst_range γ) → γ ∘s δ = δ ∘s γ"
  by (metis subst_comp_eq_if_disjoint_vars)
hence "ineq_model ∅ X (F ·pairs δ)"
  using ineq_model_subst[OF *** **]
  by blast
moreover have "rm_vars (set X) δ = δ" using ConsIneq.prem(2) by force
ultimately show ?case using * by auto
qed simp_all

```

```

private lemma strand_sem_subst_c':
  assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
  shows "[M; S ·st δ]c ∅ ⇒ [M; S]c (δ ∘s ∅)"
using assms
proof (induction S arbitrary: δ M rule: strand_sem_induct)
  case (ConsSnd M t S)
  hence "[M; [Send t] ·st δ]c ∅" "[M; S ·st δ]c ∅" by auto
  hence "[M; S]c (δ ∘s ∅)" using ConsSnd.IH[OF _] ConsSnd.prem(2) by auto
  moreover have "[M; [Send t]]c (δ ∘s ∅)"
  proof -
    have "M ⊢c t · δ · ∅" using ([M; [Send t] ·st δ]c ∅) by auto
    hence "M ⊢c t · (δ ∘s ∅)" using subst_subst_compose by metis
    thus "[M; [Send t]]c (δ ∘s ∅)" by auto
  qed
  ultimately show ?case by auto
next

```

```

case (ConsRcv M t S)
hence "[ (insert (t · δ · ∅) M); S ·st δ ]c ∅" by (simp add: subst_all_insert)
thus ?case using ConsRcv.IH ConsRcv.prem(2) by auto
next

```

```

case (ConsIneq M X F S)
have δ: "rm_vars (set X) δ = δ" using ConsIneq.prem(2) by force
hence *: "[M; S]c (δ ∘s ∅)"
  and ***: "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
  using ConsIneq unfolding bvarsst_def ineq_model_def by auto
have **: "ineq_model ∅ X (F ·pairs δ)"
  using ConsIneq.prem(1) δ by (auto simp add: subst_compose_assoc ineq_model_def)
have "∀γ. subst_domain γ = set X ∧ ground (subst_range γ)
  → (subst_domain δ ∪ range_vars δ) ∩ (subst_domain γ ∪ range_vars γ) = {}"
  using * ** *** unfolding range_vars_alt_def by auto
hence "∀γ. subst_domain γ = set X ∧ ground (subst_range γ) → γ ∘s δ = δ ∘s γ"
  by (metis subst_comp_eq_if_disjoint_vars)
hence "ineq_model (δ ∘s ∅) X F"
  using ineq_model_subst'[OF *** **]
  by blast
thus ?case using * by auto
next

```

```

case ConsEq thus ?case unfolding bvarsst_def by auto
qed simp_all

```

```

private lemma strand_sem_subst_d:
  assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
  shows "[M; S]d (δ ∘s ∅) ⇒ [M; S ·st δ]d ∅"
using assms
proof (induction S arbitrary: δ M rule: strand_sem_induct)
  case (ConsSnd M t S)
  hence "[M; S ·st δ]d ∅" "M ⊢ t · (δ ∘s ∅)" by auto
  hence "M ⊢ (t · δ) · ∅"
    using subst_comp_all[of δ ∅ M] subst_subst_compose[of t δ ∅] by simp
  thus ?case using ([M; S ·st δ]d ∅) by simp
next

```

```

case (ConsRcv M t S)
have *: "[[insert (t · δ ∘s ∅) M; S]]d (δ ∘s ∅)" using ConsRcv.prem(1) by simp
have "bvarsst (Receive t#S) = bvarsst S" by auto
hence **: "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}" using ConsRcv.prem(2) by blast
have "[[M; Receive (t · δ)#(S ·st δ)]]d ∅"
  using ConsRcv.IH[OF * **] by (simp add: subst_all_insert)
thus ?case by simp
next
case (ConsIneq M X F S)
hence *: "[[M; S ·st δ]]d ∅" and
  ***: "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
  unfolding bvarsst_def ineq_model_def by auto
have **: "ineq_model (δ ∘s ∅) X F"
  using ConsIneq by (auto simp add: subst_compose_assoc ineq_model_def)
have "∀γ. subst_domain γ = set X ∧ ground (subst_range γ)
  → (subst_domain δ ∪ range_vars δ) ∩ (subst_domain γ ∪ range_vars γ) = {}"
  using * ** *** unfolding range_vars_alt_def by auto
hence "∀γ. subst_domain γ = set X ∧ ground (subst_range γ) → γ ∘s δ = δ ∘s γ"
  by (metis subst_comp_eq_if_disjoint_vars)
hence "ineq_model ∅ X (F ·pairs δ)"
  using ineq_model_subst[OF *** **]
  by blast
moreover have "rm_vars (set X) δ = δ" using ConsIneq.prem(2) by force
ultimately show ?case using * by auto
next
case ConsEq thus ?case unfolding bvarsst_def by auto
qed simp_all

private lemma strand_sem_subst_d':
  assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
  shows "[[M; S ·st δ]]d ∅ ⇒ [[M; S]]d (δ ∘s ∅)"
using assms
proof (induction S arbitrary: δ M rule: strand_sem_induct)
case (ConsSnd M t S)
hence "[[M; [Send t] ·st δ]]d ∅" "[[M; S ·st δ]]d ∅" by auto
hence "[[M; S]]d (δ ∘s ∅)" using ConsSnd.IH[OF _] ConsSnd.prem(2) by auto
moreover have "[[M; [Send t]]]d (δ ∘s ∅)"
proof -
  have "M ⊢ t · δ · ∅" using ⟨[[M; [Send t] ·st δ]]d ∅⟩ by auto
  hence "M ⊢ t · (δ ∘s ∅)" using subst_subst_compose by metis
  thus "[[M; [Send t]]]d (δ ∘s ∅)" by auto
qed
ultimately show ?case by auto
next
case (ConsRcv M t S)
hence "[[insert (t · δ · ∅) M; S ·st δ]]d ∅" by (simp add: subst_all_insert)
thus ?case using ConsRcv.IH ConsRcv.prem(2) by auto
next
case (ConsIneq M X F S)
have δ: "rm_vars (set X) δ = δ" using ConsIneq.prem(2) by force
hence *: "[[M; S]]d (δ ∘s ∅)"
  and ***: "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
  using ConsIneq unfolding bvarsst_def ineq_model_def by auto
have **: "ineq_model ∅ X (F ·pairs δ)"
  using ConsIneq.prem(1) δ by (auto simp add: subst_compose_assoc ineq_model_def)
have "∀γ. subst_domain γ = set X ∧ ground (subst_range γ)
  → (subst_domain δ ∪ range_vars δ) ∩ (subst_domain γ ∪ range_vars γ) = {}"
  using * ** *** unfolding range_vars_alt_def by auto
hence "∀γ. subst_domain γ = set X ∧ ground (subst_range γ) → γ ∘s δ = δ ∘s γ"
  by (metis subst_comp_eq_if_disjoint_vars)
hence "ineq_model (δ ∘s ∅) X F"
  using ineq_model_subst'[OF *** **]
  by blast

```

```

  thus ?case using * by auto
next
  case ConsEq thus ?case unfolding bvarsst_def by auto
qed simp_all

lemmas strand_sem_subst =
  strand_sem_subst_c strand_sem_subst_c' strand_sem_subst_d strand_sem_subst_d'
end

lemma strand_sem_subst_subst_idem:
  assumes  $\delta: "(subst\_domain\ \delta \cup range\_vars\ \delta) \cap bvars_{st}\ S = \{\}"$ 
  shows " $\llbracket M; S \cdot_{st} \delta \rrbracket_c (\delta \circ_s \vartheta); subst\_idem\ \delta \rrbracket \implies \llbracket M; S \rrbracket_c (\delta \circ_s \vartheta)$ "
using strand_sem_subst(2)[OF assms, of M " $\delta \circ_s \vartheta$ "] subst_compose_assoc[of  $\delta\ \delta\ \vartheta$ ]
unfolding subst_idem_def by argo

lemma strand_sem_subst_comp:
  assumes " $(subst\_domain\ \vartheta \cup range\_vars\ \vartheta) \cap bvars_{st}\ S = \{\}"$ 
  and " $\llbracket M; S \rrbracket_c\ \delta$ " " $subst\_domain\ \vartheta \cap (vars_{st}\ S \cup fv_{set}\ M) = \{\}"$ 
  shows " $\llbracket M; S \rrbracket_c (\vartheta \circ_s \delta)$ "
proof -
  from assms(3) have " $subst\_domain\ \vartheta \cap vars_{st}\ S = \{\}"$ " " $subst\_domain\ \vartheta \cap fv_{set}\ M = \{\}"$  by auto
  hence " $S \cdot_{st} \vartheta = S$ " " $M \cdot_{set} \vartheta = M$ " using strand_substI set_subst_ident[of M  $\vartheta$ ] by (blast, blast)
  thus ?thesis using assms(2) by (auto simp add: strand_sem_subst(2)[OF assms(1)])
qed

lemma strand_sem_c_imp_ineqs_neq:
  assumes " $\llbracket M; S \rrbracket_c\ \mathcal{I}$ " "Inequality X  $[(t, t')]$   $\in$  set S"
  shows " $t \neq t' \wedge (\forall \delta. subst\_domain\ \delta = set\ X \wedge ground\ (subst\_range\ \delta) \longrightarrow t \cdot \delta \neq t' \cdot \delta \wedge t \cdot \delta \cdot \mathcal{I} \neq t' \cdot \delta \cdot \mathcal{I})$ "
using assms
proof (induction rule: strand_sem_induct)
  case (ConsIneq M Y F S) thus ?case
  proof (cases "Inequality X  $[(t, t')]$   $\in$  set S")
    case False
    hence "X = Y" "F =  $[(t, t')]$ " using ConsIneq by auto
    hence *: " $\forall \vartheta. subst\_domain\ \vartheta = set\ X \wedge ground\ (subst\_range\ \vartheta) \longrightarrow t \cdot \vartheta \cdot \mathcal{I} \neq t' \cdot \vartheta \cdot \mathcal{I}$ "
    using ConsIneq by (auto simp add: ineq_model_def)
    then obtain  $\vartheta$  where  $\vartheta$ : " $subst\_domain\ \vartheta = set\ X$ " " $ground\ (subst\_range\ \vartheta)$ " " $t \cdot \vartheta \cdot \mathcal{I} \neq t' \cdot \vartheta \cdot \mathcal{I}$ "
   $\mathcal{I}$ "
    using interpretation_subst_exists'[of "set X"] by maura
    hence " $t \neq t'$ " by auto
    moreover have " $\bigwedge \mathcal{I}\ \vartheta. t \cdot \vartheta \cdot \mathcal{I} \neq t' \cdot \vartheta \cdot \mathcal{I} \implies t \cdot \vartheta \neq t' \cdot \vartheta$ " by auto
    ultimately show ?thesis using * by auto
  qed simp
qed simp_all

lemma strand_sem_c_imp_ineq_model:
  assumes " $\llbracket M; S \rrbracket_c\ \mathcal{I}$ " "Inequality X F  $\in$  set S"
  shows "ineq_model  $\mathcal{I}$  X F"
using assms by (induct S rule: strand_sem_induct) force+

lemma strand_sem_wf_simple_fv_sat:
  assumes " $wf_{st}\ \{\}$  S" "simple S" " $\llbracket \{\}; S \rrbracket_c\ \mathcal{I}$ "
  shows " $\bigwedge v. v \in wf_{restrictedvars}_{st}\ S \implies ik_{st}\ S \cdot_{set}\ \mathcal{I} \vdash_c\ \mathcal{I}\ v$ "
using assms
proof (induction S rule: wfst_simple_induct)
  case (ConsRcv t S)
  have " $v \in wf_{restrictedvars}_{st}\ S$ "
  using ConsRcv.hyps(3) ConsRcv.premis(1) vars_snd_rcv_strand2
  by fastforce
  moreover have " $\llbracket \{\}; S \rrbracket_c\ \mathcal{I}$ " using  $\langle \{\}; S@[Receive\ t] \rrbracket_c\ \mathcal{I}$  by blast
  moreover have " $ik_{st}\ S \cdot_{set}\ \mathcal{I} \subseteq ik_{st}\ (S@[Receive\ t]) \cdot_{set}\ \mathcal{I}$ " by auto
  ultimately show ?case using ConsRcv.IH ideduct_synth_mono by meson

```

```

next
  case (ConsIneq X F S)
  hence "v ∈ wfrestrictedvarsst S" by fastforce
  moreover have "[{}; S]c I" using ⟨[{}; S@Inequality X F]c I⟩ by blast
  moreover have "ikst S ·set I ⊆ ikst (S@Inequality X F) ·set I" by auto
  ultimately show ?case using ConsIneq.IH ideduct_synth_mono by meson
next
  case (ConsSnd w S)
  hence *: "[{}; S]c I" "ikst S ·set I ⊢c I w" by auto
  have **: "ikst S ·set I ⊆ ikst (S@Send (Var w)) ·set I" by simp
  show ?case
  proof (cases "v = w")
    case True thus ?thesis using *(2) ideduct_synth_mono[OF _ **] by meson
  next
    case False
    hence "v ∈ wfrestrictedvarsst S" using ConsSnd.prem(1) by auto
    thus ?thesis using ConsSnd.IH[OF _ *(1)] ideduct_synth_mono[OF _ **] by metis
  qed
qed simp

lemma strand_sem_wf_ik_or_assignment_rhs_fun_subterm:
  assumes "wfst {} A" "[{}; A]c I" "Var x ∈ ikst A" "I x = Fun f T"
    "ti ∈ set T" "¬ikst A ·set I ⊢c ti" "interpretationsubst I"
  obtains S where
    "Fun f S ∈ subtermsset (ikst A) ∨ Fun f S ∈ subtermsset (assignment_rhsst A)"
    "Fun f T = Fun f S · I"
  proof -
    have "x ∈ wfrestrictedvarsst A"
      by (metis (no_types) assms(3) set_rev_mp term.set_intros(3) vars_subset_if_in_strand_ik2)
    moreover have "Fun f T · I = Fun f T"
      by (metis subst_ground_ident interpretation_grounds_all assms(4,7))
    ultimately obtain Apre Asuf where *:
      "¬(∃ w ∈ wfrestrictedvarsst Apre. Fun f T ⊆ I w)"
      "(∃ t. A = Apre@Send t#Asuf ∧ Fun f T ⊆ t · I) ∨
      (∃ t t'. A = Apre@Equality Assign t t'#Asuf ∧ Fun f T ⊆ t · I)"
    using wf_strand_first_Send_var_split[OF assms(1)] assms(4) subtermeqI' by metis
    moreover
    { fix t assume **: "A = Apre@Send t#Asuf" "Fun f T ⊆ t · I"
      hence "ikst Apre ·set I ⊢c t · I" "¬ikst Apre ·set I ⊢c ti"
        using assms(2,6) by (auto intro: ideduct_synth_mono)
      then obtain s where s: "s ∈ ikst Apre" "Fun f T ⊆ s · I"
        using assms(5) *(2) by (induct rule: intruder_synth_induct) auto
      then obtain g S where gS: "Fun g S ⊆ s" "Fun f T = Fun g S · I"
        using subterm_subst_not_img_subterm[OF s(2)] *(1) by force
      hence ?thesis using that *(1) s(1) by force
    }
    moreover
    { fix t t' assume **: "A = Apre@Equality Assign t t'#Asuf" "Fun f T ⊆ t · I"
      with assms(2) have "t · I = t' · I" by auto
      hence "Fun f T ⊆ t' · I" using *(2) by auto
      from assms(1) *(1) have "fv t' ⊆ wfrestrictedvarsst Apre"
        using wf_eq_fv[of "{}" Apre t t' Asuf] vars_snd_rcv_strand_subset2(4)[of Apre]
        by blast
      then obtain g S where gS: "Fun g S ⊆ t'" "Fun f T = Fun g S · I"
        using subterm_subst_not_img_subterm[OF ⟨Fun f T ⊆ t' · I⟩] *(1) by fastforce
      hence ?thesis using that *(1) by auto
    }
  }
  ultimately show ?thesis by auto
qed

lemma strand_sem_not_unif_is_sat_ineq:
  assumes "∄ ∅. Unifier ∅ t t'"
  shows "[M; [Inequality X [(t, t')]]]c I" "[M; [Inequality X [(t, t')]]]d I"

```

```

using assms list_ex_simps(1)[of _ "(t,t')" "[]"] prod.sel[of t t']
      strand_sem_c.simps(1,5) strand_sem_d.simps(1,5)
unfolding ineq_model_def by presburger+

lemma ineq_model_singleI[intro]:
  assumes " $\forall \delta. \text{subst\_domain } \delta = \text{set } X \wedge \text{ground } (\text{subst\_range } \delta) \longrightarrow t \cdot \delta \cdot \mathcal{I} \neq t' \cdot \delta \cdot \mathcal{I}$ "
  shows "ineq_model  $\mathcal{I}$  X [(t,t')]"
using assms unfolding ineq_model_def by auto

lemma ineq_model_singleE:
  assumes "ineq_model  $\mathcal{I}$  X [(t,t')]"
  shows " $\forall \delta. \text{subst\_domain } \delta = \text{set } X \wedge \text{ground } (\text{subst\_range } \delta) \longrightarrow t \cdot \delta \cdot \mathcal{I} \neq t' \cdot \delta \cdot \mathcal{I}$ "
using assms unfolding ineq_model_def by auto

lemma ineq_model_single_iff:
  fixes F::"('a,'b) term  $\times$  ('a,'b) term) list"
  shows "ineq_model  $\mathcal{I}$  X F  $\longleftrightarrow$ 
    ineq_model  $\mathcal{I}$  X [(Fun f (Fun c []#map fst F), Fun f (Fun c []#map snd F))]"
    (is "?A  $\longleftrightarrow$  ?B")
proof -
  let ?P = " $\lambda \delta f. \text{fst } f \cdot (\delta \circ_s \mathcal{I}) \neq \text{snd } f \cdot (\delta \circ_s \mathcal{I})$ "
  let ?Q = " $\lambda \delta t t'. t \cdot (\delta \circ_s \mathcal{I}) \neq t' \cdot (\delta \circ_s \mathcal{I})$ "
  let ?T = " $\lambda g. \text{Fun } c []\#map g F$ "
  let ?S = " $\lambda \delta g. \text{map } (\lambda x. x \cdot (\delta \circ_s \mathcal{I})) (\text{Fun } c []\#map g F)$ "
  let ?t = "Fun f (?T fst)"
  let ?t' = "Fun f (?T snd)"

  have len: " $\bigwedge g h. \text{length } (?T g) = \text{length } (?T h)$ "
    " $\bigwedge g h \delta. \text{length } (?S \delta g) = \text{length } (?T h)$ "
    " $\bigwedge g h \delta. \text{length } (?S \delta g) = \text{length } (?T h)$ "
    " $\bigwedge g h \delta \sigma. \text{length } (?S \delta g) = \text{length } (?S \sigma h)$ "
  by simp_all

  { fix  $\delta$ ::"('a,'b) subst"
    assume  $\delta$ : "subst_domain  $\delta = \text{set } X$ " "ground (subst_range  $\delta$ )"
    have "list_ex (?P  $\delta$ ) F  $\longleftrightarrow$  ?Q  $\delta$  ?t ?t'"
    proof
      assume "list_ex (?P  $\delta$ ) F"
      then obtain a where a: " $a \in \text{set } F$ " "?P  $\delta$  a" by (metis (mono_tags, lifting) Bex_set)
      thus "?Q  $\delta$  ?t ?t'" by auto
    qed (fastforce simp add: Bex_set)
  } thus ?thesis unfolding ineq_model_def by auto
qed

```

### 3.1.7 Constraint Semantics (Alternative, Equivalent Version)

These are the constraint semantics used in the CSF 2017 paper

```

fun strand_sem_c'::"('fun,'var) terms  $\Rightarrow$  ('fun,'var) strand  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  bool" ("[_;
_]'_c'")
where
  "[M; []]'_c' = ( $\lambda \mathcal{I}. \text{True}$ )"
  | "[M; Send t#S]'_c' = ( $\lambda \mathcal{I}. M \cdot_{\text{set}} \mathcal{I} \vdash_c t \cdot \mathcal{I} \wedge [M; S]'_c' \mathcal{I}$ )"
  | "[M; Receive t#S]'_c' = [insert t M; S]'_c'"
  | "[M; Equality _ t t'#S]'_c' = ( $\lambda \mathcal{I}. t \cdot \mathcal{I} = t' \cdot \mathcal{I} \wedge [M; S]'_c' \mathcal{I}$ )"
  | "[M; Inequality X F#S]'_c' = ( $\lambda \mathcal{I}. \text{ineq\_model } \mathcal{I} X F \wedge [M; S]'_c' \mathcal{I}$ )"

fun strand_sem_d'::"('fun,'var) terms  $\Rightarrow$  ('fun,'var) strand  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  bool" ("[_;
_]'_d'")
where
  "[M; []]'_d' = ( $\lambda \mathcal{I}. \text{True}$ )"
  | "[M; Send t#S]'_d' = ( $\lambda \mathcal{I}. M \cdot_{\text{set}} \mathcal{I} \vdash t \cdot \mathcal{I} \wedge [M; S]'_d' \mathcal{I}$ )"
  | "[M; Receive t#S]'_d' = [insert t M; S]'_d'"

```

```

/ "[M; Equality _ t t'#S]_d' = (λI. t · I = t' · I ∧ [M; S]_d' I)"
/ "[M; Inequality X F#S]_d' = (λI. ineq_model I X F ∧ [M; S]_d' I)"

lemma strand_sem_eq_defs:
  "[M; A]_c' I = [M ·_set I; A]_c I"
  "[M; A]_d' I = [M ·_set I; A]_d I"
proof -
  have 1: "[M; A]_c' I ⇒ [M ·_set I; A]_c I"
    by (induct A arbitrary: M rule: strand_sem_induct) force+
  have 2: "[M ·_set I; A]_c I ⇒ [M; A]_c' I"
    by (induct A arbitrary: M rule: strand_sem_c'.induct) auto
  have 3: "[M; A]_d' I ⇒ [M ·_set I; A]_d I"
    by (induct A arbitrary: M rule: strand_sem_induct) force+
  have 4: "[M ·_set I; A]_d I ⇒ [M; A]_d' I"
    by (induct A arbitrary: M rule: strand_sem_d'.induct) auto

  show "[M; A]_c' I = [M ·_set I; A]_c I" "[M; A]_d' I = [M ·_set I; A]_d I"
    by (metis 1 2, metis 3 4)
qed

lemma strand_sem_split'[dest]:
  "[M; S@S']_c' ∅ ⇒ [M; S]_c' ∅"
  "[M; S@S']_c' ∅ ⇒ [M ∪ ik_st S; S']_c' ∅"
  "[M; S@S']_d' ∅ ⇒ [M; S]_d' ∅"
  "[M; S@S']_d' ∅ ⇒ [M ∪ ik_st S; S']_d' ∅"
using strand_sem_eq_defs[of M "S@S'" ∅]
  strand_sem_eq_defs[of M S ∅]
  strand_sem_eq_defs[of "M ∪ ik_st S" S' ∅]
  strand_sem_split(2,4)
by (auto simp add: image_Un)

lemma strand_sem_append'[intro]:
  "[M; S]_c' ∅ ⇒ [M ∪ ik_st S; S']_c' ∅ ⇒ [M; S@S']_c' ∅"
  "[M; S]_d' ∅ ⇒ [M ∪ ik_st S; S']_d' ∅ ⇒ [M; S@S']_d' ∅"
using strand_sem_eq_defs[of M "S@S'" ∅]
  strand_sem_eq_defs[of M S ∅]
  strand_sem_eq_defs[of "M ∪ ik_st S" S' ∅]
by (auto simp add: image_Un)

end

```

### 3.1.8 Dual Strands

```

fun dual_st :: "('a, 'b) strand ⇒ ('a, 'b) strand" where
  "dual_st [] = []"
/ "dual_st (Receive t#S) = Send t#(dual_st S)"
/ "dual_st (Send t#S) = Receive t#(dual_st S)"
/ "dual_st (x#S) = x#(dual_st S)"

lemma dual_st_append: "dual_st (A@B) = (dual_st A)@(dual_st B)"
by (induct A rule: dual_st.induct) auto

lemma dual_st_self_inverse: "dual_st (dual_st S) = S"
proof (induction S)
  case (Cons x S) thus ?case by (cases x) auto
qed simp

lemma dual_st_trms_eq: "trms_st (dual_st S) = trms_st S"
proof (induction S)
  case (Cons x S) thus ?case by (cases x) auto
qed simp

lemma dual_st_fv: "fv_st (dual_st A) = fv_st A"

```



```

by (induct A rule: dualst.induct) auto

lemma dualst_bvars: "bvarsst (dualst A) = bvarsst A"
by (induct A rule: dualst.induct) fastforce+

end

```

## 3.2 The Lazy Intruder (Lazy\_Intruder)

```

theory Lazy_Intruder
imports Strands_and_Constraints Intruder_Deduction
begin

context intruder_model
begin

```

### 3.2.1 Definition of the Lazy Intruder

The lazy intruder constraint reduction system, defined as a relation on constraint states

```

inductive_set LI_rel::
  "((('fun,'var) strand × (('fun,'var) subst)) ×
    ('fun,'var) strand × (('fun,'var) subst)) set"
and LI_rel' (infix "↔" 50)
and LI_rel_trancl (infix "↔+" 50)
and LI_rel_rtrancl (infix "↔*" 50)
where
  "A ↔ B ≡ (A,B) ∈ LI_rel"
| "A ↔+ B ≡ (A,B) ∈ LI_rel+"
| "A ↔* B ≡ (A,B) ∈ LI_rel*"

| Compose: "[simple S; length T = arity f; public f]
  ⇒ (S@Send (Fun f T)#S',ϑ) ↔ (S@map Send T@S',ϑ)"
| Unify: "[simple S; Fun f T' ∈ ikst S; Some δ = mgu (Fun f T) (Fun f T')]
  ⇒ (S@Send (Fun f T)#S',ϑ) ↔ ((S@S') ·st δ,ϑ ∘s δ)"
| Equality: "[simple S; Some δ = mgu t t']
  ⇒ (S@Equality _ t t'#S',ϑ) ↔ ((S@S') ·st δ,ϑ ∘s δ)"

```

### 3.2.2 Lemma: The Lazy Intruder is Well-founded

```

context
begin
private lemma LI_compose_measure_lt: "((S@map Send T@S',ϑ1), (S@Send (Fun f T)#S',ϑ2)) ∈
measurest"
using strand_fv_card_map_fun_eq[of S f T S'] strand_size_map_fun_lt(2)[of T f]
by (simp add: measurest_def sizest_def)

private lemma LI_unify_measure_lt:
  assumes "Some δ = mgu (Fun f T) t" "fv t ⊆ fvst S"
  shows "((S@S') ·st δ,ϑ1), (S@Send (Fun f T)#S',ϑ2)) ∈ measurest"
proof (cases "δ = Var")
  assume "δ = Var"
  hence "(S@S') ·st δ = S@S'" by blast
  thus ?thesis
    using strand_fv_card_rm_fun_le[of S S' f T]
    by (auto simp add: measurest_def sizest_def)
next
  assume "δ ≠ Var"
  then obtain v where "v ∈ fv (Fun f T) ∪ fv t" "subst_elim δ v"
    using mgu_eliminate[OF assms(1)[symmetric]] by metis
  hence v_in: "v ∈ fvst (S@Send (Fun f T)#S')"
    using assms(2) by (auto simp add: measurest_def sizest_def)

```

```

have "range_vars  $\delta \subseteq \text{fv } (\text{Fun } f \text{ } T) \cup \text{fv}_{st} \text{ } S"$ "
  using assms(2) mgu_vars_bounded[OF assms(1)[symmetric]] by auto
hence img_bound: "range_vars  $\delta \subseteq \text{fv}_{st} \text{ } (S@Send \text{ } (\text{Fun } f \text{ } T)\#S')$ " by auto

have finite_fv: "finite ( $\text{fv}_{st} \text{ } (S@Send \text{ } (\text{Fun } f \text{ } T)\#S')$ )" by auto

have " $v \notin \text{fv}_{st} \text{ } ((S@Send \text{ } (\text{Fun } f \text{ } T)\#S') \cdot_{st} \delta)$ "
  using strand_fv_subst_subset_if_subst_elim[OF <subst_elim  $\delta \text{ } v$ >] v_in by metis
hence v_not_in: " $v \notin \text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta)$ " by auto

have " $\text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta) \subseteq \text{fv}_{st} \text{ } (S@Send \text{ } (\text{Fun } f \text{ } T)\#S')$ "
  using strand_subst_fv_bounded_if_img_bounded[OF img_bound] by simp
hence " $\text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta) \subset \text{fv}_{st} \text{ } (S@Send \text{ } (\text{Fun } f \text{ } T)\#S')$ " using v_in v_not_in by blast
hence "card ( $\text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta)) < \text{card } (\text{fv}_{st} \text{ } (S@Send \text{ } (\text{Fun } f \text{ } T)\#S'))"$ "
  using psubset_card_mono[OF finite_fv] by simp
thus ?thesis by (auto simp add: measure_st_def size_st_def)
qed

private lemma LI_equality_measure_lt:
  assumes "Some  $\delta = \text{mgu } t \text{ } t'$ "
  shows " $((S@S') \cdot_{st} \delta, \vartheta_1), (S@Equality \text{ } a \text{ } t \text{ } t'\#S', \vartheta_2)) \in \text{measure}_{st}$ "
proof (cases " $\delta = \text{Var}$ ")
  assume " $\delta = \text{Var}$ "
  hence " $(S@S') \cdot_{st} \delta = S@S'$ " by blast
  thus ?thesis
    using strand_fv_card_rm_eq_le[of  $S \text{ } S' \text{ } a \text{ } t \text{ } t'$ ]
    by (auto simp add: measure_st_def size_st_def)
next
  assume " $\delta \neq \text{Var}$ "
  then obtain v where " $v \in \text{fv } t \cup \text{fv } t'$ " "subst_elim  $\delta \text{ } v$ "
    using mgu_eliminate[OF assms(1)[symmetric]] by metis
  hence v_in: " $v \in \text{fv}_{st} \text{ } (S@Equality \text{ } a \text{ } t \text{ } t'\#S')$ " using assms by auto

  have "range_vars  $\delta \subseteq \text{fv } t \cup \text{fv } t' \cup \text{fv}_{st} \text{ } S"$ "
    using assms mgu_vars_bounded[OF assms(1)[symmetric]] by auto
  hence img_bound: "range_vars  $\delta \subseteq \text{fv}_{st} \text{ } (S@Equality \text{ } a \text{ } t \text{ } t'\#S')$ " by auto

  have finite_fv: "finite ( $\text{fv}_{st} \text{ } (S@Equality \text{ } a \text{ } t \text{ } t'\#S')$ )" by auto

  have " $v \notin \text{fv}_{st} \text{ } ((S@Equality \text{ } a \text{ } t \text{ } t'\#S') \cdot_{st} \delta)$ "
    using strand_fv_subst_subset_if_subst_elim[OF <subst_elim  $\delta \text{ } v$ >] v_in by metis
  hence v_not_in: " $v \notin \text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta)$ " by auto

  have " $\text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta) \subseteq \text{fv}_{st} \text{ } (S@Equality \text{ } a \text{ } t \text{ } t'\#S')$ "
    using strand_subst_fv_bounded_if_img_bounded[OF img_bound] by simp
  hence " $\text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta) \subset \text{fv}_{st} \text{ } (S@Equality \text{ } a \text{ } t \text{ } t'\#S')$ " using v_in v_not_in by blast
  hence "card ( $\text{fv}_{st} \text{ } ((S@S') \cdot_{st} \delta)) < \text{card } (\text{fv}_{st} \text{ } (S@Equality \text{ } a \text{ } t \text{ } t'\#S'))"$ "
    using psubset_card_mono[OF finite_fv] by simp
  thus ?thesis by (auto simp add: measure_st_def size_st_def)
qed

private lemma LI_in_measure: " $(S_1, \vartheta_1) \rightsquigarrow (S_2, \vartheta_2) \implies ((S_2, \vartheta_2), (S_1, \vartheta_1)) \in \text{measure}_{st}$ "
proof (induction rule: LI_rel.induct)
  case (Compose  $S \text{ } T \text{ } f \text{ } S' \text{ } \vartheta$ ) thus ?case using LI_compose_measure_lt[of  $S \text{ } T \text{ } S'$ ] by metis
next
  case (Unify  $S \text{ } f \text{ } U \text{ } \delta \text{ } T \text{ } S' \text{ } \vartheta$ )
  hence " $\text{fv } (\text{Fun } f \text{ } U) \subseteq \text{fv}_{st} \text{ } S"$ "
    using fv_snd_rcv_strand_subset(2)[of  $S$ ] by force
  thus ?case using LI_unify_measure_lt[OF Unify.hyps(3), of  $S \text{ } S'$ ] by metis
qed (metis LI_equality_measure_lt)

private lemma LI_in_measure_trans: " $(S_1, \vartheta_1) \rightsquigarrow^+ (S_2, \vartheta_2) \implies ((S_2, \vartheta_2), (S_1, \vartheta_1)) \in \text{measure}_{st}$ "

```

```

by (induction rule: trancl.induct, metis surjective_pairing LI_in_measure)
  (metis (no_types, lifting) surjective_pairing LI_in_measure measurest_trans trans_def)

private lemma LI_converse_wellfounded_trans: "wf ((LI_rel+)-1)"
proof -
  have "(LI_rel+)-1 ⊆ measurest" using LI_in_measure_trans by auto
  thus ?thesis using measurest_wellfounded wf_subset by metis
qed

private lemma LI_acyclic_trans: "acyclic (LI_rel+)"
using wf_acyclic[OF LI_converse_wellfounded_trans] acyclic_converse by metis

private lemma LI_acyclic: "acyclic LI_rel"
using LI_acyclic_trans acyclic_subset by (simp add: acyclic_def)

lemma LI_no_infinite_chain: "¬(∃f. ∀i. f i ~+ f (Suc i))"
proof -
  have "¬(∃f. ∀i. (f (Suc i), f i) ∈ (LI_rel+)-1)"
    using wf_iff_no_infinite_down_chain LI_converse_wellfounded_trans by metis
  thus ?thesis by simp
qed

private lemma LI_unify_finite:
  assumes "finite M"
  shows "finite {(S@Send (Fun f T)#S', ∅), ((S@S') ·st δ, ∅ ∘s δ)) | δ T'.
    simple S ∧ Fun f T' ∈ M ∧ Some δ = mgu (Fun f T) (Fun f T')}"
using assms
proof (induction M rule: finite_induct)
  case (insert m M) thus ?case
  proof (cases m)
    case (Fun g U)
    let ?a = "λδ. ((S@Send (Fun f T)#S', ∅), ((S@S') ·st δ, ∅ ∘s δ))"
    let ?A = "λB. {?a δ | δ T'. simple S ∧ Fun f T' ∈ B ∧ Some δ = mgu (Fun f T) (Fun f T')}"
    have "?A (insert m M) = (?A M) ∪ (?A {m})" by auto
    moreover have "finite (?A {m})"
    proof (cases "∃δ. Some δ = mgu (Fun f T) (Fun g U)")
      case True
      then obtain δ where δ: "Some δ = mgu (Fun f T) (Fun g U)" by blast
      have A_m_eq: "λδ'. ?a δ' ∈ ?A {m} ⟹ ?a δ = ?a δ'"
      proof -
        fix δ' assume "?a δ' ∈ ?A {m}"
        hence "∃σ. Some σ = mgu (Fun f T) (Fun g U) ∧ ?a σ = ?a δ'"
          using ⟨m = Fun g U⟩ by auto
        thus "?a δ = ?a δ'" by (metis δ option.inject)
      qed
      have "?A {m} = {} ∨ ?A {m} = {?a δ}"
      proof (cases "simple S ∧ ?A {m} ≠ {}")
        case True
        hence "simple S" "?A {m} ≠ {}" by meson+
        hence "?A {m} = {?a δ | δ. Some δ = mgu (Fun f T) (Fun g U)}" using ⟨m = Fun g U⟩ by auto
        hence "?a δ ∈ ?A {m}" using δ by auto
        show ?thesis
        proof (rule ccontr)
          assume "¬(?A {m} = {} ∨ ?A {m} = {?a δ})"
          then obtain B where B: "?A {m} = insert (?a δ) B" "?a δ ∉ B" "B ≠ {}"
            using ⟨?A {m} ≠ {}⟩ ⟨?a δ ∈ ?A {m}⟩ by (metis (no_types, lifting) Set.set_insert)
          then obtain b where b: "?a δ ≠ b" "b ∈ B" by (metis (no_types, lifting) ex_in_conv)
          then obtain δ' where δ': "b = ?a δ'" using B(1) by blast
          moreover have "?a δ' ∈ ?A {m}" using B(1) b(2) δ' by auto
          hence "?a δ = ?a δ'" by (blast dest!: A_m_eq)
        qed
      qed
    case False
    then show ?case by auto
  qed
end

```

```

      ultimately show False using b(1) by simp
    qed
  qed auto
  thus ?thesis by (metis (no_types, lifting) finite.emptyI finite_insert)
next
  case False
  hence "?A {m} = {}" using ⟨m = Fun g U⟩ by blast
  thus ?thesis by (metis finite.emptyI)
qed
ultimately show ?thesis using insert.IH by auto
qed simp
qed fastforce
end

```

### 3.2.3 Lemma: The Lazy Intruder Preserves Well-formedness

```

context
begin
private lemma LI_preserves_subst_wf_single:
  assumes "(S1, ϑ1) ~> (S2, ϑ2)" "fvst S1 ∩ bvarsst S1 = {}" "wfsubst ϑ1"
  and "subst_domain ϑ1 ∩ varsst S1 = {}" "range_vars ϑ1 ∩ bvarsst S1 = {}"
  shows "fvst S2 ∩ bvarsst S2 = {}" "wfsubst ϑ2"
  and "subst_domain ϑ2 ∩ varsst S2 = {}" "range_vars ϑ2 ∩ bvarsst S2 = {}"
using assms
proof (induction rule: LI_rel.induct)
  case (Compose S X f S' ϑ)
  { case 1 thus ?case using vars_st_snd_map by auto }
  { case 2 thus ?case using vars_st_snd_map by auto }
  { case 3 thus ?case using vars_st_snd_map by force }
  { case 4 thus ?case using vars_st_snd_map by auto }
next
  case (Unify S f U δ T S' ϑ)
  hence "fv (Fun f U) ⊆ fvst S" using fv_subset_if_in_strand_ik' by blast
  hence *: "subst_domain δ ∪ range_vars δ ⊆ fvst (S@Send (Fun f T)#S')"
    using mgu_vars_bounded[OF Unify.hyps(3)[symmetric]]
    unfolding range_vars_alt_def by (fastforce simp del: subst_range.simps)

  have "fvst (S@S') ⊆ fvst (S@Send (Fun f T)#S')" "varsst (S@S') ⊆ varsst (S@Send (Fun f T)#S')"
    by auto
  hence **: "fvst (S@S' ·st δ) ⊆ fvst (S@Send (Fun f T)#S')"
    "varsst (S@S' ·st δ) ⊆ varsst (S@Send (Fun f T)#S')"
    using subst_sends_strand_fv_to_img[of "S@S'" δ]
    strand_subst_vars_union_bound[of "S@S'" δ] *
    by blast+

  have "wfsubst δ" by (fact mgu_gives_wellformed_subst[OF Unify.hyps(3)[symmetric]])

  { case 1
    have "bvarsst (S@S' ·st δ) = bvarsst (S@Send (Fun f T)#S')"
      using bvars_subst_ident[of "S@S'" δ] by auto
    thus ?case using 1 ** by blast
  }
  { case 2
    hence "subst_domain ϑ ∩ subst_domain δ = {}" "subst_domain ϑ ∩ range_vars δ = {}"
      using * by blast+
    thus ?case by (metis wf_subst_compose[OF ⟨wfsubst ϑ⟩ ⟨wfsubst δ⟩])
  }
  { case 3
    hence "subst_domain ϑ ∩ varsst (S@S' ·st δ) = {}" using ** by blast
    moreover have "v ∈ fvst (S@Send (Fun f T)#S')" when "v ∈ subst_domain δ" for v
      using * that by blast
    hence "subst_domain δ ∩ fvst (S@S' ·st δ) = {}"
      using mgu_eliminate_dom[OF Unify.hyps(3)[symmetric],

```

```

      THEN strand_fv_subst_subset_if_subst_elim, of _ "S@Send (Fun f T)#S'"
    unfolding subst_elim_def by auto
  moreover have "bvarsst (S@S' .st δ) = bvarsst (S@Send (Fun f T)#S'"
    using bvars_subst_ident[of "S@S'" δ] by auto
  hence "subst_domain δ ∩ bvarsst (S@S' .st δ) = {}" using 3(1) * by blast
  ultimately show ?case
    using ** * subst_domain_compose[of ∅ δ] varsst_is_fvst_bvarsst[of "S@S' .st δ"]
    by blast
}
{ case 4
  have ***: "bvarsst (S@S' .st δ) = bvarsst (S@Send (Fun f T)#S'"
    using bvars_subst_ident[of "S@S'" δ] by auto
  hence "range_vars δ ∩ bvarsst (S@S' .st δ) = {}" using 4(1) * by blast
  thus ?case using subst_img_comp_subset[of ∅ δ] 4(4) *** by blast
}
next
case (Equality S δ t t' a S' ∅)
hence *: "subst_domain δ ∪ range_vars δ ⊆ fvst (S@Equality a t t'#S'"
  using mgu_vars_bounded[OF Equality.hyps(2)[symmetric]]
  unfolding range_vars_alt_def by fastforce

have "fvst (S@S') ⊆ fvst (S@Equality a t t'#S'" "varsst (S@S') ⊆ varsst (S@Equality a t t'#S'"
  by auto
hence **: "fvst (S@S' .st δ) ⊆ fvst (S@Equality a t t'#S'"
  "varsst (S@S' .st δ) ⊆ varsst (S@Equality a t t'#S'"
  using subst_sends_strand_fv_to_img[of "S@S'" δ]
  strand_subst_vars_union_bound[of "S@S'" δ] *
  by blast+

have "wfsubst δ" by (fact mgu_gives_wellformed_subst[OF Equality.hyps(2)[symmetric]])

{ case 1
  have "bvarsst (S@S' .st δ) = bvarsst (S@Equality a t t'#S'"
    using bvars_subst_ident[of "S@S'" δ] by auto
  thus ?case using 1 ** by blast
}
{ case 2
  hence "subst_domain ∅ ∩ subst_domain δ = {}" "subst_domain ∅ ∩ range_vars δ = {}"
    using * by blast+
  thus ?case by (metis wf_subst_compose[OF ⟨wfsubst ∅⟩ ⟨wfsubst δ⟩])
}
{ case 3
  hence "subst_domain ∅ ∩ varsst (S@S' .st δ) = {}" using ** by blast
  moreover have "v ∈ fvst (S@Equality a t t'#S'" when "v ∈ subst_domain δ" for v
    using * that by blast
  hence "subst_domain δ ∩ fvst (S@S' .st δ) = {}"
    using mgu_eliminator_dom[OF Equality.hyps(2)[symmetric],
      THEN strand_fv_subst_subset_if_subst_elim, of _ "S@Equality a t t'#S'"
    unfolding subst_elim_def by auto
  moreover have "bvarsst (S@S' .st δ) = bvarsst (S@Equality a t t'#S'"
    using bvars_subst_ident[of "S@S'" δ] by auto
  hence "subst_domain δ ∩ bvarsst (S@S' .st δ) = {}" using 3(1) * by blast
  ultimately show ?case
    using ** * subst_domain_compose[of ∅ δ] varsst_is_fvst_bvarsst[of "S@S' .st δ"]
    by blast
}
{ case 4
  have ***: "bvarsst (S@S' .st δ) = bvarsst (S@Equality a t t'#S'"
    using bvars_subst_ident[of "S@S'" δ] by auto
  hence "range_vars δ ∩ bvarsst (S@S' .st δ) = {}" using 4(1) * by blast
  thus ?case using subst_img_comp_subset[of ∅ δ] 4(4) *** by blast
}
}
qed

```

```

private lemma LI_preserves_subst_wf:
  assumes "(S1,  $\vartheta_1$ )  $\rightsquigarrow^*$  (S2,  $\vartheta_2$ )" "fvst S1  $\cap$  bvarsst S1 = {}" "wfsubst  $\vartheta_1$ "
  and "subst_domain  $\vartheta_1 \cap$  varsst S1 = {}" "range_vars  $\vartheta_1 \cap$  bvarsst S1 = {}"
  shows "fvst S2  $\cap$  bvarsst S2 = {}" "wfsubst  $\vartheta_2$ "
  and "subst_domain  $\vartheta_2 \cap$  varsst S2 = {}" "range_vars  $\vartheta_2 \cap$  bvarsst S2 = {}"
using assms
proof (induction S2  $\vartheta_2$  rule: rtranc1_induct2)
  case (step Si  $\vartheta_i$  Sj  $\vartheta_j$ )
  { case 1 thus ?case using LI_preserves_subst_wf_single[OF  $\langle (S_i, \vartheta_i) \rightsquigarrow (S_j, \vartheta_j) \rangle$ ] step.IH by metis }
  { case 2 thus ?case using LI_preserves_subst_wf_single[OF  $\langle (S_i, \vartheta_i) \rightsquigarrow (S_j, \vartheta_j) \rangle$ ] step.IH by metis }
  { case 3 thus ?case using LI_preserves_subst_wf_single[OF  $\langle (S_i, \vartheta_i) \rightsquigarrow (S_j, \vartheta_j) \rangle$ ] step.IH by metis }
  { case 4 thus ?case using LI_preserves_subst_wf_single[OF  $\langle (S_i, \vartheta_i) \rightsquigarrow (S_j, \vartheta_j) \rangle$ ] step.IH by metis }
qed metis

lemma LI_preserves_wellformedness:
  assumes "(S1,  $\vartheta_1$ )  $\rightsquigarrow^*$  (S2,  $\vartheta_2$ )" "wfconstr S1  $\vartheta_1$ "
  shows "wfconstr S2  $\vartheta_2$ "
proof -
  have *: "wfst {} Sj"
  when "(Si,  $\vartheta_i$ )  $\rightsquigarrow$  (Sj,  $\vartheta_j$ )" "wfconstr Si  $\vartheta_i$ " for Si  $\vartheta_i$  Sj  $\vartheta_j$ 
  using that
proof (induction rule: LI_rel.induct)
  case (Unify S f U  $\delta$  T S'  $\vartheta$ )
  have "fv (Fun f T)  $\cup$  fv (Fun f U)  $\subseteq$  fvst (S@Send (Fun f T)#S')" using Unify.hyps(2) by force
  hence "subst_domain  $\delta \cup$  range_vars  $\delta \subseteq$  fvst (S@Send (Fun f T)#S')"
  using mgu_vars_bounded[OF Unify.hyps(3)[symmetric]] by (metis subset_trans)
  hence "(subst_domain  $\delta \cup$  range_vars  $\delta) \cap$  bvarsst (S@Send (Fun f T)#S') = {}"
  using Unify.prem1 unfolding wfconstr_def by blast
  thus ?case
  using wf_unify[OF _ Unify.hyps(2) MGU_is_Unifier[OF mgu_gives_MGU], of "{}",
    OF _ Unify.hyps(3)[symmetric], of S'] Unify.prem1
  by (auto simp add: wfconstr_def)
next
  case (Equality S  $\delta$  t t' a S'  $\vartheta$ )
  have "fv t  $\cup$  fv t'  $\subseteq$  fvst (S@Equality a t t'#S')" using Equality.hyps(2) by force
  hence "subst_domain  $\delta \cup$  range_vars  $\delta \subseteq$  fvst (S@Equality a t t'#S')"
  using mgu_vars_bounded[OF Equality.hyps(2)[symmetric]] by (metis subset_trans)
  hence "(subst_domain  $\delta \cup$  range_vars  $\delta) \cap$  bvarsst (S@Equality a t t'#S') = {}"
  using Equality.prem1 unfolding wfconstr_def by blast
  thus ?case
  using wf_equality[OF _ Equality.hyps(2)[symmetric], of "{}" S a S'] Equality.prem1
  by (auto simp add: wfconstr_def)
qed (metis wf_send_compose wfconstr_def)

show ?thesis using assms
proof (induction rule: rtranc1_induct2)
  case (step Si  $\vartheta_i$  Sj  $\vartheta_j$ ) thus ?case
  using LI_preserves_subst_wf_single[OF  $\langle (S_i, \vartheta_i) \rightsquigarrow (S_j, \vartheta_j) \rangle$ ] *[OF  $\langle (S_i, \vartheta_i) \rightsquigarrow (S_j, \vartheta_j) \rangle$ ]
  by (metis wfconstr_def)
qed simp
qed

lemma LI_preserves_trm_wf:
  assumes "(S,  $\vartheta$ )  $\rightsquigarrow^*$  (S',  $\vartheta'$ )" "wftrms (trmsst S)"
  shows "wftrms (trmsst S')"
proof -
  { fix S  $\vartheta$  S'  $\vartheta'$ 
    assume "(S,  $\vartheta$ )  $\rightsquigarrow$  (S',  $\vartheta'$ )" "wftrms (trmsst S)"
    hence "wftrms (trmsst S)"
    proof (induction rule: LI_rel.induct)
      case (Compose S T f S'  $\vartheta$ )
      hence "wftrm (Fun f T)"
    end
  }

```

```

and *: "t ∈ set S ⇒ wf_trms (trms_stp t)" "t ∈ set S' ⇒ wf_trms (trms_stp t)" for t
by auto
hence "wf_trm t" when "t ∈ set T" for t using that unfolding wf_trm_def by auto
hence "wf_trms (trms_stp t)" when "t ∈ set (map Send T)" for t
  using that unfolding wf_trm_def by auto
thus ?case using * by force
next
case (Unify S f U δ T S' ∅)
have "wf_trm (Fun f T)" "wf_trm (Fun f U)"
  using Unify.prem1(1) Unify.hyps(2) wf_trm_subterm[of _ "Fun f U"]
  by (simp, force)
hence range_wf: "wf_trms (subst_range δ)"
  using mgu_wf_trm[OF Unify.hyps(3)[symmetric]] by simp

{ fix s assume "s ∈ set (S@S' ·_st δ)"
  hence "∃ s' ∈ set (S@S'). s = s' ·_stp δ ∧ wf_trms (trms_stp s')"
    using Unify.prem1(1) by (auto simp add: subst_apply_strand_def)
  moreover {
    fix s' assume s': "s = s' ·_stp δ" "wf_trms (trms_stp s')" "s' ∈ set (S@S')"
    from s'(2) have "trms_stp (s' ·_stp δ) = trms_stp s' ·_set (rm_vars (set (bvars_stp s'))) δ"
    proof (induction s')
      case (Inequality X F) thus ?case by (induct F) (auto simp add: subst_apply_pairs_def)
    qed auto
    hence "wf_trms (trms_stp s)"
      using wf_trm_subst[OF wf_trms_subst_rm_vars'[OF range_wf]] (wf_trms (trms_stp s')) s'(1)
      by simp
  }
  ultimately have "wf_trms (trms_stp s)" by auto
}
thus ?case by auto
next
case (Equality S δ t t' a S' ∅)
hence "wf_trm t" "wf_trm t'" by simp_all
hence range_wf: "wf_trms (subst_range δ)"
  using mgu_wf_trm[OF Equality.hyps(2)[symmetric]] by simp

{ fix s assume "s ∈ set (S@S' ·_st δ)"
  hence "∃ s' ∈ set (S@S'). s = s' ·_stp δ ∧ wf_trms (trms_stp s')"
    using Equality.prem1(1) by (auto simp add: subst_apply_strand_def)
  moreover {
    fix s' assume s': "s = s' ·_stp δ" "wf_trms (trms_stp s')" "s' ∈ set (S@S')"
    from s'(2) have "trms_stp (s' ·_stp δ) = trms_stp s' ·_set (rm_vars (set (bvars_stp s'))) δ"
    proof (induction s')
      case (Inequality X F) thus ?case by (induct F) (auto simp add: subst_apply_pairs_def)
    qed auto
    hence "wf_trms (trms_stp s)"
      using wf_trm_subst[OF wf_trms_subst_rm_vars'[OF range_wf]] (wf_trms (trms_stp s')) s'(1)
      by simp
  }
  ultimately have "wf_trms (trms_stp s)" by auto
}
thus ?case by auto
qed
}
with assms show ?thesis by (induction rule: rtrancl_induct2) metis+
qed
end

```

### 3.2.4 Theorem: Soundness of the Lazy Intruder

```

context
begin
private lemma LI_soundness_single:

```

```

assumes "wf_constr S1  $\vartheta_1$ " "(S1,  $\vartheta_1$ )  $\rightsquigarrow$  (S2,  $\vartheta_2$ )" " $\mathcal{I} \models_c \langle S_2, \vartheta_2 \rangle$ "
shows " $\mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle$ "
using assms(2,1,3)
proof (induction rule: LI_rel.induct)
  case (Compose S T f S'  $\vartheta$ )
  hence *: "[{ }; S]c  $\mathcal{I}$ " "[ikst S ·set  $\mathcal{I}$ ; map Send T]c  $\mathcal{I}$ " "[ikst S ·set  $\mathcal{I}$ ; S']c  $\mathcal{I}$ "
    unfolding constr_sem_c_def by force+

  have "ikst S ·set  $\mathcal{I} \vdash_c$  Fun f T ·  $\mathcal{I}$ "
    using *(2) Compose.hyps(2) ComposeC[OF _ Compose.hyps(3), of "map ( $\lambda x. x \cdot \mathcal{I}$ ) T"]
    unfolding subst_compose_def by force
  thus " $\mathcal{I} \models_c \langle S@Send (Fun f T)#S', \vartheta \rangle$ "
    using *(1,3) " $\mathcal{I} \models_c \langle S@map Send T@S', \vartheta \rangle$ "
    by (auto simp add: constr_sem_c_def)
next
  case (Unify S f U  $\delta$  T S'  $\vartheta$ )
  have " $(\vartheta \circ_s \delta)$  supports  $\mathcal{I}$ " "[{ }; S@S' ·st  $\delta$ ]c  $\mathcal{I}$ "
    using Unify.prem(2) unfolding constr_sem_c_def by metis+
  then obtain  $\sigma$  where  $\sigma$ : " $\vartheta \circ_s \delta \circ_s \sigma = \mathcal{I}$ " unfolding subst_compose_def by auto

  have  $\vartheta$ fun_id: "Fun f U ·  $\vartheta =$  Fun f U" "Fun f T ·  $\vartheta =$  Fun f T"
    using Unify.prem(1) trm_subst_ident[of "Fun f U"  $\vartheta$ ]
    fv_subset_if_in_strand_ik[of "Fun f U" S] Unify.hyps(2)
    fv_snd_rcv_strand_subset(2)[of S]
    strand_vars_split(1)[of S "Send (Fun f T)#S'"]
    unfolding wf_constr_def apply blast
    using Unify.prem(1) trm_subst_ident[of "Fun f T"  $\vartheta$ ]
    unfolding wf_constr_def by fastforce
  hence  $\vartheta\delta$ _disj:
    "subst_domain  $\vartheta \cap$  subst_domain  $\delta = \{ \}$ "
    "subst_domain  $\vartheta \cap$  range_vars  $\delta = \{ \}$ "
    "subst_domain  $\vartheta \cap$  range_vars  $\vartheta = \{ \}$ "
    using trm_subst_disj mgu_vars_bounded[OF Unify.hyps(3)[symmetric]] apply (blast,blast)
    using Unify.prem(1) unfolding wf_constr_def wf_subst_def by blast
  hence  $\vartheta\delta$ _support: " $\vartheta$  supports  $\mathcal{I}$ " " $\delta$  supports  $\mathcal{I}$ "
    by (simp_all add: subst_support_comp_split[OF " $(\vartheta \circ_s \delta)$  supports  $\mathcal{I}$ "])

  have "fv (Fun f T)  $\subseteq$  fvst (S@Send (Fun f T)#S')" "fv (Fun f U)  $\subseteq$  fvst (S@Send (Fun f T)#S')"
    using Unify.hyps(2) by force+
  hence  $\delta$ _vars_bound: "subst_domain  $\delta \cup$  range_vars  $\delta \subseteq$  fvst (S@Send (Fun f T)#S')"
    using mgu_vars_bounded[OF Unify.hyps(3)[symmetric]] by blast

  have "[ikst S ·set  $\mathcal{I}$ ; [Send (Fun f T)]]c  $\mathcal{I}$ "
  proof -
    from Unify.hyps(2) have "Fun f U ·  $\mathcal{I} \in$  ikst S ·set  $\mathcal{I}$ " by blast
    hence "Fun f U ·  $\mathcal{I} \in$  ikst S ·set  $\mathcal{I}$ " by blast
    moreover have "Unifier  $\delta$  (Fun f T) (Fun f U)"
      by (fact MGU_is_Unifier[OF mgu_gives_MGU[OF Unify.hyps(3)[symmetric]]])
    ultimately have "Fun f T ·  $\mathcal{I} \in$  ikst S ·set  $\mathcal{I}$ "
      using  $\sigma$  by (metis  $\vartheta$ fun_id subst_subst_compose)
    thus ?thesis by simp
  qed

  have "[{ }; S]c  $\mathcal{I}$ " "[ikst S ·set  $\mathcal{I}$ ; S']c  $\mathcal{I}$ "
  proof -
    have "(S@S' ·st  $\delta$ ) ·st  $\vartheta =$  S@S' ·st  $\delta$ " "(S@S') ·st  $\vartheta =$  S@S'"
    proof -
      have "subst_domain  $\vartheta \cap$  varsst (S@S') = { }"
        using Unify.prem(1) by (auto simp add: wf_constr_def)
      hence "subst_domain  $\vartheta \cap$  varsst (S@S' ·st  $\delta$ ) = { }"
        using  $\vartheta\delta$ _disj(2) strand_subst_vars_union_bound[of "S@S'"  $\delta$ ] by blast
      thus "(S@S' ·st  $\delta$ ) ·st  $\vartheta =$  S@S' ·st  $\delta$ " "(S@S') ·st  $\vartheta =$  S@S'"
        using strand_subst_comp (subst_domain  $\vartheta \cap$  varsst (S@S') = { }) by (blast,blast)
    qed
  qed

```



```

qed
moreover have "subst_idem  $\delta$ " by (fact mgu_gives_subst_idem[OF Unify.hyps(3)[symmetric]])
moreover have
  "(subst_domain  $\vartheta \cup \text{range\_vars } \vartheta) \cap \text{bvars}_{st} (S@S') = \{\}$ "
  "(subst_domain  $\vartheta \cup \text{range\_vars } \vartheta) \cap \text{bvars}_{st} (S@S' \cdot_{st} \delta) = \{\}$ "
  "(subst_domain  $\delta \cup \text{range\_vars } \delta) \cap \text{bvars}_{st} (S@S') = \{\}$ "
  using wf_constr_bvars_disj[OF Unify.prem(1)]
  wf_constr_bvars_disj'[OF Unify.prem(1)  $\delta\_vars\_bound$ ]
  by auto
ultimately have " $\llbracket \{\}; S@S' \rrbracket_c \mathcal{I}$ "
  using " $\llbracket \{\}; S@S' \cdot_{st} \delta \rrbracket_c \mathcal{I}$ "  $\sigma$ 
  strand_sem_subst(1)[of  $\vartheta$  " $S@S' \cdot_{st} \delta$ " " $\{\}$ " " $\delta \circ_s \sigma$ "]
  strand_sem_subst(2)[of  $\vartheta$  " $S@S'$ " " $\{\}$ " " $\delta \circ_s \sigma$ "]
  strand_sem_subst_subst_idem[of  $\delta$  " $S@S'$ " " $\{\}$ "  $\sigma$ ]
  unfolding constr_sem_c_def
  by (metis subst_compose_assoc)
thus " $\llbracket \{\}; S \rrbracket_c \mathcal{I}$ " " $\llbracket ik_{st} S \cdot_{set} \mathcal{I}; S' \rrbracket_c \mathcal{I}$ " by auto
qed

show " $\mathcal{I} \models_c \langle S@Send (Fun f T) \# S', \vartheta \rangle$ "
  using  $\vartheta\delta\_support(1)$  " $\llbracket ik_{st} S \cdot_{set} \mathcal{I}; [Send (Fun f T)] \rrbracket_c \mathcal{I}$ " " $\llbracket \{\}; S \rrbracket_c \mathcal{I}$ " " $\llbracket ik_{st} S \cdot_{set} \mathcal{I}; S' \rrbracket_c \mathcal{I}$ "
  by (auto simp add: constr_sem_c_def)
next
case (Equality  $S \delta t t' a S' \vartheta$ )
have " $(\vartheta \circ_s \delta)$  supports  $\mathcal{I}$ " " $\llbracket \{\}; S@S' \cdot_{st} \delta \rrbracket_c \mathcal{I}$ "
  using Equality.prem(2) unfolding constr_sem_c_def by metis+
then obtain  $\sigma$  where  $\sigma$ : " $\vartheta \circ_s \delta \circ_s \sigma = \mathcal{I}$ " unfolding subst_compose_def by auto

have " $fv\ t \subseteq vars_{st} (S@Equality\ a\ t\ t' \# S')$ " " $fv\ t' \subseteq vars_{st} (S@Equality\ a\ t\ t' \# S')$ "
  by auto
moreover have "subst_domain  $\vartheta \cap vars_{st} (S@Equality\ a\ t\ t' \# S') = \{\}$ "
  using Equality.prem(1) unfolding wf_constr_def by auto
ultimately have  $\vartheta fun\_id$ : " $t \cdot \vartheta = t'$ " " $t' \cdot \vartheta = t'$ "
  using trm_subst_ident[of  $t\ \vartheta$ ] trm_subst_ident[of  $t'\ \vartheta$ ]
  by auto
hence  $\vartheta\delta\_disj$ :
  "subst_domain  $\vartheta \cap \text{subst\_domain } \delta = \{\}$ "
  "subst_domain  $\vartheta \cap \text{range\_vars } \delta = \{\}$ "
  "subst_domain  $\vartheta \cap \text{range\_vars } \vartheta = \{\}$ "
  using trm_subst_disj mgu_vars_bounded[OF Equality.hyps(2)[symmetric]] apply (blast,blast)
  using Equality.prem(1) unfolding wf_constr_def wf_subst_def by blast
hence  $\vartheta\delta\_support$ : " $\vartheta$  supports  $\mathcal{I}$ " " $\delta$  supports  $\mathcal{I}$ "
  by (simp_all add: subst_support_comp_split[OF " $(\vartheta \circ_s \delta)$  supports  $\mathcal{I}$ "])

have " $fv\ t \subseteq fv_{st} (S@Equality\ a\ t\ t' \# S')$ " " $fv\ t' \subseteq fv_{st} (S@Equality\ a\ t\ t' \# S')$ " by auto
hence  $\delta\_vars\_bound$ : "subst_domain  $\delta \cup \text{range\_vars } \delta \subseteq fv_{st} (S@Equality\ a\ t\ t' \# S')$ "
  using mgu_vars_bounded[OF Equality.hyps(2)[symmetric]] by blast

have " $\llbracket ik_{st} S \cdot_{set} \mathcal{I}; [Equality\ a\ t\ t'] \rrbracket_c \mathcal{I}$ "
proof -
  have " $t \cdot \delta = t' \cdot \delta$ "
    using MGU_is_Unifier[OF mgu_gives_MGU[OF Equality.hyps(2)[symmetric]]]
    by metis
  hence " $t \cdot (\vartheta \circ_s \delta) = t' \cdot (\vartheta \circ_s \delta)$ " by (metis  $\vartheta fun\_id$  subst_subst_compose)
  hence " $t \cdot \mathcal{I} = t' \cdot \mathcal{I}$ " by (metis  $\sigma$  subst_subst_compose)
  thus ?thesis by simp
qed

have " $\llbracket \{\}; S \rrbracket_c \mathcal{I}$ " " $\llbracket ik_{st} S \cdot_{set} \mathcal{I}; S' \rrbracket_c \mathcal{I}$ "
proof -
  have " $(S@S' \cdot_{st} \delta) \cdot_{st} \vartheta = S@S' \cdot_{st} \delta$ " " $(S@S') \cdot_{st} \vartheta = S@S'$ "
  proof -
    have "subst_domain  $\vartheta \cap vars_{st} (S@S') = \{\}$ "

```

```

    using Equality.premis(1)
    by (fastforce simp add: wf_constr_def simp del: subst_range.simps)
  hence "subst_domain  $\vartheta \cap \text{fv}_{st} (S@S') = \{\}$ " by blast
  hence "subst_domain  $\vartheta \cap \text{fv}_{st} (S@S' \cdot_{st} \delta) = \{\}$ "
    using  $\vartheta\delta\text{-disj}(2)$  subst_sends_strand_fv_to_img[of " $S@S'$ "  $\delta$ ] by blast
  thus " $(S@S' \cdot_{st} \delta) \cdot_{st} \vartheta = S@S' \cdot_{st} \delta$ " " $(S@S') \cdot_{st} \vartheta = S@S'$ "
    using strand_subst_comp (subst_domain  $\vartheta \cap \text{vars}_{st} (S@S') = \{\}$ ) by (blast,blast)
qed
moreover have
  "(subst_domain  $\vartheta \cup \text{range\_vars } \vartheta) \cap \text{bvars}_{st} (S@S') = \{\}$ "
  "(subst_domain  $\vartheta \cup \text{range\_vars } \vartheta) \cap \text{bvars}_{st} (S@S' \cdot_{st} \delta) = \{\}$ "
  "(subst_domain  $\delta \cup \text{range\_vars } \delta) \cap \text{bvars}_{st} (S@S') = \{\}$ "
  using wf_constr_bvars_disj[OF Equality.premis(1)]
    wf_constr_bvars_disj'[OF Equality.premis(1)  $\delta\text{-vars\_bound}$ ]
  by auto
ultimately have " $\llbracket \{\}; S@S' \rrbracket_c \mathcal{I}$ "
  using " $\llbracket \{\}; S@S' \cdot_{st} \delta \rrbracket_c \mathcal{I}$ "  $\sigma$ 
    strand_sem_subst(1)[of  $\vartheta$  " $S@S' \cdot_{st} \delta$ " " $\{\}$ " " $\delta \circ_s \sigma$ "]
    strand_sem_subst(2)[of  $\vartheta$  " $S@S'$ " " $\{\}$ " " $\delta \circ_s \sigma$ "]
    strand_sem_subst_subst_idem[of  $\delta$  " $S@S'$ " " $\{\}$ "  $\sigma$ ]
    mgu_gives_subst_idem[OF Equality.hyps(2) [symmetric]]
  unfolding constr_sem_c_def
  by (metis subst_compose_assoc)
thus " $\llbracket \{\}; S \rrbracket_c \mathcal{I}$ " " $\llbracket \text{ik}_{st} S \cdot_{set} \mathcal{I}; S' \rrbracket_c \mathcal{I}$ " by auto
qed

show " $\mathcal{I} \models_c \langle S@Equality \text{ a t t' \# } S', \vartheta \rangle$ "
  using  $\vartheta\delta\text{-support}(1)$  " $\llbracket \text{ik}_{st} S \cdot_{set} \mathcal{I}; [Equality \text{ a t t'}] \rrbracket_c \mathcal{I}$ " " $\llbracket \{\}; S \rrbracket_c \mathcal{I}$ " " $\llbracket \text{ik}_{st} S \cdot_{set} \mathcal{I}; S' \rrbracket_c \mathcal{I}$ "
  by (auto simp add: constr_sem_c_def)
qed

theorem LI_soundness:
  assumes "wf_constr  $S_1 \vartheta_1$ " " $(S_1, \vartheta_1) \rightsquigarrow^* (S_2, \vartheta_2)$ " " $\mathcal{I} \models_c \langle S_2, \vartheta_2 \rangle$ "
  shows " $\mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle$ "
  using assms(2,1,3)
  proof (induction  $S_2 \vartheta_2$  rule: rtrancl_induct2)
    case (step  $S_i \vartheta_i S_j \vartheta_j$ ) thus ?case
      using LI_preserves_wellformedness[OF " $(S_1, \vartheta_1) \rightsquigarrow^* (S_i, \vartheta_i)$ " (wf_constr  $S_1 \vartheta_1$ )]
        LI_soundness_single[OF " $(S_i, \vartheta_i) \rightsquigarrow (S_j, \vartheta_j)$ " " $\mathcal{I} \models_c \langle S_j, \vartheta_j \rangle$ "]
        step.IH[OF (wf_constr  $S_1 \vartheta_1$ )]
      by metis
  qed metis
end

```

### 3.2.5 Theorem: Completeness of the Lazy Intruder

```

context
begin
private lemma LI_completeness_single:
  assumes "wf_constr  $S_1 \vartheta_1$ " " $\mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle$ " " $\neg \text{simple } S_1$ "
  shows " $\exists S_2 \vartheta_2. (S_1, \vartheta_1) \rightsquigarrow (S_2, \vartheta_2) \wedge (\mathcal{I} \models_c \langle S_2, \vartheta_2 \rangle)$ "
  using not_simple_elim[OF " $\neg \text{simple } S_1$ "]
  proof -
    { — In this case  $S_1$  isn't simple because it contains an equality constraint, so we can simply proceed with the reduction
    by computing the MGU for the equation
      assume " $\exists S' S'' \text{ a t t'}. S_1 = S'@Equality \text{ a t t' \# } S'' \wedge \text{simple } S''$ "
      then obtain  $S \text{ a t t' } S'$  where  $S_1: "S_1 = S@Equality \text{ a t t' \# } S"$  "simple  $S$ " by moura
      hence *: "wf_{st}  $\{\}$   $S$ " " $\mathcal{I} \models_c \langle S, \vartheta_1 \rangle$ " " $\vartheta_1$  supports  $\mathcal{I}$ " " $t \cdot \mathcal{I} = t' \cdot \mathcal{I}$ "
        using " $\mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle$ " (wf_constr  $S_1 \vartheta_1$ ) wf_eq_fv[of " $\{\}$ "  $S \text{ t t' } S'$ ]
          fv_snd_rcv_strand_subset(5)[of  $S$ ]
        by (auto simp add: constr_sem_c_def wf_constr_def)

      from * have "Unifier  $\mathcal{I} \text{ t t'}$ " by simp
    }
  }

```

then obtain  $\delta$  where  $\delta$ :

```
"Some  $\delta = \text{mgu } t \ t'$ " "subst_idem  $\delta$ " "subst_domain  $\delta \cup \text{range\_vars } \delta \subseteq \text{fv } t \cup \text{fv } t'$ "
using mgu_always_unifies mgu_gives_subst_idem mgu_vars_bounded by metis+
```

have " $\delta \preceq_o \mathcal{I}$ "

```
using mgu_gives_MGU[OF  $\delta(1)$  [symmetric]]
by (metis (Unifier  $\mathcal{I} \ t \ t'$ ))
```

hence " $\delta$  supports  $\mathcal{I}$ " using subst\_support\_if\_mgt\_subst\_idem[OF  $\delta(2)$ ] by metis

hence " $(\vartheta_1 \circ_s \delta)$  supports  $\mathcal{I}$ " using subst\_support\_comp  $\langle \vartheta_1 \text{ supports } \mathcal{I} \rangle$  by metis

have " $\llbracket \{ \}; S@S' \cdot_{st} \delta \rrbracket_c \mathcal{I}$ "

proof -

```
have "subst_domain  $\delta \cup \text{range\_vars } \delta \subseteq \text{fv}_{st} S_1$ " using  $\delta(3) \ S_1(1)$  by auto
```

```
hence " $\llbracket \{ \}; S_1 \cdot_{st} \delta \rrbracket_c \mathcal{I}$ "
```

```
using (subst_idem  $\delta$ )  $\langle \delta \preceq_o \mathcal{I} \rangle \langle \mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \rangle \text{strand\_sem\_subst}$ 
wf_constr_bvars_disj'(1) [OF assms(1)]
```

```
unfolding subst_idem_def constr_sem_c_def
```

```
by (metis (no_types) subst_compose_assoc)
```

```
thus " $\llbracket \{ \}; S@S' \cdot_{st} \delta \rrbracket_c \mathcal{I}$ " using  $S_1(1)$  by force
```

qed

moreover have " $(S@Equality \ a \ t \ t' \# S', \vartheta_1) \rightsquigarrow (S@S' \cdot_{st} \delta, \vartheta_1 \circ_s \delta)$ "

```
using LI_rel.Equality[OF (simple  $S$ )  $\delta(1)$ ]  $S_1$  by metis
```

ultimately have ?thesis

```
using  $S_1(1) \langle (\vartheta_1 \circ_s \delta) \text{ supports } \mathcal{I} \rangle$ 
```

```
by (auto simp add: constr_sem_c_def)
```

} moreover {

— In this case  $S_1$  isn't simple because it contains a deduction constraint for a composed term, so we must look at how this composed term is derived under the interpretation  $\mathcal{I}$

```
assume " $\exists S' \ S'' \ f \ T. \ S_1 = S'@Send \ (\text{Fun } f \ T) \# S'' \wedge \text{simple } S''$ "
```

with assms obtain  $S \ f \ T \ S'$  where  $S_1$ : " $S_1 = S@Send \ (\text{Fun } f \ T) \# S'$ " "simple  $S$ " by moura

hence " $\text{wf}_{st} \{ \} \ S$ " " $\mathcal{I} \models_c \langle S, \vartheta_1 \rangle$ " " $\vartheta_1$  supports  $\mathcal{I}$ "

```
using  $\langle \mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \rangle \langle \text{wf}_{constr} \ S_1 \ \vartheta_1 \rangle$ 
```

```
by (auto simp add: constr_sem_c_def wf_constr_def)
```

— Lemma for a common subcase

have fun\_sat: " $\mathcal{I} \models_c \langle S@(\text{map } Send \ T)@S', \vartheta_1 \rangle$ " when  $T$ : " $\bigwedge t. t \in \text{set } T \implies \text{ik}_{st} \ S \cdot_{set} \mathcal{I} \vdash_c t \cdot \mathcal{I}$ "

proof -

```
have " $\bigwedge t. t \in \text{set } T \implies \llbracket \text{ik}_{st} \ S \cdot_{set} \mathcal{I}; [\text{Send } t] \rrbracket_c \mathcal{I}$ " using  $T$  by simp
```

```
hence " $\llbracket \text{ik}_{st} \ S \cdot_{set} \mathcal{I}; \text{map } Send \ T \rrbracket_c \mathcal{I}$ " using  $\langle \mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \rangle \text{strand\_sem\_Send\_map}$  by metis
```

```
moreover have " $\llbracket \text{ik}_{st} \ (S@(\text{map } Send \ T)) \cdot_{set} \mathcal{I}; S' \rrbracket_c \mathcal{I}$ "
```

```
using  $\langle \mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \rangle \ S_1$ 
```

```
by (auto simp add: constr_sem_c_def)
```

ultimately show ?thesis

```
using  $\langle \mathcal{I} \models_c \langle S, \vartheta_1 \rangle \rangle \langle \mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \rangle$ 
```

```
by (force simp add: constr_sem_c_def)
```

qed

from  $S_1 \ \langle \mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \rangle$  have " $\text{ik}_{st} \ S \cdot_{set} \mathcal{I} \vdash_c \text{Fun } f \ T \cdot \mathcal{I}$ " by (auto simp add: constr\_sem\_c\_def)

hence ?thesis

proof cases

— Case 1:  $\mathcal{I}(f(T))$  has been derived using the AxiomC rule.

case AxiomC

hence ex\_t: " $\exists t. t \in \text{ik}_{st} \ S \wedge \text{Fun } f \ T \cdot \mathcal{I} = t \cdot \mathcal{I}$ " by auto

show ?thesis

proof (cases " $\forall T'. \text{Fun } f \ T' \in \text{ik}_{st} \ S \implies \text{Fun } f \ T \cdot \mathcal{I} \neq \text{Fun } f \ T' \cdot \mathcal{I}$ ")

— Case 1.1:  $f(T)$  is equal to a variable in the intruder knowledge under  $\mathcal{I}$ . Hence there must exists a deduction constraint in the simple prefix of the constraint in which this variable occurs/"is sent" for the first time. Since this variable itself cannot have been derived from the AxiomC rule (because it must be equal under the interpretation to  $f(T)$ , which is by assumption not in the intruder knowledge under  $\mathcal{I}$ ) it must be the case that we can derive it using the ComposeC rule. Hence we can apply the Compose rule of the lazy intruder to  $f(T)$ .

case True

have " $\exists v. \text{Var } v \in \text{ik}_{st} \ S \wedge \text{Fun } f \ T \cdot \mathcal{I} = \mathcal{I} \ v$ "

proof -

```

obtain t where "t ∈ ikst S" "Fun f T · I = t · I" using ex_t by moura
thus ?thesis
  using (∀T'. Fun f T' ∈ ikst S → Fun f T · I ≠ Fun f T' · I)
  by (cases t) auto
qed
hence "∃v ∈ wfrestrictedvarsst S. Fun f T · I = I v"
  using vars_subset_if_in_strand_ik2[of _ S] by fastforce
then obtain v Spre Ssuf
  where S: "S = Spre@Send (Var v)#Ssuf" "Fun f T · I = I v"
  "¬(∃w ∈ wfrestrictedvarsst Spre. Fun f T · I = I w)"
  using (wfst {} S) wf_simple_strand_first_Send_var_split[OF _ ⟨simple S⟩, of "Fun f T" I]
  by auto
hence "∀w. Var w ∈ ikst Spre → I v ≠ Var w · I" by auto
moreover have "∀T'. Fun f T' ∈ ikst Spre → Fun f T · I ≠ Fun f T' · I"
  using (∀T'. Fun f T' ∈ ikst S → Fun f T · I ≠ Fun f T' · I) S(1)
  by (meson contra_subsetD ik_append_subset(1))
hence "∀g T'. Fun g T' ∈ ikst Spre → I v ≠ Fun g T' · I" using S(2) by simp
ultimately have "∀t ∈ ikst Spre. I v ≠ t · I" by (metis term.exhaust)
hence "I v ∉ (ikst Spre) ·set I" by auto

```

```

have "ikst Spre ·set I ⊢c I v"
  using S1(1) S(1) ⟨I ⊢c ⟨S1, ϑ1⟩⟩
  by (auto simp add: constr_sem_c_def)
hence "ikst Spre ·set I ⊢c Fun f T · I" using (Fun f T · I = I v) by metis
hence "length T = arity f" "public f" "∧t. t ∈ set T ⇒ ikst Spre ·set I ⊢c t · I"
  using (Fun f T · I = I v) ⟨I v ∉ ikst Spre ·set I⟩
  intruder_synth.simps[of "ikst Spre ·set I" "I v"]
  by auto
hence *: "∧t. t ∈ set T ⇒ ikst S ·set I ⊢c t · I"
  using S(1) by (auto intro: ideduct_synth_mono)
hence "I ⊢c ⟨S@map Send T@S', ϑ1⟩" by (metis fun_sat)
moreover have "(S@Send (Fun f T)#S', ϑ1) ∼ (S@map Send T@S', ϑ1)"
  by (metis LI_rel.Compose[OF ⟨simple S⟩ ⟨length T = arity f⟩ ⟨public f⟩])
ultimately show ?thesis using S1 by auto

```

next

— Case 1.2:  $\mathcal{I}(f(T))$  can be derived from an interpreted composed term in the intruder knowledge. Use the *Unify* rule on this composed term to further reduce the constraint.

```

case False
then obtain T' where t: "Fun f T' ∈ ikst S" "Fun f T · I = Fun f T' · I"
  by auto
hence "fv (Fun f T') ⊆ fvst S1"
  using S1(1) fv_subset_if_in_strand_ik'[OF t(1)]
  fv_snd_rcv_strand_subset(2)[of S]
  by auto
from t have "Unifier I (Fun f T) (Fun f T')" by simp
then obtain δ where δ:
  "Some δ = mgu (Fun f T) (Fun f T'" "subst_idem δ"
  "subst_domain δ ∪ range_vars δ ⊆ fv (Fun f T) ∪ fv (Fun f T'"
  using mgu_always_unifies mgu_gives_subst_idem mgu_vars_bounded by metis+

have "δ ≤o I"
  using mgu_gives_MGU[OF δ(1)[symmetric]]
  by (metis ⟨Unifier I (Fun f T) (Fun f T')⟩)
hence "δ supports I" using subst_support_if_mgt_subst_idem[OF _ δ(2)] by metis
hence "(ϑ1 ∘s δ) supports I" using subst_support_comp ⟨ϑ1 supports I⟩ by metis

have "⌊{ }; S@S' ·st δ⌋c I"
proof -
  have "subst_domain δ ∪ range_vars δ ⊆ fvst S1"
    using δ(3) S1(1) ⟨fv (Fun f T') ⊆ fvst S1⟩
    unfolding range_vars_alt_def by (fastforce simp del: subst_range.simps)
  hence "⌊{ }; S1 ·st δ⌋c I"
    using ⟨subst_idem δ⟩ ⟨δ ≤o I⟩ ⟨I ⊢c ⟨S1, ϑ1⟩⟩ strand_sem_subst

```

```

      wf_constr_bvars_disj'(1)[OF assms(1)]
    unfolding subst_idem_def constr_sem_c_def
    by (metis (no_types) subst_compose_assoc)
  thus "{S'; S@S' ·st δ}_c I" using S1(1) by force
qed
moreover have "(S@Send (Fun f T)#S', ϑ1) ~ (S@S' ·st δ, ϑ1 ∘s δ)"
  using LI_rel.Unify[OF ⟨simple S⟩ t(1) δ(1)] S1 by metis
ultimately show ?thesis
  using S1(1) ⟨(ϑ1 ∘s δ) supports I⟩
  by (auto simp add: constr_sem_c_def)
qed
next
  — Case 2:  $\mathcal{I}(f(T))$  has been derived using the ComposeC rule. Simply use the Compose rule of the lazy intruder
  to proceed with the reduction.
  case (ComposeC T' g)
  hence "f = g" "length T = arity f" "public f"
    and "∧x. x ∈ set T ⇒ ikst S ·set I ⊢c x · I"
    by auto
  hence "I ⊢c ⟨S@⟨map Send T⟩@S', ϑ1⟩" using fun_sat by metis
  moreover have "(S1, ϑ1) ~ (S@⟨map Send T⟩@S', ϑ1)"
    using S1 LI_rel.Compose[OF ⟨simple S⟩ ⟨length T = arity f⟩ ⟨public f⟩]
    by metis
  ultimately show ?thesis by metis
qed
} moreover have "∧A B X F. S1 = A@Inequality X F#B ⇒ ineq_model I X F"
  using assms(2) by (auto simp add: constr_sem_c_def)
ultimately show ?thesis using not_simple_elim[OF ⟨¬simple S1⟩] by metis
qed

theorem LI_completeness:
  assumes "wf_constr S1 ϑ1" "I ⊢c ⟨S1, ϑ1⟩"
  shows "∃S2 ϑ2. (S1, ϑ1) ~* (S2, ϑ2) ∧ simple S2 ∧ (I ⊢c ⟨S2, ϑ2⟩)"
proof (cases "simple S1")
  case False
  let ?Stuck = "λS2 ϑ2. ¬(∃S3 ϑ3. (S2, ϑ2) ~ (S3, ϑ3) ∧ (I ⊢c ⟨S3, ϑ3⟩))"
  let ?Sats = "{((S, ϑ), (S', ϑ')). (S, ϑ) ~ (S', ϑ') ∧ (I ⊢c ⟨S, ϑ⟩) ∧ (I ⊢c ⟨S', ϑ'⟩)}"

  have simple_if_stuck:
    "∧S2 ϑ2. [(S1, ϑ1) ~+ (S2, ϑ2); I ⊢c ⟨S2, ϑ2⟩; ?Stuck S2 ϑ2] ⇒ simple S2"
    using LI_completeness_single
      LI_preserves_wellformedness
      ⟨wf_constr S1 ϑ1⟩
      trancl_into_rtrancl
    by metis

  have base: "∃b. ((S1, ϑ1), b) ∈ ?Sats"
    using LI_completeness_single[OF assms False] assms(2)
    by auto

  have *: "∧S ϑ S' ϑ'. ((S, ϑ), (S', ϑ')) ∈ ?Sats+ ⇒ (S, ϑ) ~+ (S', ϑ') ∧ (I ⊢c ⟨S', ϑ'⟩)"
  proof -
    fix S ϑ S' ϑ'
    assume "((S, ϑ), (S', ϑ')) ∈ ?Sats+"
    thus "(S, ϑ) ~+ (S', ϑ') ∧ (I ⊢c ⟨S', ϑ'⟩)"
      by (induct rule: trancl_induct2) auto
  qed

  have "∃S2 ϑ2. ((S1, ϑ1), (S2, ϑ2)) ∈ ?Sats+ ∧ ?Stuck S2 ϑ2"
  proof (rule ccontr)
    assume "¬(∃S2 ϑ2. ((S1, ϑ1), (S2, ϑ2)) ∈ ?Sats+ ∧ ?Stuck S2 ϑ2)"
    hence sat_not_stuck: "∧S2 ϑ2. ((S1, ϑ1), (S2, ϑ2)) ∈ ?Sats+ ⇒ ¬?Stuck S2 ϑ2" by blast

    have "∀S ϑ. ((S1, ϑ1), (S, ϑ)) ∈ ?Sats+ ⇒ (∃b. ((S, ϑ), b) ∈ ?Sats)"

```

```

proof (intro allI impI)
  fix S  $\vartheta$  assume a: " $((S_1, \vartheta_1), (S, \vartheta)) \in ?Sats^+$ "
  have " $\bigwedge b. ((S_1, \vartheta_1), b) \in ?Sats^+ \implies \exists c. b \rightsquigarrow c \wedge ((S_1, \vartheta_1), c) \in ?Sats^+$ "
  proof -
    fix b assume in_sat: " $((S_1, \vartheta_1), b) \in ?Sats^+$ "
    hence " $\exists c. (b, c) \in ?Sats$ " using * sat_not_stuck by (cases b) blast
    thus " $\exists c. b \rightsquigarrow c \wedge ((S_1, \vartheta_1), c) \in ?Sats^+$ "
      using trancl_into_trancl[OF in_sat] by blast
  qed
  hence " $\exists S' \vartheta'. (S, \vartheta) \rightsquigarrow (S', \vartheta') \wedge ((S_1, \vartheta_1), (S', \vartheta')) \in ?Sats^+$ " using a by auto
  then obtain S'  $\vartheta'$  where S'  $\vartheta'$ : " $(S, \vartheta) \rightsquigarrow (S', \vartheta')$ " " $((S_1, \vartheta_1), (S', \vartheta')) \in ?Sats^+$ " by auto
  hence " $\mathcal{I} \models_c \langle S', \vartheta' \rangle$ " using * by blast
  moreover have " $(S_1, \vartheta_1) \rightsquigarrow^+ (S, \vartheta)$ " using a trancl_mono by blast
  ultimately have " $((S, \vartheta), (S', \vartheta')) \in ?Sats$ " using S'  $\vartheta'$ (1) * a by blast
  thus " $\exists b. ((S, \vartheta), b) \in ?Sats$ " using S'  $\vartheta'$ (2) by blast
qed
hence " $\exists f. \forall i::nat. (f\ i, f\ (Suc\ i)) \in ?Sats$ "
  using infinite_chain_intro'[OF base] by blast
moreover have " $?Sats \subseteq LI\_rel^+$ " by auto
hence " $\neg(\exists f. \forall i::nat. (f\ i, f\ (Suc\ i)) \in ?Sats)$ "
  using LI_no_infinite_chain infinite_chain_mono by blast
ultimately show False by auto
qed
hence " $\exists S_2 \vartheta_2. (S_1, \vartheta_1) \rightsquigarrow^+ (S_2, \vartheta_2) \wedge simple\ S_2 \wedge (\mathcal{I} \models_c \langle S_2, \vartheta_2 \rangle)$ "
  using simple_if_stuck * by blast
thus ?thesis by (meson trancl_into_rtrancl)
qed (blast intro:  $\langle \mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \rangle$ )
end

```

### 3.2.6 Corollary: Soundness and Completeness as a Single Theorem

```

corollary LI_soundness_and_completeness:
  assumes "wf_constr S1  $\vartheta_1$ "
  shows " $\mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle \iff (\exists S_2 \vartheta_2. (S_1, \vartheta_1) \rightsquigarrow^* (S_2, \vartheta_2) \wedge simple\ S_2 \wedge (\mathcal{I} \models_c \langle S_2, \vartheta_2 \rangle))$ "
by (metis LI_soundness[OF assms] LI_completeness[OF assms])

end

end

```

## 3.3 The Typed Model (Typed\_Model)

```

theory Typed_Model
imports Lazy_Intruder
begin

```

Term types

```
type_synonym ('f, 'v) term_type = "('f, 'v) term"
```

Constructors for term types

```
abbreviation (input) TAtom::"'v  $\Rightarrow$  ('f, 'v) term_type" where
  "TAtom a  $\equiv$  Var a"
```

```
abbreviation (input) TComp::"['f, ('f, 'v) term_type list]  $\Rightarrow$  ('f, 'v) term_type" where
  "TComp f T  $\equiv$  Fun f T"
```

The typed model extends the intruder model with a typing function  $\Gamma$  that assigns types to terms.

```

locale typed_model = intruder_model arity public Ana
  for arity::"'fun  $\Rightarrow$  nat"
  and public::"'fun  $\Rightarrow$  bool"
  and Ana::"('fun, 'var) term  $\Rightarrow$  (('fun, 'var) term list  $\times$  ('fun, 'var) term list)"
+

```

```

fixes  $\Gamma :: ('fun, 'var) term \Rightarrow ('fun, 'atom :: finite) term\_type$ 
assumes const_type: " $\bigwedge c. \text{arity } c = 0 \implies \exists a. \forall T. \Gamma (\text{Fun } c \ T) = T\text{Atom } a$ "
    and fun_type: " $\bigwedge f \ T. \text{arity } f > 0 \implies \Gamma (\text{Fun } f \ T) = T\text{Comp } f \ (\text{map } \Gamma \ T)$ "
    and infinite_typed_consts: " $\bigwedge a. \text{infinite } \{c. \Gamma (\text{Fun } c \ []) = T\text{Atom } a \wedge \text{public } c\}$ "
    and  $\Gamma\_wf$ : " $\bigwedge t \ f \ T. T\text{Comp } f \ T \sqsubseteq \Gamma \ t \implies \text{arity } f > 0$ "
    " $\bigwedge x. wf_{trm} (\Gamma (\text{Var } x))$ "
    and no_private_funs[simp]: " $\bigwedge f. \text{arity } f > 0 \implies \text{public } f$ "
begin

```

### 3.3.1 Definitions

The set of atomic types

abbreviation " $\mathcal{T}_a \equiv UNIV :: ('atom \ set)$ "

Well-typed substitutions

definition  $wt_{subst}$  where

" $wt_{subst} \ \sigma \equiv (\forall v. \Gamma (\text{Var } v) = \Gamma (\sigma \ v))$ "

The set of sub-message patterns (SMP)

inductive\_set  $SMP :: ('fun, 'var) terms \Rightarrow ('fun, 'var) terms$  for  $M$  where

```

MP[intro]: " $t \in M \implies t \in SMP \ M$ "
| Subterm[intro]: " $\llbracket t \in SMP \ M; t' \sqsubseteq t \rrbracket \implies t' \in SMP \ M$ "
| Substitution[intro]: " $\llbracket t \in SMP \ M; wt_{subst} \ \delta; wf_{trms} \ (\text{subst\_range } \delta) \rrbracket \implies (t \cdot \delta) \in SMP \ M$ "
| Ana[intro]: " $\llbracket t \in SMP \ M; Ana \ t = (K, T); k \in \text{set } K \rrbracket \implies k \in SMP \ M$ "

```

Type-flaw resistance for sets: Unifiable sub-message patterns must have the same type (unless they are variables)

definition  $tfr_{set}$  where

" $tfr_{set} \ M \equiv (\forall s \in SMP \ M - (\text{Var } \mathcal{V}). \forall t \in SMP \ M - (\text{Var } \mathcal{V}). (\exists \delta. \text{Unifier } \delta \ s \ t) \longrightarrow \Gamma \ s = \Gamma \ t)$ "

Type-flaw resistance for strand steps: - The terms in a satisfiable equality step must have the same types - Inequality steps must satisfy the conditions of the "inequality lemma"

fun  $tfr_{stp}$  where

```

"tfr_{stp} (Equality a t t') = ((\exists \delta. \text{Unifier } \delta \ t \ t') \longrightarrow \Gamma \ t = \Gamma \ t')"
| "tfr_{stp} (Inequality X F) = (
    (\forall x \in fv_{pairs} \ F - \text{set } X. \exists a. \Gamma (\text{Var } x) = T\text{Atom } a) \vee
    (\forall f \ T. \text{Fun } f \ T \in \text{subterms}_{set} \ (\text{trms}_{pairs} \ F) \longrightarrow T = [] \vee (\exists s \in \text{set } T. s \notin \text{Var } \text{'set } X)))"
| "tfr_{stp} _ = True"

```

Type-flaw resistance for strands: - The set of terms in strands must be type-flaw resistant - The steps of strands must be type-flaw resistant

definition  $tfr_{st}$  where

" $tfr_{st} \ S \equiv tfr_{set} \ (\text{trms}_{st} \ S) \wedge \text{list\_all } tfr_{stp} \ S$ "

### 3.3.2 Small Lemmata

lemma  $tfr_{stp\_list\_all\_alt\_def}$ :

```

"list_all tfr_{stp} S \longleftrightarrow
    ((\forall a \ t \ t'. \text{Equality } a \ t \ t' \in \text{set } S \wedge (\exists \delta. \text{Unifier } \delta \ t \ t') \longrightarrow \Gamma \ t = \Gamma \ t') \wedge
    (\forall X \ F. \text{Inequality } X \ F \in \text{set } S \longrightarrow
        (\forall x \in fv_{pairs} \ F - \text{set } X. \exists a. \Gamma (\text{Var } x) = T\text{Atom } a)
        \vee (\forall f \ T. \text{Fun } f \ T \in \text{subterms}_{set} \ (\text{trms}_{pairs} \ F) \longrightarrow T = [] \vee (\exists s \in \text{set } T. s \notin \text{Var } \text{'set } X))))"
(is "?P S \longleftrightarrow ?Q S")

```

proof

show " $?P \ S \implies ?Q \ S$ "

proof (induction S)

case (Cons x S) thus ?case by (cases x) auto

qed simp

show " $?Q \ S \implies ?P \ S$ "

proof (induction S)

```

    case (Cons x S) thus ?case by (cases x) auto
  qed simp
qed

```

```

lemma  $\Gamma\_wf'$ : " $wf_{trm} \ t \implies wf_{trm} \ (\Gamma \ t)$ "
proof (induction t)
  case (Fun f T)
  hence *: " $arity \ f = length \ T \ \wedge \ t \in set \ T \implies wf_{trm} \ (\Gamma \ t)$ " unfolding  $wf_{trm\_def}$  by auto
  { assume " $arity \ f = 0$ " hence ?case using const_type[of f] by auto }
  moreover
  { assume " $arity \ f > 0$ " hence ?case using fun_type[of f] * by force }
  ultimately show ?case by auto
qed (metis  $\Gamma\_wf(2)$ )

```

```

lemma fun_type_inv: assumes " $\Gamma \ t = TComp \ f \ T$ " shows " $arity \ f > 0$ " "public f"
using  $\Gamma\_wf(1)[of \ f \ T \ t]$  assms by simp_all

```

```

lemma fun_type_inv_wf: assumes " $\Gamma \ t = TComp \ f \ T$ " " $wf_{trm} \ t$ " shows " $arity \ f = length \ T$ "
using  $\Gamma\_wf'[OF \ assms(2)] \ assms(1)$  unfolding  $wf_{trm\_def}$  by auto

```

```

lemma const_type_inv: " $\Gamma \ (Fun \ c \ X) = TAtom \ a \implies arity \ c = 0$ "
by (rule ccontr, simp add: fun_type)

```

```

lemma const_type_inv_wf: assumes " $\Gamma \ (Fun \ c \ X) = TAtom \ a$ " and " $wf_{trm} \ (Fun \ c \ X)$ " shows " $X = []$ "
by (metis assms const_type_inv length_0_conv subtermeqI'  $wf_{trm\_def}$ )

```

```

lemma const_type': " $\forall c \in \mathcal{C}. \exists a \in \mathfrak{T}_a. \forall X. \Gamma \ (Fun \ c \ X) = TAtom \ a$ " using const_type by simp
lemma fun_type': " $\forall f \in \Sigma_f. \forall X. \Gamma \ (Fun \ f \ X) = TComp \ f \ (map \ \Gamma \ X)$ " using fun_type by simp

```

```

lemma infinite_public_consts[simp]: " $infinite \ \{c. \text{public } c \wedge arity \ c = 0\}$ "

```

```

proof -
  fix a::'atom
  define A where "A  $\equiv \{c. \Gamma \ (Fun \ c \ []) = TAtom \ a \wedge \text{public } c\}$ "
  define B where "B  $\equiv \{c. \text{public } c \wedge arity \ c = 0\}$ "

  have "arity c = 0" when c: "c  $\in A$ " for c
  using c const_type_inv unfolding A_def by blast
  hence "A  $\subseteq B$ " unfolding A_def B_def by blast
  hence "infinite B"
  using infinite_typed_consts[of a, unfolded A_def[symmetric]]
  by (metis infinite_super)
  thus ?thesis unfolding B_def by blast
qed

```

```

lemma infinite_fun_syms[simp]:
  " $infinite \ \{c. \text{public } c \wedge arity \ c > 0\} \implies infinite \ \Sigma_f$ "
  " $infinite \ \mathcal{C}$ " " $infinite \ C_{pub}$ " " $infinite \ (UNIV::'fun \ set)$ "
by (metis  $\Sigma_f\_unfold$  finite_Collect_conjI,
    metis infinite_public_consts finite_Collect_conjI,
    use infinite_public_consts Cpub_unfold in (force simp add: Collect_conj_eq),
    metis UNIV_I finite_subset subsetI infinite_public_consts(1))

```

```

lemma id_univ_proper_subset[simp]: " $\Sigma_f \subset UNIV$ " " $(\exists f. arity \ f > 0) \implies \mathcal{C} \subset UNIV$ "
by (metis finite.emptyI inf_top.right_neutral top.not_eq_extremum disjoint_fun_syms
    infinite_fun_syms(2) inf_commute)
(metis top.not_eq_extremum UNIV_I const_arity_eq_zero less_irrefl)

```

```

lemma exists_fun_notin_funs_term: " $\exists f::'fun. f \notin \text{funs\_term } t$ "
by (metis UNIV_eq_I finite_fun_symbols infinite_fun_syms(4))

```

```

lemma exists_fun_notin_funs_terms:
  assumes "finite M" shows " $\exists f::'fun. f \notin \bigcup (\text{funs\_term } ` M)$ "

```



```

by (metis assms finite_fun_symbols infinite_fun_syms(4) ex_new_if_finite finite_UN)

lemma exists_notin_funsst: "∃f. f ∉ funsst (S::('fun,'var) strand)"
by (metis UNIV_eq_I finite_funsst infinite_fun_syms(4))

lemma infinite_typed_consts': "infinite {c. Γ (Fun c []) = TAtom a ∧ public c ∧ arity c = 0}"
proof -
  { fix c assume "Γ (Fun c []) = TAtom a" "public c"
    hence "arity c = 0" using const_type[of c] fun_type[of c "[]"] by auto
  } hence "{c. Γ (Fun c []) = TAtom a ∧ public c ∧ arity c = 0} =
    {c. Γ (Fun c []) = TAtom a ∧ public c}"
  by auto
  thus "infinite {c. Γ (Fun c []) = TAtom a ∧ public c ∧ arity c = 0}"
  using infinite_typed_consts[of a] by metis
qed

lemma atypes_inhabited: "∃c. Γ (Fun c []) = TAtom a ∧ wftrm (Fun c []) ∧ public c ∧ arity c = 0"
proof -
  obtain c where "Γ (Fun c []) = TAtom a" "public c" "arity c = 0"
  using infinite_typed_consts'(1)[of a] not_finite_existsD by blast
  thus ?thesis using const_type_inv[OF (Γ (Fun c []) = TAtom a)] unfolding wftrm_def by auto
qed

lemma atype_ground_term_ex: "∃t. fv t = {} ∧ Γ t = TAtom a ∧ wftrm t"
using atypes_inhabited[of a] by force

lemma fun_type_id_eq: "Γ (Fun f X) = TComp g Y ⟹ f = g"
by (metis const_type fun_type neq0_conv "term.inject"(2) "term.simps"(4))

lemma fun_type_length_eq: "Γ (Fun f X) = TComp g Y ⟹ length X = length Y"
by (metis fun_type fun_type_id_eq fun_type_inv(1) length_map term.inject(2))

lemma type_ground_inhabited: "∃t'. fv t' = {} ∧ Γ t = Γ t'"
proof -
  { fix τ::('fun, 'atom) term_type assume "∧f T. Fun f T ⊆ τ ⟹ 0 < arity f"
    hence "∃t'. fv t' = {} ∧ τ = Γ t'"
    proof (induction τ)
      case (Fun f T)
      hence "arity f > 0" by auto
      from Fun.IH Fun.prem(1) have "∃Y. map Γ Y = T ∧ (∀x ∈ set Y. fv x = {})"
      proof (induction T)
        case (Cons x X)
        hence "∧g Y. Fun g Y ⊆ Fun f X ⟹ 0 < arity g" by auto
        hence "∃Y. map Γ Y = X ∧ (∀x ∈ set Y. fv x = {})" using Cons by auto
        moreover have "∃t'. fv t' = {} ∧ x = Γ t'" using Cons by auto
        ultimately obtain y Y where
          "fv y = {}" "Γ y = x" "map Γ Y = X" "∀x ∈ set Y. fv x = {}"
          using Cons by mouna
        hence "map Γ (y#Y) = x#X ∧ (∀x ∈ set (y#Y). fv x = {})" by auto
        thus ?case by meson
      qed simp
      then obtain Y where "map Γ Y = T" "∀x ∈ set Y. fv x = {}" by metis
      hence "fv (Fun f Y) = {}" "Γ (Fun f Y) = TComp f T" using fun_type[OF (arity f > 0)] by auto
      thus ?case by (metis exI[of "λt. fv t = {} ∧ Γ t = TComp f T" "Fun f Y"])
    qed (metis atype_ground_term_ex)
  }
  thus ?thesis by (metis Γ_wf(1))
qed

lemma type_wfttype_inhabited:
  assumes "∧f T. Fun f T ⊆ τ ⟹ 0 < arity f" "wftrm τ"
  shows "∃t. Γ t = τ ∧ wftrm t"

```

```

using assms
proof (induction  $\tau$ )
  case (Fun f Y)
  have IH: " $\exists t. \Gamma t = y \wedge wf_{trm} t$ " when  $y: "y \in set Y"$  for  $y$ 
  proof -
    have " $wf_{trm} y$ "
    using Fun y unfolding wf_trm_def
    by (metis Fun_param_is_subterm term.le_less_trans)
    moreover have " $Fun g Z \sqsubseteq y \implies 0 < arity g$ " for  $g Z$ 
    using Fun y by auto
    ultimately show ?thesis using Fun.IH[OF y] by auto
  qed

  from Fun have " $arity f = length Y$ " " $arity f > 0$ " unfolding wf_trm_def by force+
  moreover from IH have " $\exists X. map \Gamma X = Y \wedge (\forall x \in set X. wf_{trm} x)$ "
  by (induct Y, simp_all, metis list.simps(9) set_ConsD)
  ultimately show ?case by (metis fun_type length_map wf_trmI)
qed (use atypes_inhabited wf_trm_def in blast)

lemma type_pgwt_inhabited: " $wf_{trm} t \implies \exists t'. \Gamma t = \Gamma t' \wedge public\_ground\_wf\_term t'$ "
proof -
  assume " $wf_{trm} t$ "
  { fix  $\tau$  assume " $\Gamma t = \tau$ "
    hence " $\exists t'. \Gamma t = \Gamma t' \wedge public\_ground\_wf\_term t'$ " using  $\langle wf_{trm} t \rangle$ 
    proof (induction  $\tau$  arbitrary: t)
      case (Var a t)
      then obtain c where " $\Gamma t = \Gamma (Fun c [])$ " " $arity c = 0$ " " $public c$ "
      using const_type_inv[of _ "[]" a] infinite_typed_consts(1)[of a] not_finite_existsD
      by force
      thus ?case using PGWT[OF  $\langle public c \rangle$ , of "[]"] by auto
    next
      case (Fun f Y t)
      have *: " $arity f > 0$ " " $public f$ " " $arity f = length Y$ "
      using fun_type_inv[OF  $\langle \Gamma t = TComp f Y \rangle$ ] fun_type_inv_wf[OF  $\langle \Gamma t = TComp f Y \rangle \langle wf_{trm} t \rangle$ ]
      by auto
      have " $\bigwedge y. y \in set Y \implies \exists t'. y = \Gamma t' \wedge public\_ground\_wf\_term t'$ "
      using Fun.prem(1) Fun.IH  $\Gamma\_wf(1)$ [of _ _ t]  $\Gamma\_wf'[OF \langle wf_{trm} t \rangle]$  type_wfttype_inhabited
      by (metis Fun_param_is_subterm term.order_trans wf_trm_subterm_eq)
      hence " $\exists X. map \Gamma X = Y \wedge (\forall x \in set X. public\_ground\_wf\_term x)$ "
      by (induct Y, simp_all, metis list.simps(9) set_ConsD)
      then obtain X where  $X: "map \Gamma X = Y"$  " $\bigwedge x. x \in set X \implies public\_ground\_wf\_term x$ " by mouna
      hence " $arity f = length X$ " using *(3) by auto
      have " $\Gamma t = \Gamma (Fun f X)$ " " $public\_ground\_wf\_term (Fun f X)$ "
      using fun_type[OF *(1), of X] Fun.prem(1) X(1) apply simp
      using PGWT[OF *(2)  $\langle arity f = length X \rangle$  X(2)] by metis
      thus ?case by metis
    qed
  }
  thus ?thesis using  $\langle wf_{trm} t \rangle$  by auto
qed

lemma pgwt_type_map:
  assumes " $public\_ground\_wf\_term t$ "
  shows " $\Gamma t = TAtom a \implies \exists f. t = Fun f []$ " " $\Gamma t = TComp g Y \implies \exists X. t = Fun g X \wedge map \Gamma X = Y$ "
proof -
  let ?A = " $\Gamma t = TAtom a \longrightarrow (\exists f. t = Fun f [])$ "
  let ?B = " $\Gamma t = TComp g Y \longrightarrow (\exists X. t = Fun g X \wedge map \Gamma X = Y)$ "
  have "?A  $\wedge$  ?B"
  proof (cases " $\Gamma t$ ")
    case (Var a)
    obtain f X where " $t = Fun f X$ " " $arity f = length X$ "
    using pgwt_fun[OF assms(1)] pgwt_arity[OF assms(1)] by fastforce+
    thus ?thesis using const_type_inv  $\langle \Gamma t = TAtom a \rangle$  by auto
  qed

```

```

next
  case (Fun g Y)
  obtain f X where *: "t = Fun f X" using pgwt_fun[OF assms(1)] by force
  hence "f = g" "map  $\Gamma$  X = Y"
    using fun_type_id_eq ( $\Gamma$  t = TComp g Y) fun_type[OF fun_type_inv(1)[OF ( $\Gamma$  t = TComp g Y)]]
    by fastforce+
  thus ?thesis using *(1) ( $\Gamma$  t = TComp g Y) by auto
qed
thus " $\Gamma$  t = TAtom a  $\implies \exists f. t = \text{Fun } f []$ " " $\Gamma$  t = TComp g Y  $\implies \exists X. t = \text{Fun } g X \wedge \text{map } \Gamma X = Y$ "
  by auto
qed

lemma wt_subst_Var[simp]: "wt_subst Var" by (metis wt_subst_def)

lemma wt_subst_trm: " $(\bigwedge v. v \in \text{fv } t \implies \Gamma (\text{Var } v) = \Gamma (\vartheta v)) \implies \Gamma t = \Gamma (t \cdot \vartheta)$ "
proof (induction t)
  case (Fun f X)
  hence *: " $\bigwedge x. x \in \text{set } X \implies \Gamma x = \Gamma (x \cdot \vartheta)$ " by auto
  show ?case
  proof (cases "f  $\in \Sigma_f$ ")
    case True
    hence " $\forall X. \Gamma (\text{Fun } f X) = \text{TComp } f (\text{map } \Gamma X)$ " using fun_type' by auto
    thus ?thesis using * by auto
  next
    case False
    hence " $\exists a \in \mathfrak{T}_a. \forall X. \Gamma (\text{Fun } f X) = \text{TAtom } a$ " using const_type' by auto
    thus ?thesis by auto
  qed
qed auto

lemma wt_subst_trm': " $\llbracket \text{wt\_subst } \sigma; \Gamma s = \Gamma t \rrbracket \implies \Gamma (s \cdot \sigma) = \Gamma (t \cdot \sigma)$ "
by (metis wt_subst_trm wt_subst_def)

lemma wt_subst_trm'': " $\text{wt\_subst } \sigma \implies \Gamma t = \Gamma (t \cdot \sigma)$ "
by (metis wt_subst_trm wt_subst_def)

lemma wt_subst_compose:
  assumes "wt_subst  $\vartheta$ " "wt_subst  $\delta$ " shows "wt_subst ( $\vartheta \circ_s \delta$ )"
proof -
  have " $\bigwedge v. \Gamma (\vartheta v) = \Gamma (\vartheta v \cdot \delta)$ " using wt_subst_trm ( $\text{wt\_subst } \delta$ ) unfolding wt_subst_def by metis
  moreover have " $\bigwedge v. \Gamma (\text{Var } v) = \Gamma (\vartheta v)$ " using ( $\text{wt\_subst } \vartheta$ ) unfolding wt_subst_def by metis
  ultimately have " $\bigwedge v. \Gamma (\text{Var } v) = \Gamma (\vartheta v \cdot \delta)$ " by metis
  thus ?thesis unfolding wt_subst_def subst_compose_def by metis
qed

lemma wt_subst_TAtom_Var_cases:
  assumes  $\vartheta$ : "wt_subst  $\vartheta$ " "wf_trms (subst_range  $\vartheta$ )"
  and x: " $\Gamma (\text{Var } x) = \text{TAtom } a$ "
  shows " $(\exists y. \vartheta x = \text{Var } y) \vee (\exists c. \vartheta x = \text{Fun } c [])$ "
proof (cases " $(\exists y. \vartheta x = \text{Var } y)$ ")
  case False
  then obtain c T where c: " $\vartheta x = \text{Fun } c T$ "
    by (cases " $\vartheta x$ ") simp_all
  hence "wf_trm (Fun c T)"
    using  $\vartheta(2)$  by fastforce
  hence " $T = []$ "
    using const_type_inv_wf[of c T a] x c wt_subst_trm''[OF  $\vartheta(1)$ , of "Var x"]
    by fastforce
  thus ?thesis
    using c by blast
qed simp

lemma wt_subst_TAtom_fv:

```

```

assumes  $\vartheta$ : "wt_subst  $\vartheta$ " " $\forall x. wf_{trm} (\vartheta x)$ "
and " $\forall x \in fv\ t - X. \exists a. \Gamma (Var\ x) = TAtom\ a$ "
shows " $\forall x \in fv\ (t \cdot \vartheta) - fv_{set} (\vartheta \cdot X). \exists a. \Gamma (Var\ x) = TAtom\ a$ "
using assms(3)
proof (induction t)
  case (Var x) thus ?case
  proof (cases "x  $\in X$ ")
    case False
    with Var obtain a where " $\Gamma (Var\ x) = TAtom\ a$ " by moura
    hence *: " $\Gamma (\vartheta x) = TAtom\ a$ " " $wf_{trm} (\vartheta x)$ " using  $\vartheta$  unfolding wt_subst_def by auto
    show ?thesis
  proof (cases " $\vartheta x$ ")
    case (Var y) thus ?thesis using * by auto
  next
    case (Fun f T)
    hence " $T = []$ " using * const_type_inv[of f T a] unfolding wf_trm_def by auto
    thus ?thesis using Fun by auto
  qed
qed auto
qed fastforce

lemma wt_subst_TAtom_subterms_subst:
  assumes "wt_subst  $\vartheta$ " " $\forall x \in fv\ t. \exists a. \Gamma (Var\ x) = TAtom\ a$ " " $wf_{trms} (\vartheta \cdot fv\ t)$ "
  shows "subterms (t  $\cdot \vartheta$ ) = subterms t  $\cdot_{set} \vartheta$ "
using assms(2,3)
proof (induction t)
  case (Var x)
  obtain a where a: " $\Gamma (Var\ x) = TAtom\ a$ " using Var.prem(1) by moura
  hence " $\Gamma (\vartheta x) = TAtom\ a$ " using wt_subst_trm''[OF assms(1), of "Var x"] by simp
  hence " $(\exists y. \vartheta x = Var\ y) \vee (\exists c. \vartheta x = Fun\ c\ [])$ "
  using const_type_inv_wf Var.prem(2) by (cases " $\vartheta x$ ") auto
  thus ?case by auto
next
  case (Fun f T)
  have "subterms (t  $\cdot \vartheta$ ) = subterms t  $\cdot_{set} \vartheta$ " when "t  $\in set\ T$ " for t
  using that Fun.prem(1,2) Fun.IH[OF that]
  by auto
  thus ?case by auto
qed

lemma wt_subst_TAtom_subterms_set_subst:
  assumes "wt_subst  $\vartheta$ " " $\forall x \in fv_{set}\ M. \exists a. \Gamma (Var\ x) = TAtom\ a$ " " $wf_{trms} (\vartheta \cdot fv_{set}\ M)$ "
  shows "subterms_set (M  $\cdot_{set} \vartheta$ ) = subterms_set M  $\cdot_{set} \vartheta$ "
proof
  show "subterms_set (M  $\cdot_{set} \vartheta$ )  $\subseteq$  subterms_set M  $\cdot_{set} \vartheta$ "
  proof
    fix t assume "t  $\in$  subterms_set (M  $\cdot_{set} \vartheta$ )"
    then obtain s where s: "s  $\in M$ " "t  $\in$  subterms (s  $\cdot \vartheta$ )" by auto
    thus "t  $\in$  subterms_set M  $\cdot_{set} \vartheta$ "
    using assms(2,3) wt_subst_TAtom_subterms_subst[OF assms(1), of s]
    by auto
  qed

  show "subterms_set M  $\cdot_{set} \vartheta \subseteq$  subterms_set (M  $\cdot_{set} \vartheta$ )"
  proof
    fix t assume "t  $\in$  subterms_set M  $\cdot_{set} \vartheta$ "
    then obtain s where s: "s  $\in M$ " "t  $\in$  subterms s  $\cdot_{set} \vartheta$ " by auto
    thus "t  $\in$  subterms_set (M  $\cdot_{set} \vartheta$ )"
    using assms(2,3) wt_subst_TAtom_subterms_subst[OF assms(1), of s]
    by auto
  qed
qed

```

```

lemma wt_subst_subst_upd:
  assumes "wt_subst  $\vartheta$ "
  and " $\Gamma$  (Var x) =  $\Gamma$  t"
  shows "wt_subst ( $\vartheta$  (x := t))"
using assms unfolding wt_subst_def
by (metis fun_upd_other fun_upd_same)

lemma wt_subst_const_fv_type_eq:
  assumes " $\forall x \in \text{fv } t. \exists a. \Gamma$  (Var x) = TAtom a"
  and " $\delta$ : 'wt_subst  $\delta$ ' 'wf_trms (subst_range  $\delta$ )'"
  shows " $\forall x \in \text{fv } (t \cdot \delta). \exists y \in \text{fv } t. \Gamma$  (Var x) =  $\Gamma$  (Var y)"
using assms(1)
proof (induction t)
  case (Var x)
  then obtain a where a: " $\Gamma$  (Var x) = TAtom a" by moura
  show ?case
  proof (cases " $\delta$  x")
    case (Fun f T)
    hence "wf_trm (Fun f T)" " $\Gamma$  (Fun f T) = TAtom a"
      using a wt_subst_trm'' [OF  $\delta$ (1), of "Var x"]  $\delta$ (2) by fastforce+
    thus ?thesis using const_type_inv_wf Fun by fastforce
    qed (use a wt_subst_trm'' [OF  $\delta$ (1), of "Var x"] in simp)
  qed fastforce

lemma TComp_term_cases:
  assumes "wf_trm t" " $\Gamma$  t = TComp f T"
  shows " $(\exists v. t = \text{Var } v) \vee (\exists T'. t = \text{Fun } f T' \wedge T = \text{map } \Gamma T' \wedge T' \neq [])$ "
proof (cases " $\exists v. t = \text{Var } v$ ")
  case False
  then obtain T' where T': "t = Fun f T'" "T = map  $\Gamma$  T'"
    using assms fun_type [OF fun_type_inv(1) [OF assms(2)]] fun_type_id_eq
    by (cases t) force+
  thus ?thesis using assms fun_type_inv(1) fun_type_inv_wf by fastforce
qed metis

lemma TAtom_term_cases:
  assumes "wf_trm t" " $\Gamma$  t = TAtom  $\tau$ "
  shows " $(\exists v. t = \text{Var } v) \vee (\exists f. t = \text{Fun } f [])$ "
using assms const_type_inv unfolding wf_trm_def by (cases t) auto

lemma subterm_eq_imp_subterm_type_eq:
  assumes "wf_trm t" " $s \sqsubseteq t$ "
  shows " $\Gamma s \sqsubseteq \Gamma t$ "
using assms(2,1)
proof (induction t)
  case (Fun f T) thus ?case
  proof (cases " $s = \text{Fun } f T$ ")
    case False
    then obtain x where x: " $x \in \text{set } T$ " " $s \sqsubseteq x$ " using Fun.prem(1) by auto
    hence "wf_trm x" using wf_trm_subterm [OF Fun.prem(2)] Fun_param_is_subterm [of _ T f] by auto
    hence " $\Gamma s \sqsubseteq \Gamma x$ " using Fun.IH [OF x] by simp
    moreover have "arity f > 0" using x fun_type_inv_wf Fun.prem
      by (metis length_pos_if_in_set term.order_refl wf_trm_def)
    ultimately show ?thesis using x Fun.prem fun_type [of f T] by auto
    qed simp
  qed simp

lemma subterm_funs_term_in_type:
  assumes "wf_trm t" " $\text{Fun } f T \sqsubseteq t$ " " $\Gamma$  (Fun f T) = TComp f (map  $\Gamma$  T)"
  shows " $f \in \text{funs\_term } (\Gamma t)$ "
using assms(2,1,3)
proof (induction t)
  case (Fun f' T')

```

```

hence [simp]: "wftrm (Fun f T)" by (metis wf_trm_subterm)
{ fix a assume  $\tau$ : " $\Gamma$  (Fun f' T') = TAtom a"
  hence "Fun f T = Fun f' T'" using Fun TAtom_term_cases subterm_Var_const by metis
  hence False using Fun.prem3(3)  $\tau$  by simp
}
moreover
{ fix g S assume  $\tau$ : " $\Gamma$  (Fun f' T') = TComp g S"
  hence "g = f'" "S = map  $\Gamma$  T'"
    using Fun.prem2(2) fun_type_id_eq[OF  $\tau$ ] fun_type[OF fun_type_inv(1)[OF  $\tau$ ]]
    by auto
  hence  $\tau'$ : " $\Gamma$  (Fun f' T') = TComp f' (map  $\Gamma$  T')" using  $\tau$  by auto
  hence "g  $\in$  funs_term ( $\Gamma$  (Fun f' T'))" using  $\tau$  by auto
  moreover {
    assume "Fun f T  $\neq$  Fun f' T'"
    then obtain x where "x  $\in$  set T'" "Fun f T  $\sqsubseteq$  x" using Fun.prem1(1) by auto
    hence "f  $\in$  funs_term ( $\Gamma$  x)"
      using Fun.IH[OF _ _ Fun.prem3(3), of x] wf_trm_subterm[OF (wftrm (Fun f' T'))], of x]
      by force
    moreover have " $\Gamma$  x  $\in$  set (map  $\Gamma$  T')" using  $\tau'$  (x  $\in$  set T') by auto
    ultimately have "f  $\in$  funs_term ( $\Gamma$  (Fun f' T'))" using  $\tau'$  by auto
  }
  ultimately have ?case by (cases "Fun f T = Fun f' T'") (auto simp add: (g = f'))
}
ultimately show ?case by (cases " $\Gamma$  (Fun f' T')") auto
qed simp

```

```

lemma wt_subst_fv_termtyp_subterm:
  assumes "x  $\in$  fv ( $\vartheta$  y)"
    and "wtsubst  $\vartheta$ "
    and "wftrm ( $\vartheta$  y)"
  shows " $\Gamma$  (Var x)  $\sqsubseteq$   $\Gamma$  (Var y)"
using subterm_imp_subtermtyp_eq[OF assms(3) var_is_subterm[OF assms(1)]]
  wt_subst_trm''[OF assms(2), of "Var y"]
by auto

```

```

lemma wt_subst_fv_set_termtyp_subterm:
  assumes "x  $\in$  fvset ( $\vartheta$  ' Y)"
    and "wtsubst  $\vartheta$ "
    and "wftrms (subst_range  $\vartheta$ )"
  shows " $\exists y \in Y. \Gamma$  (Var x)  $\sqsubseteq$   $\Gamma$  (Var y)"
using wt_subst_fv_termtyp_subterm[OF _ assms(2), of x] assms(1,3)
by fastforce

```

```

lemma funs_term_type_iff:
  assumes t: "wftrm t"
    and f: "arity f > 0"
  shows "f  $\in$  funs_term ( $\Gamma$  t)  $\longleftrightarrow$  (f  $\in$  funs_term t  $\vee$  ( $\exists x \in$  fv t. f  $\in$  funs_term ( $\Gamma$  (Var x))))"
    (is "?P t  $\longleftrightarrow$  ?Q t")
using t
proof (induction t)
  case (Fun g T)
  hence IH: "?P s  $\longleftrightarrow$  ?Q s" when "s  $\in$  set T" for s
    using that wf_trm_subterm[OF _ Fun_param_is_subterm]
    by blast
  have 0: "arity g = length T" using Fun.prem3 unfolding wf_trm_def by auto
  show ?case
  proof (cases "f = g")
    case True thus ?thesis using fun_type[OF f] by simp
  next
    case False
    have "?P (Fun g T)  $\longleftrightarrow$  ( $\exists s \in$  set T. ?P s)"
    proof
      assume *: "?P (Fun g T)"

```

```

    hence "Γ (Fun g T) = TComp g (map Γ T)"
      using const_type[of g] fun_type[of g] by force
    thus "∃ s ∈ set T. ?P s" using False * by force
  next
    assume *: "∃ s ∈ set T. ?P s"
    hence "Γ (Fun g T) = TComp g (map Γ T)"
      using 0 const_type[of g] fun_type[of g] by force
    thus "?P (Fun g T)" using False * by force
  qed
  thus ?thesis using False f IH by auto
qed
qed simp

lemma funs_term_type_iff':
  assumes M: "wf_trm M"
    and f: "arity f > 0"
  shows "f ∈ ⋃ (funs_term ' Γ ' M) ⟷
    (f ∈ ⋃ (funs_term ' M) ∨ (∃ x ∈ fv_set M. f ∈ funs_term (Γ (Var x))))" (is "?A ⟷ ?B")
proof
  assume ?A
  then obtain t where "t ∈ M" "wf_trm t" "f ∈ funs_term (Γ t)" using M by moura
  thus ?B using funs_term_type_iff[OF _ f, of t] by auto
next
  assume ?B
  then obtain t where "t ∈ M" "wf_trm t" "f ∈ funs_term t ∨ (∃ x ∈ fv t. f ∈ funs_term (Γ (Var x)))"
  using M by auto
  thus ?A using funs_term_type_iff[OF _ f, of t] by blast
qed

lemma Ana_subterm_type:
  assumes "Ana t = (K,M)"
    and "wf_trm t"
    and "m ∈ set M"
  shows "Γ m ⊆ Γ t"
proof -
  have "m ⊆ t" using Ana_subterm[OF assms(1)] assms(3) by auto
  thus ?thesis using subterm_eq_imp_subterm_type_eq[OF assms(2)] by simp
qed

lemma wf_trm_TAtom_subterms:
  assumes "wf_trm t" "Γ t = TAtom τ"
  shows "subterms t = {t}"
using assms const_type_inv unfolding wf_trm_def by (cases t) force+

lemma wf_trm_TComp_subterm:
  assumes "wf_trm s" "t ⊆ s"
  obtains f T where "Γ s = TComp f T"
proof (cases s)
  case (Var x) thus ?thesis using ⟨t ⊆ s⟩ by simp
next
  case (Fun g S)
  hence "length S > 0" using assms Fun_subterm_inside_params[of t g S] by auto
  hence "arity g > 0" by (metis ⟨wf_trm s⟩ ⟨s = Fun g S⟩ term.order_refl wf_trm_def)
  thus ?thesis using fun_type ⟨s = Fun g S⟩ that by auto
qed

lemma SMP_empty[simp]: "SMP {} = {}"
proof (rule ccontr)
  assume "SMP {} ≠ {}"
  then obtain t where "t ∈ SMP {}" by auto
  thus False by (induct t rule: SMP.induct) auto
qed

```

```

lemma SMP_I:
  assumes "s ∈ M" "wt_subst δ" "t ⊆ s · δ" "⋀v. wf_trm (δ v)"
  shows "t ∈ SMP M"
using SMP.Substitution[OF SMP.MP[OF assms(1)] assms(2)] SMP.Subterm[of "s · δ" M t] assms(3,4)
by (cases "t = s · δ") simp_all

lemma SMP_wf_trm:
  assumes "t ∈ SMP M" "wf_trms M"
  shows "wf_trm t"
using assms(1)
by (induct t rule: SMP.induct)
  (use assms(2) in blast,
   use wf_trm_subtermeq in blast,
   use wf_trm_subst in blast,
   use Ana_keys_wf' in blast)

lemma SMP_ikI[intro]: "t ∈ ikst S ⇒ t ∈ SMP (trmsst S)" by force

lemma MP_setI[intro]: "x ∈ set S ⇒ trmsstp x ⊆ trmsst S" by force

lemma SMP_setI[intro]: "x ∈ set S ⇒ trmsstp x ⊆ SMP (trmsst S)" by force

lemma SMP_subset_I:
  assumes M: "∀t ∈ M. ∃s δ. s ∈ N ∧ wt_subst δ ∧ wf_trms (subst_range δ) ∧ t = s · δ"
  shows "SMP M ⊆ SMP N"
proof
  fix t show "t ∈ SMP M ⇒ t ∈ SMP N"
  proof (induction t rule: SMP.induct)
    case (MP t)
    then obtain s δ where s: "s ∈ N" "wt_subst δ" "wf_trms (subst_range δ)" "t = s · δ"
      using M by mouna
    show ?case using SMP_I[OF s(1,2), of "s · δ"] s(3,4) wf_trm_subst_range_iff by fast
  qed (auto intro!: SMP.Substitution[of _ N])
qed

lemma SMP_union: "SMP (A ∪ B) = SMP A ∪ SMP B"
proof
  show "SMP (A ∪ B) ⊆ SMP A ∪ SMP B"
  proof
    fix t assume "t ∈ SMP (A ∪ B)"
    thus "t ∈ SMP A ∪ SMP B" by (induct rule: SMP.induct) blast+
  qed
  { fix t assume "t ∈ SMP A" hence "t ∈ SMP (A ∪ B)" by (induct rule: SMP.induct) blast+ }
  moreover { fix t assume "t ∈ SMP B" hence "t ∈ SMP (A ∪ B)" by (induct rule: SMP.induct) blast+ }
  }
  ultimately show "SMP A ∪ SMP B ⊆ SMP (A ∪ B)" by blast
qed

lemma SMP_append[simp]: "SMP (trmsst (S@S')) = SMP (trmsst S) ∪ SMP (trmsst S')" (is "?A = ?B")
using SMP_union by simp

lemma SMP_mono: "A ⊆ B ⇒ SMP A ⊆ SMP B"
proof -
  assume "A ⊆ B"
  then obtain C where "B = A ∪ C" by mouna
  thus "SMP A ⊆ SMP B" by (simp add: SMP_union)
qed

lemma SMP_Union: "SMP (⋃m ∈ M. f m) = (⋃m ∈ M. SMP (f m))"
proof
  show "SMP (⋃m ∈ M. f m) ⊆ (⋃m ∈ M. SMP (f m))"

```



```

proof
  fix t assume "t ∈ SMP (⋃m∈M. f m)"
  thus "t ∈ (⋃m∈M. SMP (f m))" by (induct t rule: SMP.induct) force+
qed
show "(⋃m∈M. SMP (f m)) ⊆ SMP (⋃m∈M. f m)"
proof
  fix t assume "t ∈ (⋃m∈M. SMP (f m))"
  then obtain m where "m ∈ M" "t ∈ SMP (f m)" by moura
  thus "t ∈ SMP (⋃m∈M. f m)" using SMP_mono[of "f m" "⋃m∈M. f m"] by auto
qed
qed

lemma SMP_singleton_ex:
  "t ∈ SMP M ⟹ (∃ m ∈ M. t ∈ SMP {m})"
  "m ∈ M ⟹ t ∈ SMP {m} ⟹ t ∈ SMP M"
using SMP_Union[of "λt. {t}" M] by auto

lemma SMP_Cons: "SMP (trmsst (x#S)) = SMP (trmsst [x]) ∪ SMP (trmsst S)"
using SMP_append[of "[x]" S] by auto

lemma SMP_Nil[simp]: "SMP (trmsst []) = {}"
proof -
  { fix t assume "t ∈ SMP (trmsst [])" hence False by induct auto }
  thus ?thesis by blast
qed

lemma SMP_subset_union_eq: assumes "M ⊆ SMP N" shows "SMP N = SMP (M ∪ N)"
proof -
  { fix t assume "t ∈ SMP (M ∪ N)" hence "t ∈ SMP N"
    using assms by (induction rule: SMP.induct) blast+
  }
  thus ?thesis using SMP_union by auto
qed

lemma SMP_subterms_subset: "subtermsset M ⊆ SMP M"
proof
  fix t assume "t ∈ subtermsset M"
  then obtain m where "m ∈ M" "t ⊆ m" by auto
  thus "t ∈ SMP M" using SMP_I[of _ _ Var] by auto
qed

lemma SMP_SMP_subset: "N ⊆ SMP M ⟹ SMP N ⊆ SMP M"
by (metis SMP_mono SMP_subset_union_eq Un_commute Un_upper2)

lemma wt_subst_rm_vars: "wtsubst δ ⟹ wtsubst (rm_vars X δ)"
using rm_vars_dom unfolding wtsubst_def by auto

lemma wt_subst_SMP_subset:
  assumes "trmsst S ⊆ SMP S'" "wtsubst δ" "wftrms (subst_range δ)"
  shows "trmsst (S ·st δ) ⊆ SMP S'"
proof
  fix t assume *: "t ∈ trmsst (S ·st δ)"
  show "t ∈ SMP S'" using trm_strand_subst_cong(2)[OF *]
  proof
    assume "∃ t'. t = t' · δ ∧ t' ∈ trmsst S"
    thus "t ∈ SMP S'" using assms SMP.Substitution by auto
  next
    assume "∃ X F. Inequality X F ∈ set S ∧ (∃ t' ∈ trmspairs F. t = t' · rm_vars (set X) δ)"
    then obtain X F t' where **:
      "Inequality X F ∈ set S" "t' ∈ trmspairs F" "t = t' · rm_vars (set X) δ"
    by force
    then obtain s where s: "s ∈ trmsstp (Inequality X F)" "t = s · rm_vars (set X) δ" by moura
    hence "s ∈ SMP (trmsst S)" using **(1) by force
  qed

```

```

    hence "t ∈ SMP (trmsst S)"
      using SMP.Substitution[OF _ wt_subst_rm_vars[OF assms(2)] wf_trms_subst_rm_vars'[OF assms(3)]]
      unfolding s(2) by blast
    thus "t ∈ SMP S'" by (metis SMP_union SMP_subset_union_eq UnCI assms(1))
  qed
qed

lemma MP_subset_SMP: "⋃ (trmsstp ' set S) ⊆ SMP (trmsst S)" "trmsst S ⊆ SMP (trmsst S)" "M ⊆ SMP M"
by auto

lemma SMP_fun_map_snd_subset: "SMP (trmsst (map Send X)) ⊆ SMP (trmsst [Send (Fun f X)])"
proof
  fix t assume "t ∈ SMP (trmsst (map Send X))" thus "t ∈ SMP (trmsst [Send (Fun f X)])"
  proof (induction t rule: SMP.induct)
    case (MP t)
    hence "t ∈ set X" by auto
    hence "t ⊆ Fun f X" by (metis subtermI')
    thus ?case using SMP.Subterm[of "Fun f X" "trmsst [Send (Fun f X)]" t] using SMP.MP by auto
  qed blast+
qed

lemma SMP_wt_subst_subset:
  assumes "t ∈ SMP (M ·set I)" "wtsubst I" "wftrms (subst_range I)"
  shows "t ∈ SMP M"
using assms wf_trm_subst_range_iff[of I] by (induct t rule: SMP.induct) blast+

lemma SMP_wt_instances_subset:
  assumes "∀ t ∈ M. ∃ s ∈ N. ∃ δ. t = s · δ ∧ wtsubst δ ∧ wftrms (subst_range δ)"
  and "t ∈ SMP M"
  shows "t ∈ SMP N"
proof -
  obtain m where m: "m ∈ M" "t ∈ SMP {m}" using SMP_singleton_ex(1)[OF assms(2)] by blast
  then obtain n δ where n: "n ∈ N" "m = n · δ" "wtsubst δ" "wftrms (subst_range δ)"
    using assms(1) by fast

  have "t ∈ SMP (N ·set δ)" using n(1,2) SMP_singleton_ex(2)[of m "N ·set δ", OF _ m(2)] by fast
  thus ?thesis using SMP_wt_subst_subset[OF _ n(3,4)] by blast
qed

lemma SMP_consts:
  assumes "∀ t ∈ M. ∃ c. t = Fun c []"
  and "∀ t ∈ M. Ana t = ([], [])"
  shows "SMP M = M"
proof
  show "SMP M ⊆ M"
  proof
    fix t show "t ∈ SMP M ⇒ t ∈ M"
    apply (induction t rule: SMP.induct)
    by (use assms in auto)
  qed
qed auto

lemma SMP_subterms_eq:
  "SMP (subtermsset M) = SMP M"
proof
  show "SMP M ⊆ SMP (subtermsset M)" using SMP_mono[of M "subtermsset M"] by blast
  show "SMP (subtermsset M) ⊆ SMP M"
  proof
    fix t show "t ∈ SMP (subtermsset M) ⇒ t ∈ SMP M" by (induction t rule: SMP.induct) blast+
  qed
qed

lemma SMP_funs_term:

```

```

assumes t: "t ∈ SMP M" "f ∈ funs_term t ∨ (∃x ∈ fv t. f ∈ funs_term (Γ (Var x)))"
and f: "arity f > 0"
and M: "wf_trms M"
and Ana_f: "∧s K T. Ana s = (K,T) ⇒ f ∈ ∪ (funs_term ' set K) ⇒ f ∈ funs_term s"
shows "f ∈ ∪ (funs_term ' M) ∨ (∃x ∈ fv_set M. f ∈ funs_term (Γ (Var x)))"
using t
proof (induction t rule: SMP.induct)
  case (Subterm t t')
  thus ?case by (metis UN_I vars_iff_subtermeq funs_term_subterms_eq(1) term.order_trans)
next
  case (Substitution t δ)
  show ?case
    using M SMP_wf_trm[OF Substitution.hyps(1)] wf_trm_subst[of δ t, OF Substitution.hyps(3)]
    funs_term_type_iff[OF _ f] wt_subst_trm''[OF Substitution.hyps(2), of t]
    Substitution.premis Substitution.IH
    by metis
next
  case (Ana t K T t')
  thus ?case
    using Ana_f[OF Ana.hyps(2)] Ana_keys_fv[OF Ana.hyps(2)]
    by fastforce
qed auto

lemma id_type_eq:
  assumes "Γ (Fun f X) = Γ (Fun g Y)"
  shows "f ∈ C ⇒ g ∈ C" "f ∈ Σ_f ⇒ g ∈ Σ_f"
using assms const_type' fun_type' id_union_univ(1)
by (metis UNIV_I UnE "term.distinct"(1))+

lemma fun_type_arg_cong:
  assumes "f ∈ Σ_f" "g ∈ Σ_f" "Γ (Fun f (x#X)) = Γ (Fun g (y#Y))"
  shows "Γ x = Γ y" "Γ (Fun f X) = Γ (Fun g Y)"
using assms fun_type' by auto

lemma fun_type_arg_cong':
  assumes "f ∈ Σ_f" "g ∈ Σ_f" "Γ (Fun f (X@x#X')) = Γ (Fun g (Y@y#Y'))" "length X = length Y"
  shows "Γ x = Γ y"
using assms
proof (induction X arbitrary: Y)
  case Nil thus ?case using fun_type_arg_cong(1)[of f g x X' y Y'] by auto
next
  case (Cons x' X Y'')
  then obtain y' Y where "Y'' = y'#Y" by (metis length_Suc_conv)
  hence "Γ (Fun f (X@x#X')) = Γ (Fun g (Y@y#Y'))" "length X = length Y"
  using Cons.premis(3,4) fun_type_arg_cong(2)[OF Cons.premis(1,2), of x' "X@x#X'"] by auto
  thus ?thesis using Cons.IH[OF Cons.premis(1,2)] by auto
qed

lemma fun_type_param_idx: "Γ (Fun f T) = Fun g S ⇒ i < length T ⇒ Γ (T ! i) = S ! i"
by (metis fun_type fun_type_id_eq fun_type_inv(1) nth_map term.inject(2))

lemma fun_type_param_ex:
  assumes "Γ (Fun f T) = Fun g (map Γ S)" "t ∈ set S"
  shows "∃s ∈ set T. Γ s = Γ t"
using fun_type_length_eq[OF assms(1)] length_map[of Γ S] assms(2)
fun_type_param_idx[OF assms(1)] nth_map in_set_conv_nth
by metis

lemma tfr_stp_all_split:
  "list_all tfr_stp (x#S) ⇒ list_all tfr_stp [x]"
  "list_all tfr_stp (x#S) ⇒ list_all tfr_stp S"
  "list_all tfr_stp (S@S') ⇒ list_all tfr_stp S"
  "list_all tfr_stp (S@S') ⇒ list_all tfr_stp S'"

```

```

"list_all tfr_stp (S@x#S')  $\impl$  list_all tfr_stp (S@S')"
by fastforce+

lemma tfr_stp_all_append:
  assumes "list_all tfr_stp S" "list_all tfr_stp S'"
  shows "list_all tfr_stp (S@S')"
using assms by fastforce

lemma tfr_stp_all_wt_subst_apply:
  assumes "list_all tfr_stp S"
  and  $\vartheta$ : "wt_subst  $\vartheta$ " "wf_trms (subst_range  $\vartheta$ )"
  "bvars_st S  $\cap$  range_vars  $\vartheta$  = {}"
  shows "list_all tfr_stp (S  $\cdot_{st}$   $\vartheta$ )"
using assms(1,4)
proof (induction S)
  case (Cons x S)
  hence IH: "list_all tfr_stp (S  $\cdot_{st}$   $\vartheta$ )"
  using tfr_stp_all_split(2)[of x S]
  unfolding range_vars_alt_def by fastforce
  thus ?case
  proof (cases x)
    case (Equality a t t')
    hence "( $\exists \delta$ . Unifier  $\delta$  t t')  $\longrightarrow \Gamma$  t =  $\Gamma$  t'" using Cons.prem by auto
    hence "( $\exists \delta$ . Unifier  $\delta$  (t  $\cdot$   $\vartheta$ ) (t'  $\cdot$   $\vartheta$ ))  $\longrightarrow \Gamma$  (t  $\cdot$   $\vartheta$ ) =  $\Gamma$  (t'  $\cdot$   $\vartheta$ )"
    by (metis Unifier_comp' wt_subst_trm'[OF assms(2)])
    moreover have "(x#S)  $\cdot_{st}$   $\vartheta$  = Equality a (t  $\cdot$   $\vartheta$ ) (t'  $\cdot$   $\vartheta$ )#(S  $\cdot_{st}$   $\vartheta$ )"
    using (x = Equality a t t') by auto
    ultimately show ?thesis using IH by auto
  next
    case (Inequality X F)
    let ? $\sigma$  = "rm_vars (set X)  $\vartheta$ "
    let ?G = "F  $\cdot_{pairs}$  ? $\sigma$ "

    let ?P = " $\lambda F$  X.  $\forall x \in fv_{pairs} F - set X. \exists a. \Gamma$  (Var x) = TAtom a"
    let ?Q = " $\lambda F$  X.
       $\forall f T. \text{Fun } f T \in \text{subterms}_{set} (\text{trms}_{pairs} F) \longrightarrow T = [] \vee (\exists s \in set T. s \notin \text{Var } 'set X) "$ "

    have 0: "set X  $\cap$  range_vars ? $\sigma$  = {}"
    using Cons.prem(2) Inequality rm_vars_img_subset[of "set X"]
    by (auto simp add: subst_domain_def range_vars_alt_def)

    have 1: "?P F X  $\vee$  ?Q F X" using Inequality Cons.prem by simp

    have 2: "fv_set (? $\sigma$  ' set X) = set X" by auto

    have "?P ?G X" when "?P F X" using that
    proof (induction F)
      case (Cons g G)
      obtain t t' where g: "g = (t,t')" by (metis surj_pair)

      have " $\forall x \in (fv (t \cdot ?\sigma) \cup fv (t' \cdot ?\sigma)) - set X. \exists a. \Gamma$  (Var x) = Var a"
      proof -
        have *: " $\forall x \in fv t - set X. \exists a. \Gamma$  (Var x) = Var a"
        " $\forall x \in fv t' - set X. \exists a. \Gamma$  (Var x) = Var a"
        using g Cons.prem by simp_all

        have **: " $\forall x. wf_{trm} (? \sigma x) "$ "
        using  $\vartheta$ (2) wf_trm_subst_range_iff[of  $\vartheta$ ] wf_trm_subst_rm_vars'[of  $\vartheta$  _ "set X"] by simp

        show ?thesis
        using wt_subst_TAtom_fv[OF wt_subst_rm_vars[OF  $\vartheta$ (1)] ** *(1)]
        wt_subst_TAtom_fv[OF wt_subst_rm_vars[OF  $\vartheta$ (1)] ** *(2)]
        wt_subst_trm'[OF wt_subst_rm_vars[OF  $\vartheta$ (1), of "set X"]] 2
      end
    end
  end
end

```

```

    by blast
  qed
  moreover have "∀x∈fv_pairs (G ·pairs ?σ) - set X. ∃a. Γ (Var x) = Var a"
    using Cons by auto
  ultimately show ?case using g by (auto simp add: subst_apply_pairs_def)
qed (simp add: subst_apply_pairs_def)
hence "?P ?G X ∨ ?Q ?G X"
  using 1 ineq_subterm_inj_cond_subst[OF 0, of "trms_pairs F"] trms_pairs_subst[of F ?σ]
  by presburger
moreover have "(x#S) ·st ∅ = Inequality X (F ·pairs ?σ) # (S ·st ∅)"
  using ⟨x = Inequality X F⟩ by auto
ultimately show ?thesis using IH by simp
qed auto
qed simp

lemma tfr_stp_all_same_type:
  "list_all tfr_stp (S@Equality a t t'#S') ⇒ Unifier δ t t' ⇒ Γ t = Γ t'"
by force+

lemma tfr_subset:
  "⋀A B. tfr_set (A ∪ B) ⇒ tfr_set A"
  "⋀A B. tfr_set B ⇒ A ⊆ B ⇒ tfr_set A"
  "⋀A B. tfr_set B ⇒ SMP A ⊆ SMP B ⇒ tfr_set A"
proof -
  show 1: "tfr_set (A ∪ B) ⇒ tfr_set A" for A B
    using SMP_union[of A B] unfolding tfr_set_def by simp

  fix A B assume B: "tfr_set B"

  show "A ⊆ B ⇒ tfr_set A"
  proof -
    assume "A ⊆ B"
    then obtain C where "B = A ∪ C" by moura
    thus ?thesis using B 1 by blast
  qed

  show "SMP A ⊆ SMP B ⇒ tfr_set A"
  proof -
    assume "SMP A ⊆ SMP B"
    then obtain C where "SMP B = SMP A ∪ C" by moura
    thus ?thesis using B unfolding tfr_set_def by blast
  qed
qed

lemma tfr_empty[simp]: "tfr_set {}"
unfolding tfr_set_def by simp

lemma tfr_consts_mono:
  assumes "∀t ∈ M. ∃c. t = Fun c []"
  and "∀t ∈ M. Ana t = ([], [])"
  and "tfr_set N"
  shows "tfr_set (N ∪ M)"
proof -
  { fix s t
    assume *: "s ∈ SMP (N ∪ M) - range Var" "t ∈ SMP (N ∪ M) - range Var" "∃δ. Unifier δ s t"
    hence **: "is_Fun s" "is_Fun t" "s ∈ SMP N ∨ s ∈ M" "t ∈ SMP N ∨ t ∈ M"
      using assms(3) SMP_consts[OF assms(1,2)] SMP_union[of N M] by auto
    moreover have "Γ s = Γ t" when "s ∈ SMP N" "t ∈ SMP N"
      using that assms(3) *(3) *(1,2) unfolding tfr_set_def by blast
    moreover have "Γ s = Γ t" when st: "s ∈ M" "t ∈ M"
    proof -
      obtain c d where "s = Fun c []" "t = Fun d []" using st assms(1) by moura
      hence "s = t" using *(3) by fast
    }
  }

```

```

      thus ?thesis by metis
    qed
  moreover have "Γ s = Γ t" when st: "s ∈ SMP N" "t ∈ M"
  proof -
    obtain c where "t = Fun c []" using st assms(1) by moura
    hence "s = t" using *(3) *(1,2) by auto
    thus ?thesis by metis
  qed
  moreover have "Γ s = Γ t" when st: "s ∈ M" "t ∈ SMP N"
  proof -
    obtain c where "s = Fun c []" using st assms(1) by moura
    hence "s = t" using *(3) *(1,2) by auto
    thus ?thesis by metis
  qed
  ultimately have "Γ s = Γ t" by metis
} thus ?thesis by (metis tfrset-def)
qed

lemma dualst-tfrstp: "list_all tfrstp S ⇒ list_all tfrstp (dualst S)"
proof (induction S)
  case (Cons x S)
  have "list_all tfrstp S" using Cons.prem by simp
  hence IH: "list_all tfrstp (dualst S)" using Cons.IH by metis
  from Cons show ?case
  proof (cases x)
    case (Equality a t t')
    hence "(∃δ. Unifier δ t t') ⇒ Γ t = Γ t'" using Cons by auto
    thus ?thesis using Equality IH by fastforce
  next
    case (Inequality X F)
    have "set (dualst (x#S)) = insert x (set (dualst S))" using Inequality by auto
    moreover have "(∀x ∈ fvpairs F - set X. ∃a. Γ (Var x) = Var a) ∨
      (∀f T. Fun f T ∈ subtermsset (trmspairs F) → T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X))"
      using Cons.prem Inequality by auto
    ultimately show ?thesis using Inequality IH by auto
  qed auto
qed simp

lemma subst_var_inv_wt:
  assumes "wtsubst δ"
  shows "wtsubst (subst_var_inv δ X)"
using assms f_inv_into_f[of _ δ X]
unfolding wtsubst-def subst_var_inv_def
by presburger

lemma subst_var_inv_wf_trms:
  "wftrms (subst_range (subst_var_inv δ X))"
using f_inv_into_f[of _ δ X]
unfolding wtsubst-def subst_var_inv_def
by auto

lemma unify_list_wt_if_same_type:
  assumes "Unification.unify E B = Some U" "∀(s,t) ∈ set E. wftrm s ∧ wftrm t ∧ Γ s = Γ t"
  and "∀(v,t) ∈ set B. Γ (Var v) = Γ t"
  shows "∀(v,t) ∈ set U. Γ (Var v) = Γ t"
using assms
proof (induction E B arbitrary: U rule: Unification.unify.induct)
  case (2 f X g Y E B U)
  hence "wftrm (Fun f X)" "wftrm (Fun g Y)" "Γ (Fun f X) = Γ (Fun g Y)" by auto

  from "2.prem"(1) obtain E' where *: "decompose (Fun f X) (Fun g Y) = Some E'"
  and [simp]: "f = g" "length X = length Y" "E' = zip X Y"

```

```

and **: "Unification.unify (E'@E) B = Some U"
by (auto split: option.splits)

have "∀ (s,t) ∈ set E'. wf_trm s ∧ wf_trm t ∧ Γ s = Γ t"
proof -
  { fix s t assume "(s,t) ∈ set E'"
    then obtain X' X'' Y' Y'' where "X = X'@s#X''" "Y = Y'@t#Y''" "length X' = length Y'"
    using zip_arg_subterm_split[of s t X Y] ⟨E' = zip X Y⟩ by metis
    hence "Γ (Fun f (X'@s#X'')) = Γ (Fun g (Y'@t#Y''))" by (metis (Γ (Fun f X) = Γ (Fun g Y)))

    from ⟨E' = zip X Y⟩ have "∀ (s,t) ∈ set E'. s ⊆ Fun f X ∧ t ⊆ Fun g Y"
    using zip_arg_subterm[of _ _ X Y] by blast
    with ⟨(s,t) ∈ set E'⟩ have "wf_trm s" "wf_trm t"
    using wf_trm_subterm ⟨wf_trm (Fun f X)⟩ ⟨wf_trm (Fun g Y)⟩ by (blast,blast)
    moreover have "f ∈ Σ_f"
    proof (rule ccontr)
      assume "f ∉ Σ_f"
      hence "f ∈ C" "arity f = 0" using const_arity_eq_zero[of f] by simp_all
      thus False using ⟨wf_trm (Fun f X)⟩ * ⟨(s,t) ∈ set E'⟩ unfolding wf_trm_def by auto
    qed
    hence "Γ s = Γ t"
    using fun_type_arg_cong' ⟨f ∈ Σ_f⟩ (Γ (Fun f (X'@s#X'')) = Γ (Fun g (Y'@t#Y'')))
    ⟨length X' = length Y'⟩ ⟨f = g⟩
    by metis
    ultimately have "wf_trm s" "wf_trm t" "Γ s = Γ t" by metis+
  }
  thus ?thesis by blast
qed
moreover have "∀ (s,t) ∈ set E. wf_trm s ∧ wf_trm t ∧ Γ s = Γ t" using "2.prem" (2) by auto
ultimately show ?case using "2.IH"[OF * ** _ "2.prem" (3)] by fastforce
next
case (3 v t E B U)
hence "Γ (Var v) = Γ t" "wf_trm t" by auto
hence "wt_subst (subst v t)"
  and *: "∀ (v, t) ∈ set ((v,t)#B). Γ (Var v) = Γ t"
  "∧ t t'. (t,t') ∈ set E ⇒ Γ t = Γ t'"
using "3.prem" (2,3) unfolding wt_subst_def subst_def by auto

show ?case
proof (cases "t = Var v")
  assume "t = Var v" thus ?case using 3 by auto
next
  assume "t ≠ Var v"
  hence "v ∉ fv t" using "3.prem" (1) by auto
  hence **: "Unification.unify (subst_list (subst v t) E) ((v, t)#B) = Some U"
  using Unification.unify.simps (3) [of v t E B] "3.prem" (1) ⟨t ≠ Var v⟩ by auto

  have "∀ (s, t) ∈ set (subst_list (subst v t) E). wf_trm s ∧ wf_trm t"
  using wf_trm_subst_singleton[OF _ ⟨wf_trm t⟩] "3.prem" (2)
  unfolding subst_list_def subst_def by auto
  moreover have "∀ (s, t) ∈ set (subst_list (subst v t) E). Γ s = Γ t"
  using * (2) [THEN wt_subst_trm' [OF ⟨wt_subst (subst v t)⟩]] by (simp add: subst_list_def)
  ultimately show ?thesis using "3.IH" (2) [OF ⟨t ≠ Var v⟩ ⟨v ∉ fv t⟩ ** _ * (1)] by auto
qed
next
case (4 f X v E B U)
hence "Γ (Var v) = Γ (Fun f X)" "wf_trm (Fun f X)" by auto
hence "wt_subst (subst v (Fun f X))"
  and *: "∀ (v, t) ∈ set ((v,(Fun f X))#B). Γ (Var v) = Γ t"
  "∧ t t'. (t,t') ∈ set E ⇒ Γ t = Γ t'"
using "4.prem" (2,3) unfolding wt_subst_def subst_def by auto

have "v ∉ fv (Fun f X)" using "4.prem" (1) by force

```

```

hence **: "Unification.unify (subst_list (subst v (Fun f X)) E) ((v, (Fun f X))#B) = Some U"
  using Unification.unify.simps(3)[of v "Fun f X" E B] "4.premis"(1) by auto

have "∀ (s, t) ∈ set (subst_list (subst v (Fun f X)) E). wf_trm s ∧ wf_trm t"
  using wf_trm_subst_singleton[OF _ ⟨wf_trm (Fun f X)⟩] "4.premis"(2)
  unfolding subst_list_def subst_def by auto
moreover have "∀ (s, t) ∈ set (subst_list (subst v (Fun f X)) E). Γ s = Γ t"
  using *(2)[THEN wt_subst_trm'[OF ⟨wt_subst (subst v (Fun f X))⟩]] by (simp add: subst_list_def)
ultimately show ?case using "4.IH"[OF ⟨v ∉ fv (Fun f X)⟩ ** _ *(1)] by auto
qed auto

lemma mgu_wt_if_same_type:
  assumes "mgu s t = Some σ" "wf_trm s" "wf_trm t" "Γ s = Γ t"
  shows "wt_subst σ"
proof -
  let ?fv_disj = "λv t S. ¬(∃ (v', t') ∈ S - {(v, t)}. (insert v (fv t)) ∩ (insert v' (fv t')) ≠ {})"
  from assms(1) obtain σ' where "Unification.unify [(s, t)] [] = Some σ'" "subst_of σ' = σ"
    by (auto split: option.splits)
  hence "∀ (v, t) ∈ set σ'. Γ (Var v) = Γ t" "distinct (map fst σ')"
    using assms(2,3,4) unify_list_wt_if_same_type unify_list_distinct[of "[s, t]"] by auto
  thus "wt_subst σ" using ⟨subst_of σ' = σ⟩ unfolding wt_subst_def
  proof (induction σ' arbitrary: σ rule: List.rev_induct)
    case (snoc tt σ')
    then obtain v t where tt = (v, t) by (metis surj_pair)
    hence σ: "σ = subst v t ∘s subst_of σ'" using snoc.premis(3) by simp

    have "∀ (v, t) ∈ set σ'. Γ (Var v) = Γ t" "distinct (map fst σ'" using snoc.premis(1,2) by auto
    then obtain σ'' where σ'': "subst_of σ' = σ''" "∀ v. Γ (Var v) = Γ (σ'' v)" by (metis snoc.IH)
    hence "Γ t = Γ (t · σ'')" for t using wt_subst_trm by blast
    hence "Γ (Var v) = Γ (σ'' v)" "Γ t = Γ (t · σ'')" using σ''(2) by auto
    moreover have "Γ (Var v) = Γ t" using snoc.premis(1) tt by simp
    moreover have σ2: "σ = Var(v := t) ∘s σ'" using σ σ''(1) unfolding subst_def by simp
    ultimately have "Γ (Var v) = Γ (σ v)" unfolding subst_compose_def by simp

    have "subst_domain (subst v t) ⊆ {v}" unfolding subst_def by (auto simp add: subst_domain_def)
    hence *: "subst_domain σ ⊆ insert v (subst_domain σ'')"
      using tt σ σ''(1) snoc.premis(2) subst_domain_compose[of _ σ'']
      by (auto simp add: subst_domain_def)

    have "v ∉ set (map fst σ'" using tt snoc.premis(2) by auto
    hence "v ∉ subst_domain σ'" using σ''(1) subst_of_dom_subset[of σ'] by auto

    { fix w assume "w ∈ subst_domain σ'"
      hence "σ w = σ'' w" using σ2 σ''(1) ⟨v ∉ subst_domain σ''⟩ unfolding subst_compose_def by
    auto
    hence "Γ (Var w) = Γ (σ w)" using σ''(2) by simp
    }
    thus ?case using ⟨Γ (Var v) = Γ (σ v)⟩ * by force
  qed simp
qed

lemma wt_Unifier_if_Unifier:
  assumes s_t: "wf_trm s" "wf_trm t" "Γ s = Γ t"
  and δ: "Unifier δ s t"
  shows "∃ ϑ. Unifier ϑ s t ∧ wt_subst ϑ ∧ wf_trms (subst_range ϑ)"
  using mgu_always_unifies[OF δ] mgu_gives_MGU[THEN MGU_is_Unifier[of s _ t]]
  mgu_wt_if_same_type[OF _ s_t] mgu_wf_trm[OF _ s_t(1,2)] wf_trm_subst_range_iff
  by fast

end

```



### 3.3.3 Automatically Proving Type-Flaw Resistance

#### Definitions: Variable Renaming

abbreviation "max\_var t  $\equiv$  Max (insert 0 (snd 'fv t))"

abbreviation "max\_var\_set X  $\equiv$  Max (insert 0 (snd 'X))"

definition "var\_rename n v  $\equiv$  Var (fst v, snd v + Suc n)"

definition "var\_rename\_inv n v  $\equiv$  Var (fst v, snd v - Suc n)"

#### Definitions: Computing a Finite Representation of the Sub-Message Patterns

A sufficient requirement for a term to be a well-typed instance of another term

definition is\_wt\_instance\_of\_cond where

```
"is_wt_instance_of_cond  $\Gamma$  t s  $\equiv$  (
   $\Gamma$  t =  $\Gamma$  s  $\wedge$  (case mgu t s of
    None  $\Rightarrow$  False
  | Some  $\delta \Rightarrow$  inj_on  $\delta$  (fv t)  $\wedge$  ( $\forall x \in$  fv t. is_Var ( $\delta$  x))))"
```

definition has\_all\_wt\_instances\_of where

```
"has_all_wt_instances_of  $\Gamma$  N M  $\equiv$   $\forall t \in N. \exists s \in M. \text{is\_wt\_instance\_of\_cond } \Gamma$  t s"
```

This function computes a finite representation of the set of sub-message patterns

definition SMP0 where

```
"SMP0 Ana  $\Gamma$  M  $\equiv$  let
  f =  $\lambda t. \text{Fun (the\_Fun } (\Gamma$  t)) (map Var (zip (args ( $\Gamma$  t)) [0..\Gamma t))])));
  g =  $\lambda M'. \text{map f (filter } (\lambda t. \text{is\_Var } t \wedge \text{is\_Fun } (\Gamma$  t)) M') @
    concat (map (fst  $\circ$  Ana) M') @ concat (map subterms_list M');
  h = remdups  $\circ$  g
  in while ( $\lambda A. \text{set (h A)} \neq \text{set A}$ ) h M"
```

These definitions are useful to refine an SMP representation set

fun generalize\_term where

```
"generalize_term _ _ n (Var x) = (Var x, n)"
| "generalize_term  $\Gamma$  p n (Fun f T) = (let  $\tau = \Gamma$  (Fun f T)
  in if p  $\tau$  then (Var ( $\tau$ , n), Suc n)
  else let (T', n') = foldr ( $\lambda t (S, m). \text{let (t', m') = generalize\_term } \Gamma$  p m t in (t' # S, m'))
    T ([], n)
  in (Fun f T', n'))"
```

definition generalize\_terms where

```
"generalize_terms  $\Gamma$  p  $\equiv$  map (fst  $\circ$  generalize_term  $\Gamma$  p 0)"
```

definition remove\_superfluous\_terms where

```
"remove_superfluous_terms  $\Gamma$  T  $\equiv$ 
  let
    f =  $\lambda S t R. \exists s \in \text{set } S - R. s \neq t \wedge \text{is\_wt\_instance\_of\_cond } \Gamma$  t s;
    g =  $\lambda S t (U, R). \text{if } f S t R \text{ then (U, insert t R) else (t \# U, R)}$ ;
    h =  $\lambda S. \text{remdups (fst (foldr (g S) S ([], \{\})))}$ 
  in while ( $\lambda S. h S \neq S$ ) h T"
```

#### Definitions: Checking Type-Flaw Resistance

definition is\_TComp\_var\_instance\_closed where

```
"is_TComp_var_instance_closed  $\Gamma$  M  $\equiv$   $\forall x \in \text{fv}_{\text{set}} (\text{set } M). \text{is\_Fun } (\Gamma$  (Var x))  $\longrightarrow$ 
  list_ex ( $\lambda t. \text{is\_Fun } t \wedge \Gamma$  t =  $\Gamma$  (Var x)  $\wedge$  list_all is_Var (args t)  $\wedge$  distinct (args t)) M"
```

definition finite\_SMP\_representation where

```
"finite_SMP_representation arity Ana  $\Gamma$  M  $\equiv$ 
  list_all (wftrm' arity) M  $\wedge$ 
  has_all_wt_instances_of  $\Gamma$  (subtermsset (set M)) (set M)  $\wedge$ 
  has_all_wt_instances_of  $\Gamma$  ( $\bigcup$  ((set  $\circ$  fst  $\circ$  Ana) ' set M)) (set M)  $\wedge$ 
  is_TComp_var_instance_closed  $\Gamma$  M"
```

definition `comp_tfrset` where

```
"comp_tfrset arity Ana  $\Gamma$  M  $\equiv$ 
  finite_SMP_representation arity Ana  $\Gamma$  M  $\wedge$ 
  (let  $\delta = \text{var\_rename } (\text{max\_var\_set } (\text{fv}_{\text{set}} (\text{set } M)))$ 
    in  $\forall s \in \text{set } M. \forall t \in \text{set } M. \text{is\_Fun } s \wedge \text{is\_Fun } t \wedge \Gamma s \neq \Gamma t \longrightarrow \text{mgu } s (t \cdot \delta) = \text{None})"$ 
```

fun `comp_tfrstp` where

```
"comp_tfrstp  $\Gamma$  ( $\langle \_ : t \doteq t' \rangle_{\text{st}}$ ) = (mgu t t'  $\neq$  None  $\longrightarrow \Gamma t = \Gamma t'$ )"
| "comp_tfrstp  $\Gamma$  ( $\forall X \langle \forall \neq : F \rangle_{\text{st}}$ ) = (
  ( $\forall x \in \text{fv}_{\text{pairs}} F - \text{set } X. \text{is\_Var } (\Gamma (\text{Var } x))) \vee$ 
  ( $\forall u \in \text{subterms}_{\text{set}} (\text{trms}_{\text{pairs}} F).
    \text{is\_Fun } u \longrightarrow (\text{args } u = [] \vee (\exists s \in \text{set } (\text{args } u). s \notin \text{Var ' set } X)))$ )"
| "comp_tfrstp _ _ = True"
```

definition `comp_tfrst` where

```
"comp_tfrst arity Ana  $\Gamma$  M S  $\equiv$ 
  list_all (comp_tfrstp  $\Gamma$ ) S  $\wedge$ 
  list_all (wftrm' arity) (trms_listst S)  $\wedge$ 
  has_all_wt_instances_of  $\Gamma$  (trmsst S) (set M)  $\wedge$ 
  comp_tfrset arity Ana  $\Gamma$  M"
```

### Small Lemmata

lemma `less_Suc_max_var_set`:

```
assumes z: "z  $\in$  X"
  and X: "finite X"
shows "snd z < Suc (max_var_set X)"
```

proof -

```
have "snd z  $\in$  snd ' X" using z by simp
hence "snd z  $\leq$  Max (insert 0 (snd ' X))" using X by simp
thus ?thesis using X by simp
```

qed

lemma (in `typed_model`) `finite_SMP_representationD`:

```
assumes "finite_SMP_representation arity Ana  $\Gamma$  M"
shows "wftrms (set M)"
  and "has_all_wt_instances_of  $\Gamma$  (subtermsset (set M)) (set M)"
  and "has_all_wt_instances_of  $\Gamma$  ( $\bigcup ((\text{set} \circ \text{fst} \circ \text{Ana}) ' \text{set } M)) (\text{set } M)$ "
  and "is_TComp_var_instance_closed  $\Gamma$  M"
```

using `assms unfolding finite_SMP_representation_def list_all_iff wftrm_code by blast+`

lemma (in `typed_model`) `is_wt_instance_of_condD`:

```
assumes t_instance_s: "is_wt_instance_of_cond  $\Gamma$  t s"
obtains  $\delta$  where
  " $\Gamma t = \Gamma s$ " "mgu t s = Some  $\delta$ "
  "inj_on  $\delta$  (fv t)" " $\delta ' (\text{fv } t) \subseteq \text{range Var}$ "
```

using `t_instance_s unfolding is_wt_instance_of_cond_def Let_def by (cases "mgu t s") fastforce+`

lemma (in `typed_model`) `is_wt_instance_of_condD'`:

```
assumes t_wf_trm: "wftrm t"
  and s_wf_trm: "wftrm s"
  and t_instance_s: "is_wt_instance_of_cond  $\Gamma$  t s"
shows " $\exists \delta. \text{wt}_{\text{subst}} \delta \wedge \text{wf}_{\text{trms}} (\text{subst\_range } \delta) \wedge t = s \cdot \delta$ "
```

proof -

```
obtain  $\delta$  where s:
  " $\Gamma t = \Gamma s$ " "mgu t s = Some  $\delta$ "
  "inj_on  $\delta$  (fv t)" " $\delta ' (\text{fv } t) \subseteq \text{range Var}$ "
by (metis is_wt_instance_of_condD[OF t_instance_s])
```

```
have 0: "wftrm t" "wftrm s" using s(1) t_wf_trm s_wf_trm by auto
```

```
note 1 = mgu_wt_if_same_type[OF s(2) 0 s(1)]
```

```

note 2 = conjunct1[OF mgu_gives_MGU[OF s(2)]]

show ?thesis
  using s(1) inj_var_ran_unifiable_has_subst_match[OF 2 s(3,4)]
    wt_subst_compose[OF 1 subst_var_inv_wt[OF 1, of "fv t"]]
    wf_trms_subst_compose[OF mgu_wf_trms[OF s(2) 0] subst_var_inv_wf_trms[of  $\delta$  "fv t"]]
  by auto
qed

lemma (in typed_model) is_wt_instance_of_condD'':
  assumes s_wf_trm: "wf_trm s"
  and t_instance_s: "is_wt_instance_of_cond  $\Gamma$  t s"
  and t_var: "t = Var x"
  shows " $\exists y. s = \text{Var } y \wedge \Gamma (\text{Var } y) = \Gamma (\text{Var } x)$ "
proof -
  obtain  $\delta$  where  $\delta$ : "wt_subst  $\delta$ " and s: "Var x = s .  $\delta$ "
  using is_wt_instance_of_condD'[OF _ s_wf_trm t_instance_s] t_var by auto
  obtain y where y: "s = Var y" using s by (cases s) auto
  show ?thesis using wt_subst_trm''[OF  $\delta$ ] s y by metis
qed

lemma (in typed_model) has_all_wt_instances_ofD:
  assumes N_instance_M: "has_all_wt_instances_of  $\Gamma$  N M"
  and t_in_N: "t  $\in$  N"
  obtains s  $\delta$  where
    "s  $\in$  M" "t =  $\Gamma$  s" "mgu t s = Some  $\delta$ "
    "inj_on  $\delta$  (fv t)" " $\delta$  ' (fv t)  $\subseteq$  range Var"
by (metis t_in_N N_instance_M is_wt_instance_of_condD has_all_wt_instances_of_def)

lemma (in typed_model) has_all_wt_instances_ofD':
  assumes N_wf_trms: "wf_trms N"
  and M_wf_trms: "wf_trms M"
  and N_instance_M: "has_all_wt_instances_of  $\Gamma$  N M"
  and t_in_N: "t  $\in$  N"
  shows " $\exists \delta. \text{wt\_subst } \delta \wedge \text{wf\_trms } (\text{subst\_range } \delta) \wedge t \in M \cdot_{\text{set}} \delta$ "
using assms is_wt_instance_of_condD' unfolding has_all_wt_instances_of_def by fast

lemma (in typed_model) has_all_wt_instances_ofD'':
  assumes N_wf_trms: "wf_trms N"
  and M_wf_trms: "wf_trms M"
  and N_instance_M: "has_all_wt_instances_of  $\Gamma$  N M"
  and t_in_N: "Var x  $\in$  N"
  shows " $\exists y. \text{Var } y \in M \wedge \Gamma (\text{Var } y) = \Gamma (\text{Var } x)$ "
using assms is_wt_instance_of_condD' unfolding has_all_wt_instances_of_def by fast

lemma (in typed_model) has_all_instances_of_if_subset:
  assumes "N  $\subseteq$  M"
  shows "has_all_wt_instances_of  $\Gamma$  N M"
using assms inj_onI mgu_same_empty
unfolding has_all_wt_instances_of_def is_wt_instance_of_cond_def
by (smt option.case_eq_if option.discI option.sel subsetD term.discI(1) term.inject(1))

lemma (in typed_model) SMP_I':
  assumes N_wf_trms: "wf_trms N"
  and M_wf_trms: "wf_trms M"
  and N_instance_M: "has_all_wt_instances_of  $\Gamma$  N M"
  and t_in_N: "t  $\in$  N"
  shows "t  $\in$  SMP M"
using has_all_wt_instances_ofD'[OF N_wf_trms M_wf_trms N_instance_M t_in_N]
  SMP.Substitution[OF SMP.MP[of _ M]]
by blast

```

**Lemma: Proving Type-Flaw Resistance**

```

locale typed_model' = typed_model arity public Ana  $\Gamma$ 
  for arity::"'fun  $\Rightarrow$  nat"
  and public::"'fun  $\Rightarrow$  bool"
  and Ana::"'(fun, (('fun, 'atom::finite) term_type  $\times$  nat)) term
     $\Rightarrow$  (('fun, (('fun, 'atom) term_type  $\times$  nat)) term list
       $\times$  ('fun, (('fun, 'atom) term_type  $\times$  nat)) term list)"
  and  $\Gamma$ ::"'(fun, (('fun, 'atom) term_type  $\times$  nat)) term  $\Rightarrow$  ('fun, 'atom) term_type"
+
  assumes  $\Gamma\_Var\_fst$ : " $\bigwedge \tau \ n \ m. \ \Gamma \ (Var \ (\tau, n)) = \Gamma \ (Var \ (\tau, m))$ "
  and Ana_const: " $\bigwedge c \ T. \ arity \ c = 0 \implies Ana \ (Fun \ c \ T) = ([], [])$ "
  and Ana_subst'_or_Ana_keys_subterm:
    " $(\forall f \ T \ \delta \ K \ R. \ Ana \ (Fun \ f \ T) = (K, R) \longrightarrow Ana \ (Fun \ f \ T \cdot \delta) = (K \cdot_{list} \delta, R \cdot_{list} \delta)) \vee$ 
 $(\forall t \ K \ R \ k. \ Ana \ t = (K, R) \longrightarrow k \in set \ K \longrightarrow k \sqsubseteq t)$ "
begin

lemma var_rename_inv_comp: " $t \cdot (var\_rename \ n \circ_s \ var\_rename\_inv \ n) = t$ "
proof (induction t)
  case (Fun f T)
  hence "map ( $\lambda t. \ t \cdot var\_rename \ n \circ_s \ var\_rename\_inv \ n$ )  $T = T$ " by (simp add: map_idI)
  thus ?case by (metis subst_apply_term.simps(2))
qed (simp add: var_rename_def var_rename_inv_def)

lemma var_rename_fv_disjoint:
  " $fv \ s \cap fv \ (t \cdot var\_rename \ (max\_var \ s)) = \{\}$ "
proof -
  have 1: " $\forall v \in fv \ s. \ snd \ v \leq max\_var \ s$ " by simp
  have 2: " $\forall v \in fv \ (t \cdot var\_rename \ n). \ snd \ v > n$ " for n unfolding var_rename_def by (induct t) auto
  show ?thesis using 1 2 by force
qed

lemma var_rename_fv_set_disjoint:
  assumes "finite M" "s  $\in$  M"
  shows " $fv \ s \cap fv \ (t \cdot var\_rename \ (max\_var\_set \ (fv_{set} \ M))) = \{\}$ "
proof -
  have 1: " $\forall v \in fv \ s. \ snd \ v \leq max\_var\_set \ (fv_{set} \ M)$ " using assms
  proof (induction M rule: finite_induct)
    case (insert t M) thus ?case
      proof (cases "t = s")
        case False
        hence " $\forall v \in fv \ s. \ snd \ v \leq max\_var\_set \ (fv_{set} \ M)$ " using insert by simp
        moreover have " $max\_var\_set \ (fv_{set} \ M) \leq max\_var\_set \ (fv_{set} \ (insert \ t \ M))$ "
          using insert.hyps(1) insert.premis
          by force
        ultimately show ?thesis by auto
      qed simp
    qed simp
  qed

  have 2: " $\forall v \in fv \ (t \cdot var\_rename \ n). \ snd \ v > n$ " for n unfolding var_rename_def by (induct t) auto

  show ?thesis using 1 2 by force
qed

lemma var_rename_fv_set_disjoint':
  assumes "finite M"
  shows " $fv_{set} \ M \cap fv_{set} \ (N \cdot_{set} \ var\_rename \ (max\_var\_set \ (fv_{set} \ M))) = \{\}$ "
using var_rename_fv_set_disjoint[OF assms] by auto

lemma var_rename_is_renaming[simp]:
  " $subst\_range \ (var\_rename \ n) \subseteq range \ Var$ "
  " $subst\_range \ (var\_rename\_inv \ n) \subseteq range \ Var$ "
unfolding var_rename_def var_rename_inv_def by auto

```

```

lemma var_rename_wt[simp]:
  "wt_subst (var_rename n)"
  "wt_subst (var_rename_inv n)"
by (auto simp add: var_rename_def var_rename_inv_def wt_subst_def  $\Gamma$ _Var_fst)

lemma var_rename_wt':
  assumes "wt_subst  $\delta$ " "s = m  $\cdot$   $\delta$ "
  shows "wt_subst (var_rename_inv n  $\circ_s$   $\delta$ )" "s = m  $\cdot$  var_rename n  $\cdot$  var_rename_inv n  $\circ_s$   $\delta$ "
using assms(2) wt_subst_compose[OF var_rename_wt(2)[of n] assms(1)] var_rename_inv_comp[of m n]
by force+

lemma var_rename_wf_trms_range[simp]:
  "wf_trms (subst_range (var_rename n))"
  "wf_trms (subst_range (var_rename_inv n))"
using var_rename_is_renaming by fastforce+

lemma Fun_range_case:
  "( $\forall f T. \text{Fun } f T \in M \longrightarrow P f T$ )  $\longleftrightarrow$  ( $\forall u \in M. \text{case } u \text{ of } \text{Fun } f T \Rightarrow P f T \mid \_ \Rightarrow \text{True}$ )"
  "( $\forall f T. \text{Fun } f T \in M \longrightarrow P f T$ )  $\longleftrightarrow$  ( $\forall u \in M. \text{is\_Fun } u \longrightarrow P (\text{the\_Fun } u) (\text{args } u)$ )"
by (auto split: "term.splits")

lemma is_TComp_var_instance_closedD:
  assumes x: " $\exists y \in \text{fv}_{\text{set}} (\text{set } M). \Gamma (\text{Var } x) = \Gamma (\text{Var } y)$ " " $\Gamma (\text{Var } x) = \text{TComp } f T$ "
  and closed: "is_TComp_var_instance_closed  $\Gamma M$ "
  shows " $\exists g U. \text{Fun } g U \in \text{set } M \wedge \Gamma (\text{Fun } g U) = \Gamma (\text{Var } x) \wedge (\forall u \in \text{set } U. \text{is\_Var } u) \wedge \text{distinct } U$ "
using assms unfolding is_TComp_var_instance_closed_def list_all_iff list_ex_iff by fastforce

lemma is_TComp_var_instance_closedD':
  assumes " $\exists y \in \text{fv}_{\text{set}} (\text{set } M). \Gamma (\text{Var } x) = \Gamma (\text{Var } y)$ " " $\text{TComp } f T \sqsubseteq \Gamma (\text{Var } x)$ "
  and closed: "is_TComp_var_instance_closed  $\Gamma M$ "
  and wf: "wf_trms (set M)"
  shows " $\exists g U. \text{Fun } g U \in \text{set } M \wedge \Gamma (\text{Fun } g U) = \text{TComp } f T \wedge (\forall u \in \text{set } U. \text{is\_Var } u) \wedge \text{distinct } U$ "
using assms(1,2)
proof (induction " $\Gamma (\text{Var } x)$ " arbitrary: x)
  case (Fun g U)
  note IH = Fun.hyps(1)
  have g: "arity g > 0" "public g" using Fun.hyps(2) fun_type_inv[of "Var x"]  $\Gamma$ _Var_fst by simp_all
  then obtain V where V:
    "Fun g V  $\in$  set M" " $\Gamma (\text{Fun } g V) = \Gamma (\text{Var } x)$ " " $\forall v \in \text{set } V. \exists x. v = \text{Var } x$ "
    "distinct V" "length U = length V"
  using is_TComp_var_instance_closedD[OF Fun.prem(1) Fun.hyps(2)[symmetric] closed(1)]
  by (metis Fun.hyps(2) fun_type_id_eq fun_type_length_eq is_VarE)
  hence U: "U = map  $\Gamma$  V" using fun_type[OF g(1), of V] Fun.hyps(2) by simp
  hence 1: " $\Gamma v \in \text{set } U$ " when v: "v  $\in$  set V" for v using v by simp

  have 2: " $\exists y \in \text{fv}_{\text{set}} (\text{set } M). \Gamma (\text{Var } z) = \Gamma (\text{Var } y)$ " when z: "Var z  $\in$  set V" for z
  using V(1) fv_subset_subterms Fun_param_in_subterms[OF z] by fastforce

  show ?case
  proof (cases "TComp f T =  $\Gamma (\text{Var } x)$ ")
    case False
    then obtain u where u: "u  $\in$  set U" "TComp f T  $\sqsubseteq$  u"
    using Fun.prem(2) Fun.hyps(2) by mouna
    then obtain y where y: "Var y  $\in$  set V" " $\Gamma (\text{Var } y) = u$ " using U V(3)  $\Gamma$ _Var_fst by auto
    show ?thesis using IH[OF _ 2[OF y(1)]] u y(2) by metis
  qed (use V in fastforce)
qed simp

lemma TComp_var_instance_wt_subst_exists:
  assumes gT: " $\Gamma (\text{Fun } g T) = \text{TComp } g (\text{map } \Gamma U)$ " "wf_trm (Fun g T)"
  and U: " $\forall u \in \text{set } U. \exists y. u = \text{Var } y$ " "distinct U"
  shows " $\exists \vartheta. \text{wt\_subst } \vartheta \wedge \text{wf\_trms } (\text{subst\_range } \vartheta) \wedge \text{Fun } g T = \text{Fun } g U \cdot \vartheta$ "

```

proof -

```
define the_i where "the_i  $\equiv \lambda y. \text{THE } x. x < \text{length } U \wedge U ! x = \text{Var } y$ "
define  $\vartheta$  where  $\vartheta: " \vartheta \equiv \lambda y. \text{if } \text{Var } y \in \text{set } U \text{ then } T ! \text{the\_i } y \text{ else } \text{Var } y"$ 
```

```
have g: "arity g > 0" using gT(1,2) fun_type_inv(1) by blast
```

```
have UT: "length U = length T" using fun_type_length_eq gT(1) by fastforce
```

```
have 1: "the_i y < length U  $\wedge U ! \text{the\_i } y = \text{Var } y$ " when y: "Var y  $\in \text{set } U$ " for y
  using theI'[OF distinct_Ex1[OF U(2) y]] unfolding the_i_def by simp
```

```
have 2: "wt_subst  $\vartheta$ "
  using  $\vartheta$  1 gT(1) fun_type[OF g] UT
  unfolding wt_subst_def
  by (metis (no_types, lifting) nth_map term.inject(2))
```

```
have " $\forall i < \text{length } T. U ! i \cdot \vartheta = T ! i$ "
  using  $\vartheta$  1 U(1) UT distinct_Ex1[OF U(2)] in_set_conv_nth
  by (metis (no_types, lifting) subst_apply_term.simps(1))
hence "T = map ( $\lambda t. t \cdot \vartheta$ ) U" by (simp add: UT nth_equalityI)
hence 3: "Fun g T = Fun g U  $\cdot \vartheta$ " by simp
```

```
have "subst_range  $\vartheta \subseteq \text{set } T$ " using  $\vartheta$  1 U(1) UT by (auto simp add: subst_domain_def)
hence 4: "wf_trms (subst_range  $\vartheta$ )" using gT(2) wf_trm_param by auto
```

```
show ?thesis by (metis 2 3 4)
```

qed

lemma TComp\_var\_instance\_closed\_has\_Var:

```
assumes closed: "is_TComp_var_instance_closed  $\Gamma$  M"
  and wf_M: "wf_trms (set M)"
  and wf_ $\delta$ x: "wf_trm ( $\delta$  x)"
  and y_ex: " $\exists y \in \text{fv}_{\text{set}} (\text{set } M). \Gamma (\text{Var } x) = \Gamma (\text{Var } y)$ "
  and t: " $t \sqsubseteq \delta$  x"
  and  $\delta$ _wt: "wt_subst  $\delta$ "
shows " $\exists y \in \text{fv}_{\text{set}} (\text{set } M). \Gamma (\text{Var } y) = \Gamma t$ "
```

proof (cases " $\Gamma (\text{Var } x)$ ")

```
case (Var a)
hence "t =  $\delta$  x"
  using t wf_ $\delta$ x  $\delta$ _wt
  by (metis (full_types) const_type_inv_wf fun_if_subterm subterm_eq_Var_const(2) wt_subst_def)
thus ?thesis using y_ex wt_subst_trm'[OF  $\delta$ _wt, of "Var x"] by fastforce
```

next

```
case (Fun f T)
hence  $\Gamma_{\delta}$ x: " $\Gamma (\delta$  x) = TComp f T" using wt_subst_trm'[OF  $\delta$ _wt, of "Var x"] by auto
```

show ?thesis

proof (cases "t =  $\delta$  x")

```
case False
hence t_subt_ $\delta$ x: "t  $\sqsubset \delta$  x" using t(1)  $\Gamma_{\delta}$ x by fastforce
```

```
obtain T' where T': " $\delta$  x = Fun f T'" using  $\Gamma_{\delta}$ x t_subt_ $\delta$ x fun_type_id_eq by (cases " $\delta$  x") auto
```

```
obtain g S where gS: "Fun g S  $\sqsubseteq \delta$  x" "t  $\in \text{set } S$ " using Fun_ex_if_subterm[OF t_subt_ $\delta$ x] by blast
```

```
have gS_wf: "wf_trm (Fun g S)" by (rule wf_trm_subterm_eq[OF wf_ $\delta$ x gS(1)])
hence "arity g > 0" using gS(2) by (metis length_pos_if_in_set wf_trm_arity)
hence gS_ $\Gamma$ : " $\Gamma (\text{Fun } g S) = \text{TComp } g (\text{map } \Gamma S)$ " using fun_type by blast
```

obtain h U where hU:

```
"Fun h U  $\in \text{set } M$ " " $\Gamma (\text{Fun } h U) = \text{Fun } g (\text{map } \Gamma S)$ " " $\forall u \in \text{set } U. \text{is\_Var } u$ "
using is_TComp_var_instance_closedD'[OF y_ex _ closed wf_M]
  subterm_eq_imp_subterm_typeeq[OF wf_ $\delta$ x] gS  $\Gamma_{\delta}$ x Fun gS_ $\Gamma$ 
```

```

by metis

obtain y where y: "Var y ∈ set U" "Γ (Var y) = Γ t"
  using hU(3) fun_type_param_ex[OF hU(2) gS(2)] by fast

have "y ∈ fvset (set M)" using hU(1) y(1) by force
thus ?thesis using y(2) closed by metis
qed (metis y_ex Fun Γ_δx)
qed

lemma TComp_var_instance_closed_has_Fun:
  assumes closed: "is_TComp_var_instance_closed Γ M"
  and wf_M: "wftrms (set M)"
  and wf_δx: "wftrms (δ x)"
  and y_ex: "∃y ∈ fvset (set M). Γ (Var x) = Γ (Var y)"
  and t: "t ⊆ δ x"
  and δ_wt: "wtsubst δ"
  and t_Γ: "Γ t = TComp g T"
  and t_fun: "is_Fun t"
  shows "∃m ∈ set M. ∃ϑ. wtsubst ϑ ∧ wftrms (subst_range ϑ) ∧ t = m · ϑ ∧ is_Fun m"
proof -
  obtain T'' where T'': "t = Fun g T'" using t_Γ t_fun fun_type_id_eq by blast

  have g: "arity g > 0" using t_Γ fun_type_inv[of t] by simp_all

  have "TComp g T ⊆ Γ (Var x)" using δ_wt t t_Γ
  by (metis wf_δx subtermeq_imp_subtermtypreeq wtsubst_def)
  then obtain U where U:
    "Fun g U ∈ set M" "Γ (Fun g U) = TComp g T" "∀u ∈ set U. ∃y. u = Var y"
    "distinct U" "length T'' = length U"
  using is_TComp_var_instance_closedD'[OF y_ex _ closed wf_M]
  by (metis t_Γ T'' fun_type_id_eq fun_type_length_eq is_VarE)
  hence UT': "T = map Γ U" using fun_type[OF g, of U] by simp

  show ?thesis
  using TComp_var_instance_wt_subst_exists UT' T'' U(1,3,4) t t_Γ wf_δx wf_trm_subtermeq
  by (metis term.disc(2))
qed

lemma TComp_var_and_subterm_instance_closed_has_subterms_instances:
  assumes M_var_inst_cl: "is_TComp_var_instance_closed Γ M"
  and M_subterms_cl: "has_all_wt_instances_of Γ (subtermsset (set M)) (set M)"
  and M_wf: "wftrms (set M)"
  and t: "t ⊆set set M"
  and s: "s ⊆ t · δ"
  and δ: "wtsubst δ" "wftrms (subst_range δ)"
  shows "∃m ∈ set M. ∃ϑ. wtsubst ϑ ∧ wftrms (subst_range ϑ) ∧ s = m · ϑ"
using subterm_subst_unfold[OF s]
proof
  assume "∃s'. s' ⊆ t ∧ s = s' · δ"
  then obtain s' where s': "s' ⊆ t" "s = s' · δ" by blast
  then obtain ϑ where ϑ: "wtsubst ϑ" "wftrms (subst_range ϑ)" "s' ∈ set M ·set ϑ"
  using t has_all_wt_instances_ofD'[OF wf_trms_subterms[OF M_wf] M_wf M_subterms_cl]
  term.order_trans[of s' t]
  by blast
  then obtain m where m: "m ∈ set M" "s' = m · ϑ" by blast

  have "s = m · (ϑ ∘s δ)" using s'(2) m(2) by simp
  thus ?thesis
  using m(1) wt_subst_compose[OF ϑ(1) δ(1)] wf_trms_subst_compose[OF ϑ(2) δ(2)] by blast
next
  assume "∃x ∈ fv t. s ⊆ δ x"
  then obtain x where x: "x ∈ fv t" "s ⊆ δ x" "s ⊆ δ x" by blast

```

```

note 0 = TComp_var_instance_closed_has_Var[OF M_var_inst_cl M_wf]
note 1 = has_all_wt_instances_ofD'[OF wf_trms_subterms[OF M_wf] M_wf M_subterms_cl]

have  $\delta x_{wf}$ : " $wf_{trm} (\delta x)$ " and  $s_{wf\_trm}$ : " $wf_{trm} s$ "
  using  $\delta(2)$  wf_trm_subterm[OF _ x(2)] by fastforce+

have  $x_{fv\_ex}$ : " $\exists y \in fv_{set} (set M). \Gamma (Var x) = \Gamma (Var y)$ "
  using x(1) s_fv_subset_subterms[OF t] by auto

obtain y where y: " $y \in fv_{set} (set M)$ " " $\Gamma (Var y) = \Gamma s$ "
  using 0[of  $\delta x s$ , OF  $\delta x_{wf} x_{fv\_ex} x(3) \delta(1)$ ] by metis
then obtain z where z: " $Var z \in set M$ " " $\Gamma (Var z) = \Gamma s$ "
  using 1[of y] vars_iff_subtermeq_set[of y "set M"] by metis

define  $\vartheta$  where " $\vartheta \equiv Var(z := s)::('fun, 'atom) term \times nat$ ) subst"

have " $wt_{subst} \vartheta$ " " $wf_{trms} (subst\_range \vartheta)$ " " $s = Var z \cdot \vartheta$ "
  using z(2) s_wf_trm unfolding  $\vartheta\_def$  wt_subst_def by force+
thus ?thesis using z(1) by blast
qed

context
begin
private lemma SMP_D_aux1:
  assumes "t  $\in$  SMP (set M)"
    and closed: "has_all_wt_instances_of  $\Gamma$  (subtermsset (set M)) (set M)"
      "is_TComp_var_instance_closed  $\Gamma$  M"
    and wf_M: " $wf_{trms} (set M)$ "
  shows " $\forall x \in fv t. \exists y \in fv_{set} (set M). \Gamma (Var y) = \Gamma (Var x)$ "
using assms(1)
proof (induction t rule: SMP.induct)
  case (MP t) show ?case
  proof
    fix x assume x: " $x \in fv t$ "
    hence " $Var x \in subterms_{set} (set M)$ " using MP.hyps vars_iff_subtermeq by fastforce
    then obtain  $\delta s$  where  $\delta$ : " $wt_{subst} \delta$ " " $wf_{trms} (subst\_range \delta)$ "
      and s: " $s \in set M$ " " $Var x = s \cdot \delta$ "
      using has_all_wt_instances_ofD'[OF wf_trms_subterms[OF wf_M] wf_M closed(1)] by blast
    then obtain y where y: " $s = Var y$ " by (cases s) auto
    thus " $\exists y \in fv_{set} (set M). \Gamma (Var y) = \Gamma (Var x)$ "
      using s wt_subst_trm'[OF  $\delta(1)$ , of "Var y"] by force
  qed
next
  case (Subterm t t')
  hence " $fv t' \subseteq fv t$ " using subtermeq_vars_subset by auto
  thus ?case using Subterm.IH by auto
next
  case (Substitution t  $\delta$ )
  note IH = Substitution.IH
  show ?case
  proof
    fix x assume x: " $x \in fv (t \cdot \delta)$ "
    then obtain y where y: " $y \in fv t$ " " $\Gamma (Var x) \sqsubseteq \Gamma (Var y)$ "
      using Substitution.hyps(2,3)
      by (metis subst_apply_img_var subtermeqI' subtermeq_imp_subtermtypeeq
        vars_iff_subtermeq wt_subst_def wf_trm_subst_rangeD)
    let ?P = " $\lambda x. \exists y \in fv_{set} (set M). \Gamma (Var y) = \Gamma (Var x)$ "
    show "?P x" using y IH
    proof (induction " $\Gamma (Var y)$ " arbitrary: y t)
      case (Var a)
      hence " $\Gamma (Var x) = \Gamma (Var y)$ " by auto
      thus ?case using Var(2,4) by auto
    end
  end
end

```



```

next
  case (Fun f T)
  obtain z where z: "∃ w ∈ fvset (set M). Γ (Var z) = Γ (Var w)" "Γ (Var z) = Γ (Var y)"
    using Fun.prem(1,3) by blast
  show ?case
  proof (cases "Γ (Var x) = Γ (Var y)")
    case True thus ?thesis using Fun.prem by auto
  next
    case False
    then obtain τ where τ: "τ ∈ set T" "Γ (Var x) ⊆ τ" using Fun.prem(2) Fun.hyps(2) by auto
    then obtain U where U:
      "Fun f U ∈ set M" "Γ (Fun f U) = Γ (Var z)" "∀ u ∈ set U. ∃ v. u = Var v" "distinct U"
      using is_TComp_var_instance_closedD'[OF z(1) _ closed(2) wf_M] Fun.hyps(2) z(2)
      by (metis fun_type_id_eq subtermeqI' is_VarE)
    hence 1: "∀ x ∈ fv (Fun f U). ∃ y ∈ fvset (set M). Γ (Var y) = Γ (Var x)" by force

    have "arity f > 0" using U(2) z(2) Fun.hyps(2) fun_type_inv(1) by metis
    hence "Γ (Fun f U) = TComp f (map Γ U)" using fun_type by auto
    then obtain u where u: "Var u ∈ set U" "Γ (Var u) = τ"
      using τ(1) U(2,3) z(2) Fun.hyps(2) by auto
    show ?thesis
      using Fun.hyps(1)[of u "Fun f U"] u τ 1
      by force
  qed
qed
qed
next
  case (Ana t K T k)
  have "fv k ⊆ fv t" using Ana_keys_fv[OF Ana.hyps(2)] Ana.hyps(3) by auto
  thus ?case using Ana.IH by auto
qed

private lemma SMP_D_aux2:
  fixes t::('fun, ('fun, 'atom) term × nat) term"
  assumes t_SMP: "t ∈ SMP (set M)"
    and t_Var: "∃ x. t = Var x"
    and M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
  shows "∃ m ∈ set M. ∃ δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ"
proof -
  have M_wf: "wftrms (set M)"
    and M_var_inst_cl: "is_TComp_var_instance_closed Γ M"
    and M_subterms_cl: "has_all_wt_instances_of Γ (subtermsset (set M)) (set M)"
    and M_Ana_cl: "has_all_wt_instances_of Γ (⋃ ((set ∘ fst ∘ Ana) ' set M)) (set M)"
    using finite_SMP_representationD[OF M_SMP_repr] by blast+

  have M_Ana_wf: "wftrms (⋃ ((set ∘ fst ∘ Ana) ' set M))"
  proof
    fix k assume "k ∈ ⋃ ((set ∘ fst ∘ Ana) ' set M)"
    then obtain m where m: "m ∈ set M" "k ∈ set (fst (Ana m))" by force
    thus "wftrm k" using M_wf Ana_keys_wf'[of m "fst (Ana m)" _ k] surjective_pairing by blast
  qed

  note 0 = has_all_wt_instances_ofD'[OF wf_trms_subterms[OF M_wf] M_wf M_subterms_cl]
  note 1 = has_all_wt_instances_ofD'[OF M_Ana_wf M_wf M_Ana_cl]

  obtain x y where x: "t = Var x" and y: "y ∈ fvset (set M)" "Γ (Var y) = Γ (Var x)"
    using t_Var SMP_D_aux1[OF t_SMP M_subterms_cl M_var_inst_cl M_wf] by fastforce
  then obtain m δ where m: "m ∈ set M" "m · δ = Var y" and δ: "wtsubst δ"
    using 0[of "Var y"] vars_iff_subtermeq_set[of y "set M"] by fastforce
  obtain z where z: "m = Var z" using m(2) by (cases m) auto

  define ∅ where "∅ ≡ Var(z := Var x)::('fun, ('fun, 'atom) term × nat) subst"

```

```

have "Γ (Var z) = Γ (Var x)" using y(2) m(2) z wt_subst_trm''[OF δ, of m] by argo
hence "wt_subst ∅" "wf_trms (subst_range ∅)" unfolding ∅_def wt_subst_def by force+
moreover have "t = m · ∅" using x z unfolding ∅_def by simp
ultimately show ?thesis using m(1) by blast
qed

private lemma SMP_D_aux3:
  assumes hyps: "t' ⊆ t" and wf_t: "wf_trm t" and prems: "is_Fun t'"
  and IH:
    "((∃f. t = Fun f []) ∧ (∃m ∈ set M. ∃δ. wt_subst δ ∧ wf_trms (subst_range δ) ∧ t = m · δ)) ∨
    (∃m ∈ set M. ∃δ. wt_subst δ ∧ wf_trms (subst_range δ) ∧ t = m · δ ∧ is_Fun m)"
  and M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
  shows "((∃f. t' = Fun f []) ∧ (∃m ∈ set M. ∃δ. wt_subst δ ∧ wf_trms (subst_range δ) ∧ t' = m · δ))
  ∨
    (∃m ∈ set M. ∃δ. wt_subst δ ∧ wf_trms (subst_range δ) ∧ t' = m · δ ∧ is_Fun m)"
proof (cases "∃f. t = Fun f [] ∨ t' = Fun f []")
  case True
  have M_wf: "wf_trms (set M)"
  and M_var_inst_cl: "is_TComp_var_instance_closed Γ M"
  and M_subterms_cl: "has_all_wt_instances_of Γ (subterms_set (set M)) (set M)"
  and M_Ana_cl: "has_all_wt_instances_of Γ (⋃((set ∘ fst ∘ Ana) ' set M)) (set M)"
  using finite_SMP_representationD[OF M_SMP_repr] by blast+

  note 0 = has_all_wt_instances_ofD'[OF wf_trms_subterms[OF M_wf] M_wf M_subterms_cl]
  note 1 = TComp_var_instance_closed_has_Fun[OF M_var_inst_cl M_wf]
  note 2 = TComp_var_and_subterm_instance_closed_has_subterms_instances[
    OF M_var_inst_cl M_subterms_cl M_wf]

  have wf_t': "wf_trm t'" using hyps wf_t wf_trm_subterm by blast

  obtain c where "t = Fun c [] ∨ t' = Fun c []" using True by moura
  thus ?thesis
  proof
    assume c: "t' = Fun c []"
    show ?thesis
    proof (cases "∃f. t = Fun f []")
      case True
      hence "t = t'" using c hyps by force
      thus ?thesis using IH by fast
    next
      case False
      note F = this
      then obtain m δ where m: "m ∈ set M" "t = m · δ"
        and δ: "wt_subst δ" "wf_trms (subst_range δ)"
        using IH by blast

      show ?thesis using subterm_subst_unfold[OF hyps[unfolded m(2)]]
    proof
      assume "∃m'. m' ⊆ m ∧ t' = m' · δ"
      then obtain m' where m': "m' ⊆ m" "t' = m' · δ" by moura
      obtain n ∅ where n: "n ∈ set M" "m' = n · ∅" and ∅: "wt_subst ∅" "wf_trms (subst_range ∅)"
        using 0[of m'] m(1) m'(1) by blast
      have "t' = n · (∅ ∘s δ)" using m'(2) n(2) by auto
      thus ?thesis
        using c n(1) wt_subst_compose[OF ∅(1) δ(1)] wf_trms_subst_compose[OF ∅(2) δ(2)] by blast
    next
      assume "∃x ∈ fv m. t' ⊆ δ x"
      then obtain x where x: "x ∈ fv m" "t' ⊆ δ x" "t' ⊆ δ x" by moura
      have δx_wf: "wf_trm (δ x)" using δ(2) by fastforce

      have x_fv_ex: "∃y ∈ fv_set (set M). Γ (Var x) = Γ (Var y)" using x m by auto

      show ?thesis

```

```

proof (cases "Γ t'")
  case (Var a)
  show ?thesis
    using c m 2[OF _ hyps[unfolded m(2)] δ]
    by fast
next
  case (Fun g S)
  show ?thesis
    using c 1[of δ x t', OF δx_wf xfv_ex x(3) δ(1) Fun]
    by blast
qed
qed
qed
qed (use IH hyps in simp)
next
case False
note F = False
then obtain m δ where m:
  "m ∈ set M" "wt_subst δ" "t = m · δ" "is_Fun m" "wf_trms (subst_range δ)"
  using IH by moura
obtain f T where fT: "t' = Fun f T" "arity f > 0" "Γ t' = TComp f (map Γ T)"
  using F prems fun_type wf_trm_subtermeq[OF wf_t hyps]
  by (metis is_FunE length_greater_0_conv subtermeqI' wf_trm_def)

have closed: "has_all_wt_instances_of Γ (subterms_set (set M)) (set M)"
  "is_TComp_var_instance_closed Γ M"
  using M_SMP_repr unfolding finite_SMP_representation_def by metis+

have M_wf: "wf_trms (set M)"
  using finite_SMP_representationD[OF M_SMP_repr] by blast

show ?thesis
proof (cases "∃ x ∈ fv m. t' ⊆ δ x")
  case True
  then obtain x where x: "x ∈ fv m" "t' ⊆ δ x" by moura
  have 1: "x ∈ fv_set (set M)" using m(1) x(1) by auto
  have 2: "is_Fun (δ x)" using prems x(2) by auto
  have 3: "wf_trm (δ x)" using m(5) by (simp add: wf_trm_subst_rangeD)
  have "¬(∃ f. δ x = Fun f [])" using F x(2) by auto
  hence "∃ f T. Γ (Var x) = TComp f T" using 2 3 m(2)
    by (metis (no_types) fun_type is_FunE length_greater_0_conv subtermeqI' wf_trm_def wt_subst_def)
  moreover have "∃ f T. Γ t' = Fun f T"
    using False prems wf_trm_subtermeq[OF wf_t hyps]
    by (metis (no_types) fun_type is_FunE length_greater_0_conv subtermeqI' wf_trm_def)
  ultimately show ?thesis
    using TComp_var_instance_closed_has_Fun 1 x(2) m(2) prems closed 3 M_wf
    by metis
next
case False
then obtain m' where m': "m' ⊆ m" "t' = m' · δ" "is_Fun m'"
  using hyps m(3) subterm_subst_not_img_subterm
  by blast
then obtain ϑ m'' where ϑ: "wt_subst ϑ" "wf_trms (subst_range ϑ)" "m'' ∈ set M" "m' = m'' · ϑ"
  using m(1) has_all_wt_instances_ofD'[OF wf_trms_subterms[OF M_wf] M_wf closed(1)] by blast
hence t'_m'': "t' = m'' · ϑ ∘s δ" using m'(2) by fastforce

note ϑδ = wt_subst_compose[OF ϑ(1) m(2)] wf_trms_subst_compose[OF ϑ(2) m(5)]

show ?thesis
proof (cases "is_Fun m''")
  case True thus ?thesis using ϑ(3,4) m'(2,3) m(4) fT t'_m'' ϑδ by blast
next
case False

```

```

then obtain x where x: "m'' = Var x" by maura
hence "∃y ∈ fvset (set M). Γ (Var x) = Γ (Var y)" "t' ⊆ (∅ ∘s δ) x"
  "Γ (Var x) = Fun f (map Γ T)" "wftrm ((∅ ∘s δ) x)"
  using ∅δ t'_m'' ∅(3) fv_subset[OF ∅(3)] fT(3) subst_apply_term.simps(1)[of x "∅ ∘s δ"]
    wt_subst_trm''[OF ∅δ(1), of "Var x"]
  by (fastforce, blast, argo, fastforce)
thus ?thesis
  using x TComp_var_instance_closed_has_Fun[
    of M "∅ ∘s δ" x t' f "map Γ T", OF closed(2) M_wf _ _ _ ∅δ(1) fT(3) prems]
  by blast
qed
qed
qed

lemma SMP_D:
  assumes "t ∈ SMP (set M)" "is_Fun t"
  and M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
  shows "( (∃f. t = Fun f []) ∧ (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ) ) ∨
    (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ ∧ is_Fun m) "
proof -
  have wf_M: "wftrms (set M)"
  and closed: "has_all_wt_instances_of Γ (subtermsset (set M)) (set M)"
    "has_all_wt_instances_of Γ (⋃ ((set ∘ fst ∘ Ana) ' set M)) (set M)"
    "is_TComp_var_instance_closed Γ M"
  using finite_SMP_representationD[OF M_SMP_repr] by blast+

show ?thesis using assms(1,2)
proof (induction t rule: SMP.induct)
  case (MP t)
  moreover have "wtsubst Var" "wftrms (subst_range Var)" "t = t · Var" by simp_all
  ultimately show ?case by blast
next
  case (Subterm t t')
  hence t_fun: "is_Fun t" by auto
  note * = Subterm.hyps(2) SMP_wf_trm[OF Subterm.hyps(1) wf_M(1)]
    Subterm.prem Subterm.IH[OF t_fun] M_SMP_repr
  show ?case by (rule SMP_D_aux3[OF *])
next
  case (Substitution t δ)
  have wf: "wftrm t" by (metis Substitution.hyps(1) wf_M(1) SMP_wf_trm)
  hence wf': "wftrm (t · δ)" using Substitution.hyps(3) wf_trm_subst by blast
  show ?case
  proof (cases "Γ t")
    case (Var a)
    hence 1: "Γ (t · δ) = TAtom a" using Substitution.hyps(2) by (metis wt_subst_trm'')
    then obtain c where c: "t · δ = Fun c []"
    using TAtom_term_cases[OF wf' 1] Substitution.prem by fastforce
    hence "(∃x. t = Var x) ∨ t = t · δ" by (cases t) auto
    thus ?thesis
  proof
    assume t_Var: "∃x. t = Var x"
    then obtain x where x: "t = Var x" "δ x = Fun c []" "Γ (Var x) = TAtom a"
    using c 1 wt_subst_trm''[OF Substitution.hyps(2), of t] by force

    obtain m ∅ where m: "m ∈ set M" "t = m · ∅" and ∅: "wtsubst ∅" "wftrms (subst_range ∅)"
    using SMP_D_aux2[OF Substitution.hyps(1) t_Var M_SMP_repr] by maura

    have "m · (∅ ∘s δ) = Fun c []" using c m(2) by auto
    thus ?thesis
    using c m(1) wt_subst_compose[OF ∅(1) Substitution.hyps(2)]
      wf_trms_subst_compose[OF ∅(2) Substitution.hyps(3)]
    by metis
  qed (use c Substitution.IH in auto)

```

```

next
case (Fun f T)
hence 1: " $\Gamma (t \cdot \delta) = TComp\ f\ T$ " using Substitution.hyps(2) by (metis wt_subst_trm'')
have 2: " $\neg(\exists f. t = Fun\ f\ [])$ " using Fun TComp_term_cases[OF wf] by auto
obtain T'' where T'': " $t \cdot \delta = Fun\ f\ T''$ "
  using 1 2 fun_type_id_eq Fun Substitution.prem
  by fastforce
have f: "arity f > 0" "public f" using fun_type_inv[OF 1] by metis+

show ?thesis
proof (cases t)
case (Fun g U)
then obtain m  $\vartheta$  where m:
  " $m \in set\ M$ " " $wt_{subst}\ \vartheta$ " " $t = m \cdot \vartheta$ " " $is\_Fun\ m$ " " $wf_{trms}\ (subst\_range\ \vartheta)$ "
  using Substitution.IH Fun 2 by moura
have " $wt_{subst}\ (\vartheta \circ_s \delta)$ " " $t \cdot \delta = m \cdot (\vartheta \circ_s \delta)$ " " $wf_{trms}\ (subst\_range\ (\vartheta \circ_s \delta))$ "
  using wt_subst_compose[OF m(2) Substitution.hyps(2)] m(3)
  wf_trms_subst_compose[OF m(5) Substitution.hyps(3)]
  by auto
thus ?thesis using m(1,4) by metis
next
case (Var x)
then obtain y where y: " $y \in fv_{set}\ (set\ M)$ " " $\Gamma (Var\ y) = \Gamma (Var\ x)$ "
  using SMP_D_aux1[OF Substitution.hyps(1) closed(1,3) wf_M] Fun
  by moura
hence 3: " $\Gamma (Var\ y) = TComp\ f\ T$ " using Var Fun  $\Gamma\_Var\_fst$  by simp

obtain h V where V:
  " $Fun\ h\ V \in set\ M$ " " $\Gamma (Fun\ h\ V) = \Gamma (Var\ y)$ " " $\forall u \in set\ V. \exists z. u = Var\ z$ " "distinct V"
  by (metis is_VarE is_TComp_var_instance_closedD[OF _ 3 closed(3)] y(1))
moreover have "length T'' = length V" using 3 V(2) fun_type_length_eq 1 T'' by metis
ultimately have TV: " $T = map\ \Gamma\ V$ "
  by (metis fun_type[OF f(1)] 3 fun_type_id_eq term.inject(2))

obtain  $\vartheta$  where  $\vartheta$ : " $wt_{subst}\ \vartheta$ " " $wf_{trms}\ (subst\_range\ \vartheta)$ " " $t \cdot \delta = Fun\ h\ V \cdot \vartheta$ "
  using TComp_var_instance_wt_subst_exists 1 3 T'' TV V(2,3,4) wf'
  by (metis fun_type_id_eq)

have 9: " $\Gamma (Fun\ h\ V) = \Gamma (\delta\ x)$ " using y(2) Substitution.hyps(2) V(2) 1 3 Var by auto

show ?thesis using Var  $\vartheta$  9 V(1) by force
qed
qed
next
case (Ana t K T k)
have 1: " $is\_Fun\ t$ " using Ana.hyps(2,3) by auto
then obtain f U where U: " $t = Fun\ f\ U$ " by moura

have 2: " $fv\ k \subseteq fv\ t$ " using Ana_keys_fv[OF Ana.hyps(2)] Ana.hyps(3) by auto

have wf_t: " $wf_{trm}\ t$ "
  using SMP_wf_trm[OF Ana.hyps(1)] wf_trm_code wf_M
  by auto
hence wf_k: " $wf_{trm}\ k$ "
  using Ana_keys_wf'[OF Ana.hyps(2)] wf_trm_code Ana.hyps(3)
  by auto

have wf_M_keys: " $wf_{trms}\ (\bigcup((set \circ fst \circ Ana) \text{ ` } set\ M))$ "
proof
fix t assume "t  $\in (\bigcup((set \circ fst \circ Ana) \text{ ` } set\ M))$ "
then obtain s where s: " $s \in set\ M$ " " $t \in (set \circ fst \circ Ana)\ s$ " by blast
obtain K R where KR: " $Ana\ s = (K,R)$ " by (metis surj_pair)
hence "t  $\in set\ K$ " using s(2) by simp

```

```

    thus "wftrm t" using Ana_keys_wf' [OF KR] wf_M s(1) by blast
  qed

show ?case using Ana_subst'_or_Ana_keys_subterm
proof
  assume "∀ t K T k. Ana t = (K, T) → k ∈ set K → k ⊆ t"
  hence *: "k ⊆ t" using Ana.hyps(2,3) by auto
  show ?thesis by (rule SMP_D_aux3[OF * wf_t Ana.premis Ana.IH[OF 1] M_SMP_repr])
next
  assume Ana_subst':
    "∀ f T δ K M. Ana (Fun f T) = (K, M) → Ana (Fun f T · δ) = (K ·list δ, M ·list δ)"

  have "arity f > 0" using Ana_const[of f U] U Ana.hyps(2,3) by fastforce
  hence "U ≠ []" using wf_t U unfolding wf_trm_def by force
  then obtain m δ where m: "m ∈ set M" "wtsubst δ" "wftrms (subst_range δ)" "t = m · δ" "is_Fun
m"

    using Ana.IH[OF 1] U by auto
  hence "Ana (t · δ) = (K ·list δ, T ·list δ)" using Ana_subst' U Ana.hyps(2) by auto
  obtain Km Tm where Ana_m: "Ana m = (Km, Tm)" by moura
  hence "Ana (m · δ) = (Km ·list δ, Tm ·list δ)"
    using Ana_subst' U m(4) is_FunE[OF m(5)] Ana.hyps(2)
    by metis
  then obtain km where km: "km ∈ set Km" "k = km · δ" using Ana.hyps(2,3) m(4) by auto
  then obtain ∅ km' where ∅: "wtsubst ∅" "wftrms (subst_range ∅)"
    and km': "km' ∈ set M" "km = km' · ∅"
    using Ana_m m(1) has_all_wt_instances_ofD'[OF wf_M_keys wf_M closed(2), of km] by force

  have k∅δ: "k = km' · ∅ ∘s δ" "wtsubst (∅ ∘s δ)" "wftrms (subst_range (∅ ∘s δ))"
    using km(2) km' wt_subst_compose[OF ∅(1) m(2)] wf_trms_subst_compose[OF ∅(2) m(3)]
    by auto

  show ?case
  proof (cases "is_Fun km'")
    case True thus ?thesis using k∅δ km'(1) by blast
  next
    case False
    note F = False
    then obtain x where x: "km' = Var x" by auto
    hence 3: "x ∈ fvset (set M)" using fv_subset[OF km'(1)] by auto
    obtain kf kT where kf: "k = Fun kf kT" using Ana.premis by auto
    show ?thesis
    proof (cases "kT = []")
      case True thus ?thesis using k∅δ(1) k∅δ(2) k∅δ(3) kf km'(1) by blast
    next
      case False
      hence 4: "arity kf > 0" using wf_k kf TAtom_term_cases const_type by fastforce
      then obtain kT' where kT': "Γ k = TComp kf kT'" by (simp add: fun_type kf)
      then obtain V where V:
        "Fun kf V ∈ set M" "Γ (Fun kf V) = Γ (Var x)" "∀ u ∈ set V. ∃ v. u = Var v"
        "distinct V" "is_Fun (Fun kf V)"
        using is_TComp_var_instance_closedD[OF _ _ closed(3), of x]
        x m(2) k∅δ(1) 3 wt_subst_trm'' [OF k∅δ(2)]
        by (metis fun_type_id_eq term.disc(2) is_VarE)
      have 5: "kT' = map Γ V"
        using fun_type[OF 4] x kT' k∅δ m(2) V(2)
        by (metis term.inject(2) wt_subst_trm'')
      thus ?thesis
        using TComp_var_instance_wt_subst_exists wf_k kf 4 V(3,4) kT' V(1,5)
        by metis
    qed
  qed
qed
qed
qed

```

qed

lemma SMP\_D':

fixes M

defines " $\delta \equiv \text{var\_rename } (\text{max\_var\_set } (\text{fv\_set } (\text{set } M)))$ "

assumes M\_SMP\_repr: "finite\_SMP\_representation arity Ana  $\Gamma$  M"

and s: "s  $\in$  SMP (set M)" "is\_Fun s" " $\nexists f. s = \text{Fun } f []$ "

and t: "t  $\in$  SMP (set M)" "is\_Fun t" " $\nexists f. t = \text{Fun } f []$ "

obtains  $\sigma$  s0  $\vartheta$  t0

where "wt\_subst  $\sigma$ " "wf\_trms (subst\_range  $\sigma$ )" "s0  $\in$  set M" "is\_Fun s0" "s = s0  $\cdot$   $\sigma$ " " $\Gamma$  s =  $\Gamma$  s0"

and "wt\_subst  $\vartheta$ " "wf\_trms (subst\_range  $\vartheta$ )" "t0  $\in$  set M" "is\_Fun t0" "t = t0  $\cdot$   $\delta \cdot \vartheta$ " " $\Gamma$  t =  $\Gamma$  t0"

proof -

obtain  $\sigma$  s0 where

s0: "wt\_subst  $\sigma$ " "wf\_trms (subst\_range  $\sigma$ )" "s0  $\in$  set M" "s = s0  $\cdot$   $\sigma$ " "is\_Fun s0"

using s(3) SMP\_D[OF s(1,2) M\_SMP\_repr] unfolding  $\delta\_def$  by metis

obtain  $\vartheta$  t0 where t0:

"wt\_subst  $\vartheta$ " "wf\_trms (subst\_range  $\vartheta$ )" "t0  $\in$  set M" "t = t0  $\cdot$   $\delta \cdot \vartheta$ " "is\_Fun t0"

using t(3) SMP\_D[OF t(1,2) M\_SMP\_repr] var\_rename\_wt'[of \_ t]

wf\_trms\_subst\_compose\_Var\_range(1)[OF \_ var\_rename\_is\_renaming(2)]

unfolding  $\delta\_def$  by metis

have " $\Gamma$  s =  $\Gamma$  s0" " $\Gamma$  t =  $\Gamma$  (t0  $\cdot$   $\delta$ )" " $\Gamma$  (t0  $\cdot$   $\delta$ ) =  $\Gamma$  t0"

using s0 t0 wt\_subst\_trm'' by (metis, metis, metis  $\delta\_def$  var\_rename\_wt(1))

thus ?thesis using s0 t0 that by simp

qed

lemma SMP\_D'':

fixes t: "('fun, ('fun, 'atom) term  $\times$  nat) term"

assumes t\_SMP: "t  $\in$  SMP (set M)"

and M\_SMP\_repr: "finite\_SMP\_representation arity Ana  $\Gamma$  M"

shows " $\exists m \in \text{set } M. \exists \delta. \text{wt\_subst } \delta \wedge \text{wf\_trms } (\text{subst\_range } \delta) \wedge t = m \cdot \delta$ "

proof (cases "( $\exists x. t = \text{Var } x$ )  $\vee$  ( $\exists c. t = \text{Fun } c []$ )")

case True

have M\_wf: "wf\_trms (set M)"

and M\_var\_inst\_cl: "is\_TComp\_var\_instance\_closed  $\Gamma$  M"

and M\_subterms\_cl: "has\_all\_wt\_instances\_of  $\Gamma$  (subterms\_set (set M)) (set M)"

and M\_Ana\_cl: "has\_all\_wt\_instances\_of  $\Gamma$  ( $\bigcup ((\text{set} \circ \text{fst} \circ \text{Ana}) ' \text{set } M))$  (set M)) (set M)"

using finite\_SMP\_representationD[OF M\_SMP\_repr] by blast+

have M\_Ana\_wf: "wf\_trms ( $\bigcup ((\text{set} \circ \text{fst} \circ \text{Ana}) ' \text{set } M))$ "

proof

fix k assume "k  $\in \bigcup ((\text{set} \circ \text{fst} \circ \text{Ana}) ' \text{set } M)$ "

then obtain m where m: "m  $\in$  set M" "k  $\in$  set (fst (Ana m))" by force

thus "wf\_trm k" using M\_wf Ana\_keys\_wf'[of m "fst (Ana m)" \_ k] surjective\_pairing by blast

qed

show ?thesis using True

proof

assume " $\exists x. t = \text{Var } x$ "

then obtain x y where x: "t = Var x" and y: "y  $\in \text{fv\_set } (\text{set } M)$ " " $\Gamma$  (Var y) =  $\Gamma$  (Var x)"

using SMP\_D\_aux1[OF t\_SMP M\_subterms\_cl M\_var\_inst\_cl M\_wf] by fastforce

then obtain m  $\delta$  where m: "m  $\in$  set M" "m  $\cdot$   $\delta = \text{Var } y$ " and  $\delta$ : "wt\_subst  $\delta$ "

using has\_all\_wt\_instances\_ofD'[OF wf\_trms\_subterms[OF M\_wf] M\_wf M\_subterms\_cl, of "Var y"]

vars\_iff\_subtermeq\_set[of y "set M"]

by fastforce

obtain z where z: "m = Var z" using m(2) by (cases m) auto

define  $\vartheta$  where " $\vartheta \equiv \text{Var}(z := \text{Var } x)::('fun, ('fun, 'atom) \text{term} \times \text{nat}) \text{subst}$ "

have " $\Gamma$  (Var z) =  $\Gamma$  (Var x)" using y(2) m(2) z wt\_subst\_trm''[OF  $\delta$ , of m] by argo

hence "wt\_subst  $\vartheta$ " "wf\_trms (subst\_range  $\vartheta$ )" unfolding  $\vartheta\_def$  wt\_subst\_def by force+

```

    moreover have "t = m ·  $\vartheta$ " using x z unfolding  $\vartheta\_def$  by simp
    ultimately show ?thesis using m(1) by blast
qed (use SMP_D[OF t_SMP _ M_SMP_repr] in blast)
qed (use SMP_D[OF t_SMP _ M_SMP_repr] in blast)
end

lemma tfrset_if_comp_tfrset:
  assumes "comp_tfrset arity Ana  $\Gamma$  M"
  shows "tfrset (set M)"
proof -
  let ? $\delta$  = "var_rename (max_var_set (fvset (set M)))"
  have M_SMP_repr: "finite_SMP_representation arity Ana  $\Gamma$  M"
    by (metis comp_tfrset_def assms)

  have M_finite: "finite (set M)"
    using assms card_gt_0_iff unfolding comp_tfrset_def by blast

  show ?thesis
proof (unfold tfrset_def; intro ballI impI)
  fix s t assume "s ∈ SMP (set M) - Var'V" "t ∈ SMP (set M) - Var'V"
  hence st: "s ∈ SMP (set M)" "is_Fun s" "t ∈ SMP (set M)" "is_Fun t" by auto
  have "¬(∃ $\delta$ . Unifier  $\delta$  s t)" when st_type_neq: " $\Gamma$  s ≠  $\Gamma$  t"
  proof (cases "∃f. s = Fun f [] ∨ t = Fun f []")
    case False
    then obtain  $\sigma$  s0  $\vartheta$  t0 where
      s0: "s0 ∈ set M" "is_Fun s0" "s = s0 ·  $\sigma$ " " $\Gamma$  s =  $\Gamma$  s0"
      and t0: "t0 ∈ set M" "is_Fun t0" "t = t0 · ? $\delta$  ·  $\vartheta$ " " $\Gamma$  t =  $\Gamma$  t0"
      using SMP_D'[OF M_SMP_repr st(1,2) _ st(3,4)] by metis
    hence "¬(∃ $\delta$ . Unifier  $\delta$  s0 (t0 · ? $\delta$ ))"
      using assms mgu_None_is_subst_neq st_type_neq wt_subst_trm''[OF var_rename_wt(1)]
      unfolding comp_tfrset_def Let_def by metis
    thus ?thesis
      using vars_term_disjoint_imp_unifier[OF var_rename_fv_set_disjoint[OF M_finite]] s0(1) t0(1)
      unfolding s0(3) t0(3) by (metis (no_types, hide_lams) subst_subst_compose)
  qed (use st_type_neq st(2,4) in auto)
  thus " $\Gamma$  s =  $\Gamma$  t" when "∃ $\delta$ . Unifier  $\delta$  s t" by (metis that)
qed
qed

lemma tfrset_if_comp_tfrset':
  assumes "let N = SMP0 Ana  $\Gamma$  M in set M ⊆ set N ∧ comp_tfrset arity Ana  $\Gamma$  N"
  shows "tfrset (set M)"
by (rule tfr_subset(2)[
  OF tfrset_if_comp_tfrset[OF conjunct2[OF assms[unfolded Let_def]]]
  conjunct1[OF assms[unfolded Let_def]]])

lemma tfrstp_is_comp_tfrstp: "tfrstp a = comp_tfrstp  $\Gamma$  a"
proof (cases a)
  case (Equality ac t t')
  thus ?thesis
    using mgu_always_unifies[of t _ t'] mgu_gives_MGU[of t t']
    by auto
next
  case (Inequality X F)
  thus ?thesis
    using tfrstp.simps(2)[of X F]
    comp_tfrstp.simps(2)[of  $\Gamma$  X F]
    Fun_range_case(2)[of "subtermsset (trmspairs F)"]
    unfolding is_Var_def
    by auto
qed auto

lemma tfrst_if_comp_tfrst:

```



```

assumes "comp_tfrst arity Ana  $\Gamma$  M S"
shows "tfrst S"
unfolding tfrst_def
proof
  have comp_tfrset_M: "comp_tfrset arity Ana  $\Gamma$  M"
    using assms unfolding comp_tfrst_def by blast

  have wftrms_M: "wftrms (set M)"
    and wftrms_S: "wftrms (trmsst S)"
    and S_trms_instance_M: "has_all_wt_instances_of  $\Gamma$  (trmsst S) (set M)"
    using assms wftrm_code trms_listst_is_trmsst
    unfolding comp_tfrst_def comp_tfrset_def finite_SMP_representation_def list_all_iff
    by blast+

  show "tfrset (trmsst S)"
    using tfr_subset(3)[OF tfrset_if_comp_tfrset[OF comp_tfrset_M] SMP_SMP_subset]
      SMP_I'[OF wftrms_S wftrms_M S_trms_instance_M]
    by blast

  have "list_all (comp_tfrstp  $\Gamma$ ) S" by (metis assms comp_tfrst_def)
  thus "list_all tfrstp S" by (induct S) (simp_all add: tfrstp_is_comp_tfrstp)
qed

lemma tfrst_if_comp_tfrst':
  assumes "comp_tfrst arity Ana  $\Gamma$  (SMP0 Ana  $\Gamma$  (trms_listst S)) S"
  shows "tfrst S"
by (rule tfrst_if_comp_tfrst[OF assms])

```

### Lemmata for Checking Ground SMP (GSMP) Disjointness

```

context
begin
private lemma ground_SMP_disjointI_aux1:
  fixes M:: "('fun, ('fun, 'atom) term  $\times$  nat) term set"
  assumes f_def: "f  $\equiv$   $\lambda$ M. {t  $\cdot$   $\delta$  | t  $\delta$ . t  $\in$  M  $\wedge$  wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  fv (t  $\cdot$   $\delta$ ) = {}}"
  and g_def: "g  $\equiv$   $\lambda$ M. {t  $\in$  M. fv t = {}}"
  shows "f (SMP M) = g (SMP M)"
proof
  have "t  $\in$  f (SMP M)" when t: "t  $\in$  SMP M" "fv t = {}" for t
  proof -
    define  $\delta$  where " $\delta \equiv$  Var::('fun, ('fun, 'atom) term  $\times$  nat) subst"
    have "wtsubst  $\delta$ " "wftrms (subst_range  $\delta$ )" "t = t  $\cdot$   $\delta$ "
      using subst_apply_term_empty[of t] that(2) wt_subst_Var wf_trm_subst_range_Var
      unfolding  $\delta$ _def by auto
    thus ?thesis using SMP.Substitution[OF t(1), of  $\delta$ ] t(2) unfolding f_def by fastforce
  qed
  thus "g (SMP M)  $\subseteq$  f (SMP M)" unfolding g_def by blast
qed (use f_def g_def in blast)

private lemma ground_SMP_disjointI_aux2:
  fixes M:: "('fun, ('fun, 'atom) term  $\times$  nat) term list"
  assumes f_def: "f  $\equiv$   $\lambda$ M. {t  $\cdot$   $\delta$  | t  $\delta$ . t  $\in$  M  $\wedge$  wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  fv (t  $\cdot$   $\delta$ ) = {}}"
  and M_SMP_repr: "finite_SMP_representation arity Ana  $\Gamma$  M"
  shows "f (set M) = f (SMP (set M))"
proof
  have M_wf: "wftrms (set M)"
    and M_var_inst_cl: "is_TComp_var_instance_closed  $\Gamma$  M"
    and M_subterms_cl: "has_all_wt_instances_of  $\Gamma$  (subtermsset (set M)) (set M)"
    and M_Ana_cl: "has_all_wt_instances_of  $\Gamma$  ( $\bigcup$  ((set  $\circ$  fst  $\circ$  Ana) ' set M)) (set M)"
    using finite_SMP_representationD[OF M_SMP_repr] by blast+

```

```

show "f (SMP (set M))  $\subseteq$  f (set M)"
proof
  fix t assume "t  $\in$  f (SMP (set M))"
  then obtain s  $\delta$  where s: "t = s  $\cdot$   $\delta$ " "s  $\in$  SMP (set M)" "fv (s  $\cdot$   $\delta$ ) = {}"
    and  $\delta$ : "wtsubst  $\delta$ " "wftrms (subst_range  $\delta$ )"
    unfolding f_def by blast

  have t_wf: "wftrm t" using SMP_wf_trm[OF s(2) M_wf] s(1) wf_trm_subst[OF  $\delta$ (2)] by blast

  obtain m  $\vartheta$  where m: "m  $\in$  set M" "s = m  $\cdot$   $\vartheta$ " and  $\vartheta$ : "wtsubst  $\vartheta$ " "wftrms (subst_range  $\vartheta$ )"
    using SMP_D''[OF s(2) M_SMP_repr] by blast

  have "t = m  $\cdot$  ( $\vartheta$   $\circ_s$   $\delta$ )" "fv (m  $\cdot$  ( $\vartheta$   $\circ_s$   $\delta$ )) = {}" using s(1,3) m(2) by simp_all
  thus "t  $\in$  f (set M)"
    using m(1) wt_subst_compose[OF  $\vartheta$ (1)  $\delta$ (1)] wf_trms_subst_compose[OF  $\vartheta$ (2)  $\delta$ (2)]
    unfolding f_def by blast
qed
qed (auto simp add: f_def)

private lemma ground_SMP_disjointI_aux3:
  fixes A B C: "('fun, ('fun, 'atom) term  $\times$  nat) term set"
  defines "P  $\equiv$   $\lambda$ t s.  $\exists$  $\delta$ . wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  Unifier  $\delta$  t s"
  assumes f_def: "f  $\equiv$   $\lambda$ M. {t  $\cdot$   $\delta$  | t  $\delta$ . t  $\in$  M  $\wedge$  wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  fv (t  $\cdot$   $\delta$ ) = {}}"
  and Q_def: "Q  $\equiv$   $\lambda$ t. intruder_synth' public arity {} t"
  and R_def: "R  $\equiv$   $\lambda$ t.  $\exists$ u  $\in$  C. is_wt_instance_of_cond  $\Gamma$  t u"
  and AB: "wftrms A" "wftrms B" "fvset A  $\cap$  fvset B = {}"
  and C: "wftrms C"
  and ABC: " $\forall$ t  $\in$  A.  $\forall$ s  $\in$  B. P t s  $\longrightarrow$  (Q t  $\wedge$  Q s)  $\vee$  (R t  $\wedge$  R s)"
  shows "f A  $\cap$  f B  $\subseteq$  f C  $\cup$  {m. {}  $\vdash_c$  m}"
proof
  fix t assume "t  $\in$  f A  $\cap$  f B"
  then obtain ta tb  $\delta_a$   $\delta_b$  where
    ta: "t = ta  $\cdot$   $\delta_a$ " "ta  $\in$  A" "wtsubst  $\delta_a$ " "wftrms (subst_range  $\delta_a$ )" "fv (ta  $\cdot$   $\delta_a$ ) = {}"
    and tb: "t = tb  $\cdot$   $\delta_b$ " "tb  $\in$  B" "wtsubst  $\delta_b$ " "wftrms (subst_range  $\delta_b$ )" "fv (tb  $\cdot$   $\delta_b$ ) = {}"
    unfolding f_def by blast

  have ta_tb_wf: "wftrm ta" "wftrm tb" "fv ta  $\cap$  fv tb = {}" " $\Gamma$  ta =  $\Gamma$  tb"
    using ta(1,2) tb(1,2) AB fv_subset_subterms
    wt_subst_trm''[OF ta(3), of ta] wt_subst_trm''[OF tb(3), of tb]
    by (fast, fast, blast, simp)

  obtain  $\vartheta$  where  $\vartheta$ : "Unifier  $\vartheta$  ta tb" "wtsubst  $\vartheta$ " "wftrms (subst_range  $\vartheta$ )"
    using vars_term_disjoint_imp_unifier[OF ta_tb_wf(3), of  $\delta_a$   $\delta_b$ ]
    ta(1) tb(1) wt_Unifier_if_Unifier[OF ta_tb_wf(1,2,4)]
    by blast
  hence "(Q ta  $\wedge$  Q tb)  $\vee$  (R ta  $\wedge$  R tb)" using ABC ta(2) tb(2) unfolding P_def by blast+
  thus "t  $\in$  f C  $\cup$  {m. {}  $\vdash_c$  m}"
proof
  show "Q ta  $\wedge$  Q tb  $\implies$  ?thesis"
    using ta(1) pgwt_ground[of ta] pgwt_is_empty_synth[of ta] subst_ground_ident[of ta  $\delta_a$ ]
    unfolding Q_def f_def intruder_synth_code[symmetric] by simp
next
  assume "R ta  $\wedge$  R tb"
  then obtain ua  $\sigma_a$  where ua: "ta = ua  $\cdot$   $\sigma_a$ " "ua  $\in$  C" "wtsubst  $\sigma_a$ " "wftrms (subst_range  $\sigma_a$ )"
    using  $\vartheta$  ABC ta_tb_wf(1,2) ta(2) tb(2) C is_wt_instance_of_condD'
    unfolding P_def R_def by metis

  have "t = ua  $\cdot$  ( $\sigma_a$   $\circ_s$   $\delta_a$ )" "fv t = {}"
    using ua(1) ta(1,5) tb(1,5) by auto
  thus ?thesis
    using ua(2) wt_subst_compose[OF ua(3) ta(3)] wf_trms_subst_compose[OF ua(4) ta(4)]
    unfolding f_def by blast

```

```

qed
qed

lemma ground_SMP_disjointI:
  fixes A B:: "('fun, ('fun, 'atom) term × nat) term list" and C
  defines "f ≡ λM. {t · δ | t δ. t ∈ M ∧ wt_subst δ ∧ wf_trms (subst_range δ) ∧ fv (t · δ) = {}}"
    and "g ≡ λM. {t ∈ M. fv t = {}}"
    and "Q ≡ λt. intruder_synth' public arity {} t"
    and "R ≡ λt. ∃ u ∈ C. is_wt_instance_of_cond Γ t u"
  assumes AB_fv_disj: "fv_set (set A) ∩ fv_set (set B) = {"
    and A_SMP_repr: "finite_SMP_representation arity Ana Γ A"
    and B_SMP_repr: "finite_SMP_representation arity Ana Γ B"
    and C_wf: "wf_trms C"
    and ABC: "∀ t ∈ set A. ∀ s ∈ set B. Γ t = Γ s ∧ mgu t s ≠ None ⟶ (Q t ∧ Q s) ∨ (R t ∧ R s)"
  shows "g (SMP (set A)) ∩ g (SMP (set B)) ⊆ f C ∪ {m. {} ⊢_c m}"
proof -
  have AB_wf: "wf_trms (set A)" "wf_trms (set B)"
    using A_SMP_repr B_SMP_repr
    unfolding finite_SMP_representation_def wf_trms_code list_all_iff
    by blast+

  let ?P = "λt s. ∃ δ. wt_subst δ ∧ wf_trms (subst_range δ) ∧ Unifier δ t s"
  have ABC': "∀ t ∈ set A. ∀ s ∈ set B. ?P t s ⟶ (Q t ∧ Q s) ∨ (R t ∧ R s)"
    by (metis (no_types) ABC mgu_None_is_subst_neq wt_subst_trm'')

  show ?thesis
    using ground_SMP_disjointI_aux1[OF f_def g_def, of "set A"]
      ground_SMP_disjointI_aux1[OF f_def g_def, of "set B"]
      ground_SMP_disjointI_aux2[OF f_def A_SMP_repr]
      ground_SMP_disjointI_aux2[OF f_def B_SMP_repr]
      ground_SMP_disjointI_aux3[OF f_def Q_def R_def AB_wf AB_fv_disj C_wf ABC']
    by argo
qed

end

end

end

```

## 3.4 The Typing Result (Typing\_Result)

```

theory Typing_Result
imports Typed_Model
begin

```

### 3.4.1 The Typing Result for the Composition-Only Intruder

```

context typed_model
begin

```

#### Well-typedness and Type-Flaw Resistance Preservation

```

context
begin

```

```

private lemma LI_preserves_tfr_stp_all_single:
  assumes "(S, ∅) ~ (S', ∅)" "wf_constr S ∅" "wt_subst ∅"
  and "list_all tfr_stp S" "tfr_set (trms_st S)" "wf_trms (trms_st S)"
  shows "list_all tfr_stp S'"
using assms
proof (induction rule: LI_rel.induct)

```

```

case (Compose S X f S'  $\emptyset$ )
hence "list_all tfrstp S" "list_all tfrstp S'" by simp_all
moreover have "list_all tfrstp (map Send X)" by (induct X) auto
ultimately show ?case by simp
next
case (Unify S f Y  $\delta$  X S'  $\emptyset$ )
hence "list_all tfrstp (S@S')" by simp

have "fvst (S@Send (Fun f X)#S')  $\cap$  bvarsst (S@S') = {}"
  using Unify.premis(1) by (auto simp add: wfconstr-def)
moreover have "fv (Fun f X)  $\subseteq$  fvst (S@Send (Fun f X)#S')" by auto
moreover have "fv (Fun f Y)  $\subseteq$  fvst (S@Send (Fun f X)#S')"
  using Unify.hyps(2) fv_subset_if_in_strand_ik'[of "Fun f Y" S] by force
ultimately have bvars_disj:
  "bvarsst (S@S')  $\cap$  fv (Fun f X) = {}" "bvarsst (S@S')  $\cap$  fv (Fun f Y) = {}"
  by blast+

have "wftrm (Fun f X)" using Unify.premis(5) by simp
moreover have "wftrm (Fun f Y)"
proof -
  obtain x where "x  $\in$  set S" "Fun f Y  $\in$  subtermsset (trmsstp x)" "wftrms (trmsstp x)"
  using Unify.hyps(2) Unify.premis(5) by force+
  thus ?thesis using wftrm-subterm by auto
qed
moreover have
  "Fun f X  $\in$  SMP (trmsst (S@Send (Fun f X)#S'))" "Fun f Y  $\in$  SMP (trmsst (S@Send (Fun f X)#S'))"
  using SMP_append[of S "Send (Fun f X)#S'"] SMP_Cons[of "Send (Fun f X)" S']
  SMP_ikI[OF Unify.hyps(2)]
  by auto
hence " $\Gamma$  (Fun f X) =  $\Gamma$  (Fun f Y)"
  using Unify.premis(4) mgu_gives_MGU[OF Unify.hyps(3)[symmetric]]
  unfolding tfrset-def by blast
ultimately have "wtsubst  $\delta$ " using mgu_wt_if_same_type[OF Unify.hyps(3)[symmetric]] by metis
moreover have "wftrms (subst_range  $\delta$ )"
  using mgu_wf_trm[OF Unify.hyps(3)[symmetric]]  $\langle$ wftrm (Fun f X) $\rangle$   $\langle$ wftrm (Fun f Y) $\rangle$ 
  by (metis wftrm-subst_range_iff)
moreover have "bvarsst (S@S')  $\cap$  range_vars  $\delta$  = {}"
  using mgu_vars_bounded[OF Unify.hyps(3)[symmetric]] bvars_disj by fast
ultimately show ?case using tfrstp-all_wt_subst_apply[OF (list_all tfrstp (S@S'))] by metis
next
case (Equality S  $\delta$  t t' a S'  $\emptyset$ )
have "list_all tfrstp (S@S'" " $\Gamma$  t =  $\Gamma$  t'"
  using tfrstp-all_same_type[of S a t t' S']
  tfrstp-all_split(5)[of S _ S']
  MGU_is_Unifier[OF mgu_gives_MGU[OF Equality.hyps(2)[symmetric]]]
  Equality.premis(3)
  by blast+
moreover have "wftrm t" "wftrm t'" using Equality.premis(5) by auto
ultimately have "wtsubst  $\delta$ "
  using mgu_wt_if_same_type[OF Equality.hyps(2)[symmetric]]
  by metis
moreover have "wftrms (subst_range  $\delta$ )"
  using mgu_wf_trm[OF Equality.hyps(2)[symmetric]]  $\langle$ wftrm t $\rangle$   $\langle$ wftrm t' $\rangle$ 
  by (metis wftrm-subst_range_iff)
moreover have "fvst (S@Equality a t t'#S')  $\cap$  bvarsst (S@Equality a t t'#S') = {}"
  using Equality.premis(1) by (auto simp add: wfconstr-def)
hence "bvarsst (S@S')  $\cap$  fv t = {}" "bvarsst (S@S')  $\cap$  fv t' = {}" by auto
hence "bvarsst (S@S')  $\cap$  range_vars  $\delta$  = {}"
  using mgu_vars_bounded[OF Equality.hyps(2)[symmetric]] by fast
ultimately show ?case using tfrstp-all_wt_subst_apply[OF (list_all tfrstp (S@S'))] by metis
qed

private lemma LI_in_SMP_subset_single:

```

```

assumes "(S,  $\vartheta$ )  $\rightsquigarrow$  (S',  $\vartheta'$ )" "wf_constr S  $\vartheta$ " "wt_subst  $\vartheta$ "
      "tfr_set (trms_st S)" "wf_trms (trms_st S)" "list_all tfr_stp S"
and "trms_st S  $\subseteq$  SMP M"
shows "trms_st S'  $\subseteq$  SMP M"
using assms
proof (induction rule: LI_rel.induct)
case (Compose S X f S'  $\vartheta$ )
hence "SMP (trms_st [Send (Fun f X)])  $\subseteq$  SMP M"
proof -
  have "SMP (trms_st [Send (Fun f X)])  $\subseteq$  SMP (trms_st (S@Send (Fun f X)#S'))"
  using trms_st_append SMP_mono by auto
  thus ?thesis
  using SMP_union[of "trms_st (S@Send (Fun f X)#S')]" M]
    SMP_subset_union_eq[OF Compose.prem6]
  by auto
qed
thus ?case using Compose.prem6 by auto
next
case (Unify S f Y  $\delta$  X S'  $\vartheta$ )
have "Fun f X  $\in$  SMP (trms_st (S@Send (Fun f X)#S'))" by auto
moreover have "MGU  $\delta$  (Fun f X) (Fun f Y)"
  by (metis mgu_gives_MGU[OF Unify.hyps(3)[symmetric]])
moreover have
  " $\bigwedge x. x \in \text{set } S \implies \text{wf}_{trms} (\text{trms}_{stp} x)$ " "wf_trm (Fun f X)"
  using Unify.prem4 by force+
moreover have "Fun f Y  $\in$  SMP (trms_st (S@Send (Fun f X)#S'))"
  by (meson SMP_ikI Unify.hyps(2) contra_subsetD ik_append_subset(1))
ultimately have "wf_trm (Fun f Y)" " $\Gamma$  (Fun f X) =  $\Gamma$  (Fun f Y)"
  using ik_st_subterm_exD[OF <Fun f Y  $\in$  ik_st S>] <tfr_set (trms_st (S@Send (Fun f X)#S'))>
  unfolding tfr_set_def by (metis (full_types) SMP_wf_trm Unify.prem4, blast)
hence "wt_subst  $\delta$ " by (metis mgu_wt_if_same_type[OF Unify.hyps(3)[symmetric] <wf_trm (Fun f X)>])
moreover have "wf_trms (subst_range  $\delta$ )"
  using mgu_wf_trm[OF Unify.hyps(3)[symmetric] <wf_trm (Fun f X)> <wf_trm (Fun f Y)>] by simp
ultimately have "trms_st ((S@Send (Fun f X)#S')  $\cdot_{st}$   $\delta$ )  $\subseteq$  SMP M"
  using SMP.Substitution Unify.prem6 wt_subst_SMP_subset by metis
thus ?case by auto
next
case (Equality S  $\delta$  t t' a S'  $\vartheta$ )
hence " $\Gamma$  t =  $\Gamma$  t'"
  using tfr_stp_all_same_type MGU_is_Unifier[OF mgu_gives_MGU[OF Equality.hyps(2)[symmetric]]]
  by metis
moreover have "t  $\in$  SMP (trms_st (S@Equality a t t'#S'))" "t'  $\in$  SMP (trms_st (S@Equality a t t'#S'))"
  using Equality.prem1 by auto
moreover have "MGU  $\delta$  t t'" using mgu_gives_MGU[OF Equality.hyps(2)[symmetric]] by metis
moreover have " $\bigwedge x. x \in \text{set } S \implies \text{wf}_{trms} (\text{trms}_{stp} x)$ " "wf_trm t" "wf_trm t'"
  using Equality.prem4 by force+
ultimately have "wt_subst  $\delta$ " by (metis mgu_wt_if_same_type[OF Equality.hyps(2)[symmetric] <wf_trm t>])
moreover have "wf_trms (subst_range  $\delta$ )"
  using mgu_wf_trm[OF Equality.hyps(2)[symmetric] <wf_trm t> <wf_trm t'>] by simp
ultimately have "trms_st ((S@Equality a t t'#S')  $\cdot_{st}$   $\delta$ )  $\subseteq$  SMP M"
  using SMP.Substitution Equality.prem6 wt_subst_SMP_subset by metis
thus ?case by auto
qed

private lemma LI_preserves_tfr_single:
  assumes "(S,  $\vartheta$ )  $\rightsquigarrow$  (S',  $\vartheta'$ )" "wf_constr S  $\vartheta$ " "wt_subst  $\vartheta$ " "wf_trms (subst_range  $\vartheta$ )"
      "tfr_set (trms_st S)" "wf_trms (trms_st S)"
      "list_all tfr_stp S"
  shows "tfr_set (trms_st S')  $\wedge$  wf_trms (trms_st S')"
using assms
proof (induction rule: LI_rel.induct)
case (Compose S X f S'  $\vartheta$ )
let ?SMPmap = "SMP (trms_st (S@map Send X@S')) - (Var'V)"

```

```

have "?SMPmap  $\subseteq$  SMP (trmsst (S@Send (Fun f X)#S')) - (Var'V)"
  using SMP_fun_map_snd_subset[of X f]
    SMP_append[of "map Send X" S'] SMP_Cons[of "Send (Fun f X)" S']
    SMP_append[of S "Send (Fun f X)#S'"] SMP_append[of S "map Send X@S'"]
  by auto
hence " $\forall s \in ?SMPmap. \forall t \in ?SMPmap. (\exists \delta. \text{Unifier } \delta \ s \ t) \longrightarrow \Gamma \ s = \Gamma \ t$ "
  using Compose unfolding tfrset_def by (meson subsetCE)
thus ?case
  using LI_preserves_trm_wf[OF r_into_rtrancl[OF LI_rel.Compose[OF Compose.hyps]], of S']
    Compose.prem5(5)
  unfolding tfrset_def by blast
next
case (Unify S f Y  $\delta$  X S'  $\vartheta$ )
let ?SMP $\delta$  = "SMP (trmsst (S@S' .st  $\delta$ )) - (Var'V)"

have "SMP (trmsst (S@S' .st  $\delta$ ))  $\subseteq$  SMP (trmsst (S@Send (Fun f X)#S'))"
proof
  fix s assume "s  $\in$  SMP (trmsst (S@S' .st  $\delta$ ))" thus "s  $\in$  SMP (trmsst (S@Send (Fun f X)#S'))"
    using LI_in_SMP_subset_single[
      OF LI_rel.Unify[OF Unify.hyps] Unify.prem5(1,2,4,5,6)
      MP_subset_SMP(2)[of "S@Send (Fun f X)#S'"]
    ] by (metis SMP_union SMP_subset_union_eq Un_iff)
qed
hence " $\forall s \in ?SMP\delta. \forall t \in ?SMP\delta. (\exists \delta. \text{Unifier } \delta \ s \ t) \longrightarrow \Gamma \ s = \Gamma \ t$ "
  using Unify.prem5(4) unfolding tfrset_def by (meson Diff_iff subsetCE)
thus ?case
  using LI_preserves_trm_wf[OF r_into_rtrancl[OF LI_rel.Unify[OF Unify.hyps]], of S']
    Unify.prem5(5)
  unfolding tfrset_def by blast
next
case (Equality S  $\delta$  t t' a S'  $\vartheta$ )
let ?SMP $\delta$  = "SMP (trmsst (S@S' .st  $\delta$ )) - (Var'V)"

have "SMP (trmsst (S@S' .st  $\delta$ ))  $\subseteq$  SMP (trmsst (S@Equality a t t'#S'))"
proof
  fix s assume "s  $\in$  SMP (trmsst (S@S' .st  $\delta$ ))" thus "s  $\in$  SMP (trmsst (S@Equality a t t'#S'))"
    using LI_in_SMP_subset_single[
      OF LI_rel.Equality[OF Equality.hyps] Equality.prem5(1,2,4,5,6)
      MP_subset_SMP(2)[of "S@Equality a t t'#S'"]
    ] by (metis SMP_union SMP_subset_union_eq Un_iff)
qed
hence " $\forall s \in ?SMP\delta. \forall t \in ?SMP\delta. (\exists \delta. \text{Unifier } \delta \ s \ t) \longrightarrow \Gamma \ s = \Gamma \ t$ "
  using Equality.prem5 unfolding tfrset_def by (meson Diff_iff subsetCE)
thus ?case
  using LI_preserves_trm_wf[OF r_into_rtrancl[OF LI_rel.Equality[OF Equality.hyps]], of _ S']
    Equality.prem5
  unfolding tfrset_def by blast
qed

private lemma LI_preserves_welltypedness_single:
  assumes "(S,  $\vartheta$ )  $\rightsquigarrow$  (S',  $\vartheta'$ )" "wfconstr S  $\vartheta$ " "wtsubst  $\vartheta$ " "wftrms (subst_range  $\vartheta$ )"
  and "tfrset (trmsst S)" "wftrms (trmsst S)" "list_all tfrstp S"
  shows "wtsubst  $\vartheta'$   $\wedge$  wftrms (subst_range  $\vartheta'$ )"
using assms
proof (induction rule: LI_rel.induct)
  case (Unify S f Y  $\delta$  X S'  $\vartheta$ )
  have "wftrm (Fun f X)" using Unify.prem5(5) unfolding tfrset_def by simp
  moreover have "wftrm (Fun f Y)"
  proof -
    obtain x where "x  $\in$  set S" "Fun f Y  $\in$  subtermsset (trmsstp x)" "wftrms (trmsstp x)"
    using Unify.hyps(2) Unify.prem5(5) unfolding tfrset_def by force
    thus ?thesis using wf_trm_subterm by auto
  qed
  qed

```

```

moreover have
  "Fun f X ∈ SMP (trmsst (S@Send (Fun f X)#S'))" "Fun f Y ∈ SMP (trmsst (S@Send (Fun f X)#S'))"
  using SMP_append[of S "Send (Fun f X)#S'"] SMP_Cons[of "Send (Fun f X)" S']
  SMP_ikI[OF Unify.hyps(2)]
  by auto
hence "Γ (Fun f X) = Γ (Fun f Y)"
  using Unify.prem(4) mgu_gives_MGU[OF Unify.hyps(3)[symmetric]]
  unfolding tfrset_def by blast
ultimately have "wtsubst δ" using mgu_wt_if_same_type[OF Unify.hyps(3)[symmetric]] by metis

```

```

have "wftrms (subst_range δ)"
  by (meson mgu_wf_trm[OF Unify.hyps(3)[symmetric] <wftrm (Fun f X)> <wftrm (Fun f Y)>]
  wf_trm_subst_range_iff)

```

```

hence "wftrms (subst_range (ϑ ∘s δ))"
  using wf_trm_subst_range_iff wf_trm_subst <wftrms (subst_range ϑ)>
  unfolding subst_compose_def
  by (metis (no_types, lifting))

```

```

thus ?case by (metis wt_subst_compose[OF <wtsubst ϑ> <wtsubst δ>])

```

next

```

case (Equality S δ t t' a S' ϑ)
have "wftrm t" "wftrm t'" using Equality.prem(5) by simp_all
moreover have "Γ t = Γ t'"
  using <list_all tfrstp (S@Equality a t t'#S')>
  MGU_is_Unifier[OF mgu_gives_MGU[OF Equality.hyps(2)[symmetric]]]
  by auto
ultimately have "wtsubst δ" using mgu_wt_if_same_type[OF Equality.hyps(2)[symmetric]] by metis

```

```

have "wftrms (subst_range δ)"
  by (meson mgu_wf_trm[OF Equality.hyps(2)[symmetric] <wftrm t> <wftrm t'>] wf_trm_subst_range_iff)
hence "wftrms (subst_range (ϑ ∘s δ))"
  using wf_trm_subst_range_iff wf_trm_subst <wftrms (subst_range ϑ)>
  unfolding subst_compose_def
  by (metis (no_types, lifting))
thus ?case by (metis wt_subst_compose[OF <wtsubst ϑ> <wtsubst δ>])

```

qed metis

lemma LI\_preserves\_welltypedness:

```

assumes "(S, ϑ) ~* (S', ϑ')" "wfconstr S ϑ" "wtsubst ϑ" "wftrms (subst_range ϑ)"
  and "tfrset (trmsst S)" "wftrms (trmsst S)" "list_all tfrstp S"
shows "wtsubst ϑ'" (is "?A ϑ'")
  and "wftrms (subst_range ϑ')" (is "?B ϑ'")

```

proof -

```

have "?A ϑ' ∧ ?B ϑ'" using assms
proof (induction S ϑ rule: converse_rtrancl_induct2)
  case (step S1 ϑ1 S2 ϑ2)
  hence "?A ϑ2 ∧ ?B ϑ2" using LI_preserves_welltypedness_single by presburger
  moreover have "wfconstr S2 ϑ2"
    by (fact LI_preserves_wellformedness[OF r_into_rtrancl[OF step.hyps(1)] step.prem(1)])
  moreover have "tfrset (trmsst S2)" "wftrms (trmsst S2)"
    using LI_preserves_tfr_single[OF step.hyps(1)] step.prem by presburger+
  moreover have "list_all tfrstp S2"
    using LI_preserves_tfr_stp_all_single[OF step.hyps(1)] step.prem by fastforce
  ultimately show ?case using step.IH by presburger

```

qed simp

```

thus "?A ϑ'" "?B ϑ'" by simp_all

```

qed

lemma LI\_preserves\_tfr:

```

assumes "(S, ϑ) ~* (S', ϑ')" "wfconstr S ϑ" "wtsubst ϑ" "wftrms (subst_range ϑ)"
  and "tfrset (trmsst S)" "wftrms (trmsst S)" "list_all tfrstp S"
shows "tfrset (trmsst S')" (is "?A S'")
  and "wftrms (trmsst S')" (is "?B S'")
  and "list_all tfrstp S'" (is "?C S'")

```

```

proof -
  have "?A S'  $\wedge$  ?B S'  $\wedge$  ?C S'" using assms
  proof (induction S  $\varnothing$  rule: converse_rtrancl_induct2)
    case (step S1  $\varnothing$ 1 S2  $\varnothing$ 2)
    have "wfconstr S2  $\varnothing$ 2" "tfrset (trmsst S2)" "wftrms (trmsst S2)" "list_all tfrstp S2"
      using LI_preserves_wellformedness[OF r_into_rtrancl[OF step.hyps(1)] step.prem(1)]
      LI_preserves_tfr_single[OF step.hyps(1) step.prem(1,2)]
      LI_preserves_tfr_stp_all_single[OF step.hyps(1) step.prem(1,2)]
      step.prem(3,4,5,6)
    by metis+
    moreover have "wtsubst  $\varnothing$ 2" "wftrms (subst_range  $\varnothing$ 2)"
      using LI_preserves_welltypedness[OF r_into_rtrancl[OF step.hyps(1)] step.prem]
      by simp_all
    ultimately show ?case using step.IH by presburger
  qed blast
  thus "?A S'" "?B S'" "?C S'" by simp_all
qed
end

```

### Simple Constraints are Well-typed Satisfiable

Proving the existence of a well-typed interpretation

```

context
begin
lemma wt_interpretation_exists:
  obtains  $\mathcal{I}::('fun, 'var) \text{subst}$ 
  where "interpretationsubst  $\mathcal{I}$ " "wtsubst  $\mathcal{I}$ " "subst_range  $\mathcal{I} \subseteq \text{public\_ground\_wf\_terms}$ "
proof
  define  $\mathcal{I}$  where " $\mathcal{I} = (\lambda x. (\text{SOME } t. \Gamma (\text{Var } x) = \Gamma t \wedge \text{public\_ground\_wf\_term } t))$ "
  { fix x t assume " $\mathcal{I} x = t$ "
    hence " $\Gamma (\text{Var } x) = \Gamma t \wedge \text{public\_ground\_wf\_term } t$ "
      using someI_ex[of " $\lambda t. \Gamma (\text{Var } x) = \Gamma t \wedge \text{public\_ground\_wf\_term } t$ ",
        OF type_pgwt_inhabited[of "Var x"]]
    unfolding  $\mathcal{I}$ _def wftrm_def by simp
  } hence props: " $\mathcal{I} v = t \implies \Gamma (\text{Var } v) = \Gamma t \wedge \text{public\_ground\_wf\_term } t$ " for v t by metis

  have " $\mathcal{I} v \neq \text{Var } v$ " for v using props pgwt_ground by force
  hence "subst_domain  $\mathcal{I} = \text{UNIV}$ " by auto
  moreover have "ground (subst_range  $\mathcal{I}$ )" by (simp add: props pgwt_ground)
  ultimately show "interpretationsubst  $\mathcal{I}$ " by metis
  show "wtsubst  $\mathcal{I}$ " unfolding wtsubst_def using props by simp
  show "subst_range  $\mathcal{I} \subseteq \text{public\_ground\_wf\_terms}$ " by (auto simp add: props)
qed

```

```

lemma wt_grounding_subst_exists:
  " $\exists \varnothing. \text{wt}_{\text{subst}} \varnothing \wedge \text{wf}_{\text{trms}} (\text{subst\_range } \varnothing) \wedge \text{fv } (t \cdot \varnothing) = \{\}$ "
proof -
  obtain  $\varnothing$  where  $\varnothing$ : "interpretationsubst  $\varnothing$ " "wtsubst  $\varnothing$ " "subst_range  $\varnothing \subseteq \text{public\_ground\_wf\_terms}$ "
    using wt_interpretation_exists by blast
  show ?thesis using pgwt_wellformed interpretation_grounds[OF  $\varnothing$ (1)]  $\varnothing$ (2,3) by blast
qed

```

```

private fun fresh_pgwt::"'fun set  $\Rightarrow$  ('fun, 'atom) term_type  $\Rightarrow$  ('fun, 'var) term" where
  "fresh_pgwt S (TAtom a) =
    Fun (SOME c. c  $\notin$  S  $\wedge$   $\Gamma$  (Fun c []) = TAtom a  $\wedge$  public c) []"
  | "fresh_pgwt S (TComp f T) = Fun f (map (fresh_pgwt S) T)"

```

```

private lemma fresh_pgwt_same_type:
  assumes "finite S" "wftrm t"
  shows " $\Gamma$  (fresh_pgwt S ( $\Gamma$  t)) =  $\Gamma$  t"

```

```

proof -
  let ?P = " $\lambda \tau::('fun, 'atom) \text{term\_type}. \text{wf}_{\text{trm}} \tau \wedge (\forall f T. \text{TComp } f T \sqsubseteq \tau \longrightarrow 0 < \text{arity } f)$ "

```



```

{ fix  $\tau$  assume "?P  $\tau$ " hence " $\Gamma$  (fresh_pgwt S  $\tau$ ) =  $\tau$ "
proof (induction  $\tau$ )
  case (Var a)
  let ?P = " $\lambda c. c \notin S \wedge \Gamma$  (Fun c []) = Var a  $\wedge$  public c"
  let ?Q = " $\lambda c. \Gamma$  (Fun c []) = Var a  $\wedge$  public c"
  have " {c. ?Q c} - S = {c. ?P c}" by auto
  hence "infinite {c. ?P c}"
    using Diff_infinite_finite[OF assms(1) infinite_typed_consts[of a]]
    by metis
  hence " $\exists c. ?P c$ " using not_finite_existsD by blast
  thus ?case using someI_ex[of ?P] by auto
next
  case (Fun f T)
  have f: "0 < arity f" using Fun.premis fun_type_inv by auto
  have " $\bigwedge t. t \in \text{set } T \implies ?P t$ "
    using Fun.premis wf_trm_subtermeq term.le_less_trans Fun_param_is_subterm
    by metis
  hence " $\bigwedge t. t \in \text{set } T \implies \Gamma$  (fresh_pgwt S t) = t" using Fun.premis Fun.IH by auto
  hence "map  $\Gamma$  (map (fresh_pgwt S) T) = T" by (induct T) auto
  thus ?case using fun_type[OF f] by simp
qed
} thus ?thesis using assms(1)  $\Gamma_{wf}'$ [OF assms(2)]  $\Gamma_{wf}$ (1) by auto
qed

private lemma fresh_pgwt_empty_synth:
  assumes "finite S" "wf_trm t"
  shows "{ }  $\vdash_c$  fresh_pgwt S ( $\Gamma$  t)"
proof -
  let ?P = " $\lambda \tau :: ('fun, 'atom) \text{term\_type}. wf_{trm} \tau \wedge (\forall f T. TComp f T \sqsubseteq \tau \longrightarrow 0 < \text{arity } f)$ "
  { fix  $\tau$  assume "?P  $\tau$ " hence "{ }  $\vdash_c$  fresh_pgwt S  $\tau$ "
  proof (induction  $\tau$ )
    case (Var a)
    let ?P = " $\lambda c. c \notin S \wedge \Gamma$  (Fun c []) = Var a  $\wedge$  public c"
    let ?Q = " $\lambda c. \Gamma$  (Fun c []) = Var a  $\wedge$  public c"
    have " {c. ?Q c} - S = {c. ?P c}" by auto
    hence "infinite {c. ?P c}"
      using Diff_infinite_finite[OF assms(1) infinite_typed_consts[of a]]
      by metis
    hence " $\exists c. ?P c$ " using not_finite_existsD by blast
    thus ?case
      using someI_ex[of ?P] intruder_synth.ComposeC[of "[]" _ "{}"] const_type_inv
      by auto
  next
    case (Fun f T)
    have f: "0 < arity f" "length T = arity f" "public f"
      using Fun.premis fun_type_inv unfolding wf_trm_def by auto
    have " $\bigwedge t. t \in \text{set } T \implies ?P t$ "
      using Fun.premis wf_trm_subtermeq term.le_less_trans Fun_param_is_subterm
      by metis
    hence " $\bigwedge t. t \in \text{set } T \implies \{ \} \vdash_c \text{fresh\_pgwt } S t$ " using Fun.premis Fun.IH by auto
    moreover have "length (map (fresh_pgwt S) T) = arity f" using f(2) by auto
    ultimately show ?case using intruder_synth.ComposeC[of "map (fresh_pgwt S) T" f] f by auto
  qed
} thus ?thesis using assms(1)  $\Gamma_{wf}'$ [OF assms(2)]  $\Gamma_{wf}$ (1) by auto
qed

private lemma fresh_pgwt_has_fresh_const:
  assumes "finite S" "wf_trm t"
  obtains c where "Fun c []  $\sqsubseteq$  fresh_pgwt S ( $\Gamma$  t)" "c  $\notin$  S"
proof -
  let ?P = " $\lambda \tau :: ('fun, 'atom) \text{term\_type}. wf_{trm} \tau \wedge (\forall f T. TComp f T \sqsubseteq \tau \longrightarrow 0 < \text{arity } f)$ "
  { fix  $\tau$  assume "?P  $\tau$ " hence " $\exists c. \text{Fun } c [] \sqsubseteq \text{fresh\_pgwt } S \tau \wedge c \notin S$ "
  proof (induction  $\tau$ )

```

```

case (Var a)
let ?P = "λc. c ∉ S ∧ Γ (Fun c []) = Var a ∧ public c"
let ?Q = "λc. Γ (Fun c []) = Var a ∧ public c"
have "{c. ?Q c} - S = {c. ?P c}" by auto
hence "infinite {c. ?P c}"
  using Diff_infinite_finite[OF assms(1) infinite_typed_consts[of a]]
  by metis
hence "∃c. ?P c" using not_finite_existsD by blast
thus ?case using someI_ex[of ?P] by auto
next
case (Fun f T)
have f: "0 < arity f" "length T = arity f" "public f" "T ≠ []"
  using Fun.premis fun_type_inv unfolding wf_trm_def by auto
obtain t' where t': "t' ∈ set T" by (meson all_not_in_conv f(4) set_empty)
have "∧t. t ∈ set T ⇒ ?P t"
  using Fun.premis wf_trm_subtermeq term.le_less_trans Fun_param_is_subterm
  by metis
hence "∧t. t ∈ set T ⇒ ∃c. Fun c [] ⊆ fresh_pgwt S t ∧ c ∉ S"
  using Fun.premis Fun.IH by auto
then obtain c where c: "Fun c [] ⊆ fresh_pgwt S t'" "c ∉ S" using t' by metis
thus ?case using t' by auto
qed
} thus ?thesis using that assms Γ_wf'[OF assms(2)] Γ_wf(1) by blast
qed

private lemma fresh_pgwt_subterm_fresh:
  assumes "finite S" "wf_trm t" "wf_trm s" "funs_term s ⊆ S"
  shows "s ∉ subterms (fresh_pgwt S (Γ t))"
proof -
let ?P = "λτ::('fun,'atom) term_type. wf_trm τ ∧ (∀f T. TComp f T ⊆ τ ⇒ 0 < arity f)"
{ fix τ assume "?P τ" hence "s ∉ subterms (fresh_pgwt S τ)"
  proof (induction τ)
    case (Var a)
    let ?P = "λc. c ∉ S ∧ Γ (Fun c []) = Var a ∧ public c"
    let ?Q = "λc. Γ (Fun c []) = Var a ∧ public c"
    have "{c. ?Q c} - S = {c. ?P c}" by auto
    hence "infinite {c. ?P c}"
      using Diff_infinite_finite[OF assms(1) infinite_typed_consts[of a]]
      by metis
    hence "∃c. ?P c" using not_finite_existsD by blast
    thus ?case using someI_ex[of ?P] assms(4) by auto
  next
    case (Fun f T)
    have f: "0 < arity f" "length T = arity f" "public f"
      using Fun.premis fun_type_inv unfolding wf_trm_def by auto
    have "∧t. t ∈ set T ⇒ ?P t"
      using Fun.premis wf_trm_subtermeq term.le_less_trans Fun_param_is_subterm
      by metis
    hence "∧t. t ∈ set T ⇒ s ∉ subterms (fresh_pgwt S t)" using Fun.premis Fun.IH by auto
    moreover have "s ≠ fresh_pgwt S (Fun f T)"
    proof -
      obtain c where c: "Fun c [] ⊆ fresh_pgwt S (Fun f T)" "c ∉ S"
        using fresh_pgwt_has_fresh_const[OF assms(1)] type_wfttype_inhabited Fun.premis
        by metis
      hence "¬Fun c [] ⊆ s" using assms(4) subtermeq_imp_funs_term_subset by force
      thus ?thesis using c(1) by auto
    qed
    ultimately show ?case by auto
  qed
} thus ?thesis using assms(1) Γ_wf'[OF assms(2)] Γ_wf(1) by auto
qed

private lemma wt_fresh_pgwt_term_exists:

```

```

assumes "finite T" "wftrm s" "wftrms T"
obtains t where "Γ t = Γ s" "{ } ⊢c t" "∀ s ∈ T. ∀ u ∈ subterms s. u ∉ subterms t"
proof -
  have finite_S: "finite (⋃ (funs_term ' T))" using assms(1) by auto

  have 1: "Γ (fresh_pgwt (⋃ (funs_term ' T)) (Γ s)) = Γ s"
    using fresh_pgwt_same_type[OF finite_S assms(2)] by auto

  have 2: "{ } ⊢c fresh_pgwt (⋃ (funs_term ' T)) (Γ s)"
    using fresh_pgwt_empty_synth[OF finite_S assms(2)] by auto

  have 3: "∀ v ∈ T. ∀ u ∈ subterms v. u ∉ subterms (fresh_pgwt (⋃ (funs_term ' T)) (Γ s))"
    using fresh_pgwt_subterm_fresh[OF finite_S assms(2)] assms(3)
    wf_trm_subtermeq subtermeq_imp_funs_term_subset
    by force

  show ?thesis by (rule that[OF 1 2 3])
qed

lemma wt_bij_finite_subst_exists:
  assumes "finite (S::'var set)" "finite (T::('fun,'var) terms)" "wftrms T"
  shows "∃ σ::('fun,'var) subst.
    subst_domain σ = S
    ∧ bij_betw σ (subst_domain σ) (subst_range σ)
    ∧ subtermsset (subst_range σ) ⊆ {t. { } ⊢c t} - T
    ∧ (∀ s ∈ subst_range σ. ∀ u ∈ subst_range σ. (∃ v. v ⊆ s ∧ v ⊆ u) → s = u)
    ∧ wtsubst σ
    ∧ wftrms (subst_range σ)"
using assms
proof (induction rule: finite_induct)
  case empty
  have "subst_domain Var = { }"
    "bij_betw Var (subst_domain Var) (subst_range Var)"
    "subtermsset (subst_range Var) ⊆ {t. { } ⊢c t} - T"
    "∀ s ∈ subst_range Var. ∀ u ∈ subst_range Var. (∃ v. v ⊆ s ∧ v ⊆ u) → s = u"
    "wtsubst Var"
    "wftrms (subst_range Var)"
  unfolding bij_betw_def
  by auto
  thus ?case by (force simp add: subst_domain_def)
next
  case (insert x S)
  then obtain σ where σ:
    "subst_domain σ = S" "bij_betw σ (subst_domain σ) (subst_range σ)"
    "subtermsset (subst_range σ) ⊆ {t. { } ⊢c t} - T"
    "∀ s ∈ subst_range σ. ∀ u ∈ subst_range σ. (∃ v. v ⊆ s ∧ v ⊆ u) → s = u"
    "wtsubst σ" "wftrms (subst_range σ)"
  by (auto simp del: subst_range.simps)

  have *: "finite (T ∪ subst_range σ)"
    using insert.premis(1) insert.hyps(1) σ(1) by simp
  have **: "wftrm (Var x)" by simp
  have ***: "wftrms (T ∪ subst_range σ)" using assms(3) σ(6) by blast
  obtain t where t:
    "Γ t = Γ (Var x)" "{ } ⊢c t"
    "∀ s ∈ T ∪ subst_range σ. ∀ u ∈ subterms s. u ∉ subterms t"
    using wt_fresh_pgwt_term_exists[OF * ** ***] by auto

  obtain ϑ where ϑ: "ϑ ≡ λy. if x = y then t else σ y" by simp

  have t_ground: "fv t = { }" using t(2) pgwt_ground[of t] pgwt_is_empty_synth[of t] by auto
  hence x_dom: "x ∉ subst_domain σ" "x ∈ subst_domain ϑ" using insert.hyps(2) σ(1) ϑ by auto
  moreover have "subst_range σ ⊆ subtermsset (subst_range σ)" by auto

```

```

hence ground_imgs: "ground (subst_range  $\sigma$ )"
  using  $\sigma(3)$  pgwt_ground pgwt_is_empty_synth
  by force
ultimately have x_img: " $\sigma x \notin \text{subst\_range } \sigma$ "
  using ground_subst_dom_iff_img
  by (auto simp add: subst_domain_def)

have "ground (insert t (subst_range  $\sigma$ ))"
  using ground_imgs x_dom t_ground
  by auto
have  $\vartheta\_dom$ : "subst_domain  $\vartheta = \text{insert } x (\text{subst\_domain } \sigma)$ "
  using  $\vartheta$  t_ground by (auto simp add: subst_domain_def)
have  $\vartheta\_img$ : "subst_range  $\vartheta = \text{insert } t (\text{subst\_range } \sigma)$ "
proof
  show "subst_range  $\vartheta \subseteq \text{insert } t (\text{subst\_range } \sigma)$ "
  proof
    fix t' assume "t'  $\in$  subst_range  $\vartheta$ "
    then obtain y where "y  $\in$  subst_domain  $\vartheta$ " "t' =  $\vartheta$  y" by auto
    thus "t'  $\in$  insert t (subst_range  $\sigma$ )" using  $\vartheta$  by (auto simp add: subst_domain_def)
  qed
  show "insert t (subst_range  $\sigma$ )  $\subseteq$  subst_range  $\vartheta$ "
  proof
    fix t' assume t': "t'  $\in$  insert t (subst_range  $\sigma$ )"
    hence "fv t' = {}" using ground_imgs x_img t_ground by auto
    hence "t'  $\neq$  Var x" by auto
    show "t'  $\in$  subst_range  $\vartheta$ "
    proof (cases "t' = t")
      case False
      hence "t'  $\in$  subst_range  $\sigma$ " using t' by auto
      then obtain y where " $\sigma$  y  $\in$  subst_range  $\sigma$ " "t' =  $\sigma$  y" by auto
      hence "y  $\in$  subst_domain  $\sigma$ " "t'  $\neq$  Var y"
        using ground_subst_dom_iff_img[OF ground_imgs(1)]
        by (auto simp add: subst_domain_def simp del: subst_range.simps)
      hence "x  $\neq$  y" using x_dom by auto
      hence " $\vartheta$  y =  $\sigma$  y" unfolding  $\vartheta$  by auto
      thus ?thesis using <t'  $\neq$  Var y> <t' =  $\sigma$  y> subst_imgI[of  $\vartheta$  y] by auto
    qed (metis subst_imgI  $\vartheta$  <t'  $\neq$  Var x>)
  qed
qed
hence  $\vartheta\_ground\_img$ : "ground (subst_range  $\vartheta$ )"
  using ground_imgs t_ground
  by auto

have "subst_domain  $\vartheta = \text{insert } x S$ " using  $\vartheta\_dom$   $\sigma(1)$  by auto
moreover have "bij_betw  $\vartheta$  (subst_domain  $\vartheta$ ) (subst_range  $\vartheta$ )"
proof (intro bij_betwI')
  fix y z assume *: "y  $\in$  subst_domain  $\vartheta$ " "z  $\in$  subst_domain  $\vartheta$ "
  hence "fv ( $\vartheta$  y) = {}" "fv ( $\vartheta$  z) = {}" using  $\vartheta\_ground\_img$  by auto
  { assume " $\vartheta$  y =  $\vartheta$  z" hence "y = z"
    proof (cases " $\vartheta$  y  $\in$  subst_range  $\sigma \wedge \vartheta$  z  $\in$  subst_range  $\sigma$ ")
      case True
      hence **: "y  $\in$  subst_domain  $\sigma$ " "z  $\in$  subst_domain  $\sigma$ "
        using  $\vartheta$   $\vartheta\_dom$  True * t(3) by (metis Un_iff term.order_refl insertE)+
      hence "y  $\neq$  x" "z  $\neq$  x" using x_dom by auto
      hence " $\vartheta$  y =  $\sigma$  y" " $\vartheta$  z =  $\sigma$  z" using  $\vartheta$  by auto
      thus ?thesis using < $\vartheta$  y =  $\vartheta$  z>  $\sigma(2)$  ** unfolding bij_betw_def inj_on_def by auto
    qed (metis  $\vartheta$  * < $\vartheta$  y =  $\vartheta$  z>  $\vartheta\_dom$  ground_imgs(1) ground_subst_dom_iff_img insertE)
  }
  thus "< $\vartheta$  y =  $\vartheta$  z> = (y = z)" by auto
next
  fix y assume "y  $\in$  subst_domain  $\vartheta$ " thus " $\vartheta$  y  $\in$  subst_range  $\vartheta$ " by auto
next
  fix t assume "t  $\in$  subst_range  $\vartheta$ " thus " $\exists z \in \text{subst\_domain } \vartheta. t = \vartheta z$ " by auto

```

```

qed
moreover have "subtermsset (subst_range  $\vartheta$ )  $\subseteq$  {t. {}  $\vdash_c$  t} - T"
proof -
  { fix s assume "s  $\sqsubseteq$  t"
    hence "s  $\in$  {t. {}  $\vdash_c$  t} - T"
      using t(2,3)
      by (metis Diff_eq_empty_iff Diff_iff Un_upper1 term.order_refl
        deduct_synth_subterm mem_Collect_eq)
  } thus ?thesis using  $\sigma$ (3)  $\vartheta$   $\vartheta\_img$  by auto
qed
moreover have "wtsubst  $\vartheta$ " using  $\vartheta$  t(1)  $\sigma$ (5) unfolding wtsubst_def by auto
moreover have "wftrms (subst_range  $\vartheta$ )"
  using  $\vartheta$   $\sigma$ (6) t(2) pgwt_is_empty_synth pgwt_wellformed
  wf_trm_subst_range_iff[of  $\sigma$ ] wf_trm_subst_range_iff[of  $\vartheta$ ]
  by metis
moreover have " $\forall s \in \text{subst\_range } \vartheta. \forall u \in \text{subst\_range } \vartheta. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u$ "
  using  $\sigma$ (4)  $\vartheta\_img$  t(3) by (auto simp del: subst_range.simps)
ultimately show ?case by blast
qed

private lemma wt_bij_finite_tatom_subst_exists_single:
  assumes "finite (S::'var set)" "finite (T::('fun,'var) terms)"
  and " $\bigwedge x. x \in S \implies \Gamma (\text{Var } x) = \text{TAtom } a$ "
  shows " $\exists \sigma::('fun,'var) \text{subst. subst\_domain } \sigma = S$ 
     $\wedge$  bij_betw  $\sigma$  (subst_domain  $\sigma$ ) (subst_range  $\sigma$ )
     $\wedge$  subst_range  $\sigma \subseteq ((\lambda c. \text{Fun } c []) ' \{c. \Gamma (\text{Fun } c []) = \text{TAtom } a \wedge$ 
      public c  $\wedge$  arity c = 0}) - T
     $\wedge$  wtsubst  $\sigma$ 
     $\wedge$  wftrms (subst_range  $\sigma$ )"

proof -
  let ?U = "{c.  $\Gamma (\text{Fun } c []) = \text{TAtom } a \wedge$  public c  $\wedge$  arity c = 0}"

  obtain  $\sigma$  where  $\sigma$ :
    "subst_domain  $\sigma = S$ " "bij_betw  $\sigma$  (subst_domain  $\sigma$ ) (subst_range  $\sigma$ )"
    "subst_range  $\sigma \subseteq ((\lambda c. \text{Fun } c []) ' ?U)$  - T"
  using bij_finite_const_subst_exists'[OF assms(1,2) infinite_typed_consts'[of a]]
  by auto

  { fix x assume "x  $\notin$  subst_domain  $\sigma$ " hence " $\Gamma (\text{Var } x) = \Gamma (\sigma x)$ " by auto }
  moreover
  { fix x assume "x  $\in$  subst_domain  $\sigma$ "
    hence " $\exists c \in ?U. \sigma x = \text{Fun } c [] \wedge$  arity c = 0" using  $\sigma$  by auto
    hence " $\Gamma (\sigma x) = \text{TAtom } a$ " "wftrm ( $\sigma x$ )" using assms(3) const_type wf_trmI[of "[]"] by auto
    hence " $\Gamma (\text{Var } x) = \Gamma (\sigma x)$ " "wftrm ( $\sigma x$ )" using assms(3)  $\sigma$ (1) by force+
  }
  ultimately have "wtsubst  $\sigma$ " "wftrms (subst_range  $\sigma$ )"
    using wf_trm_subst_range_iff[of  $\sigma$ ]
    unfolding wtsubst_def
    by force+
  thus ?thesis using  $\sigma$  by auto
qed

lemma wt_bij_finite_tatom_subst_exists:
  assumes "finite (S::'var set)" "finite (T::('fun,'var) terms)"
  and " $\bigwedge x. x \in S \implies \exists a. \Gamma (\text{Var } x) = \text{TAtom } a$ "
  shows " $\exists \sigma::('fun,'var) \text{subst. subst\_domain } \sigma = S$ 
     $\wedge$  bij_betw  $\sigma$  (subst_domain  $\sigma$ ) (subst_range  $\sigma$ )
     $\wedge$  subst_range  $\sigma \subseteq ((\lambda c. \text{Fun } c []) ' \mathcal{C}_{pub})$  - T
     $\wedge$  wtsubst  $\sigma$ 
     $\wedge$  wftrms (subst_range  $\sigma$ )"

using assms
proof (induction rule: finite_induct)
  case empty

```

```

have "subst_domain Var = {}"
  "bij_betw Var (subst_domain Var) (subst_range Var)"
  "subst_range Var  $\subseteq ((\lambda c. \text{Fun } c \text{ []}) ' C_{pub}) - T"$ 
  "wt_subst Var"
  "wf_trms (subst_range Var)"
  unfolding bij_betw_def
  by auto
thus ?case by (auto simp add: subst_domain_def)
next
case (insert x S)
then obtain a where a: " $\Gamma$  (Var x) = TAtom a" by fastforce

from insert obtain  $\sigma$  where  $\sigma$ :
  "subst_domain  $\sigma$  = S" "bij_betw  $\sigma$  (subst_domain  $\sigma$ ) (subst_range  $\sigma$ )"
  "subst_range  $\sigma \subseteq ((\lambda c. \text{Fun } c \text{ []}) ' C_{pub}) - T"$  "wt_subst  $\sigma$ "
  "wf_trms (subst_range  $\sigma$ )"
  by auto

let ?S' = "{y  $\in$  S.  $\Gamma$  (Var y) = TAtom a}"
let ?T' = "T  $\cup$  subst_range  $\sigma$ "

have *: "finite (insert x ?S')" using insert by simp
have **: "finite ?T'" using insert.premis(1) insert.hyps(1)  $\sigma$ (1) by simp
have ***: " $\bigwedge y. y \in \text{insert } x \text{ ?S}' \implies \Gamma$  (Var y) = TAtom a" using a by auto

obtain  $\delta$  where  $\delta$ :
  "subst_domain  $\delta$  = insert x ?S'" "bij_betw  $\delta$  (subst_domain  $\delta$ ) (subst_range  $\delta$ )"
  "subst_range  $\delta \subseteq ((\lambda c. \text{Fun } c \text{ []}) ' C_{pub}) - ?T'"$  "wt_subst  $\delta$ " "wf_trms (subst_range  $\delta$ )"
  using wt_bij_finite_tatom_subst_exists_single[OF * ** ***] const_type_inv[of _ "[]" a]
  by blast

obtain  $\vartheta$  where  $\vartheta$ : " $\vartheta \equiv \lambda y. \text{if } x = y \text{ then } \delta \ y \text{ else } \sigma \ y$ " by simp

have x_dom: "x  $\notin$  subst_domain  $\sigma$ " "x  $\in$  subst_domain  $\delta$ " "x  $\in$  subst_domain  $\vartheta$ "
  using insert.hyps(2)  $\sigma$ (1)  $\delta$ (1)  $\vartheta$  by (auto simp add: subst_domain_def)
moreover have ground_imgs: "ground (subst_range  $\sigma$ )" "ground (subst_range  $\delta$ )"
  using pgwt_ground  $\sigma$ (3)  $\delta$ (3) by auto
ultimately have x_img: " $\sigma \ x \notin$  subst_range  $\sigma$ " " $\delta \ x \in$  subst_range  $\delta$ "
  using ground_subst_dom_iff_img by (auto simp add: subst_domain_def)

have "ground (insert ( $\delta \ x$ ) (subst_range  $\sigma$ ))" using ground_imgs x_dom by auto
have  $\vartheta$ _dom: "subst_domain  $\vartheta$  = insert x (subst_domain  $\sigma$ )"
  using  $\delta$ (1)  $\vartheta$  by (auto simp add: subst_domain_def)
have  $\vartheta$ _img: "subst_range  $\vartheta$  = insert ( $\delta \ x$ ) (subst_range  $\sigma$ )"
proof
  show "subst_range  $\vartheta \subseteq$  insert ( $\delta \ x$ ) (subst_range  $\sigma$ )"
  proof
    fix t assume "t  $\in$  subst_range  $\vartheta$ "
    then obtain y where "y  $\in$  subst_domain  $\vartheta$ " "t =  $\vartheta \ y$ " by auto
    thus "t  $\in$  insert ( $\delta \ x$ ) (subst_range  $\sigma$ )" using  $\vartheta$  by (auto simp add: subst_domain_def)
  qed
  show "insert ( $\delta \ x$ ) (subst_range  $\sigma$ )  $\subseteq$  subst_range  $\vartheta$ "
  proof
    fix t assume t: "t  $\in$  insert ( $\delta \ x$ ) (subst_range  $\sigma$ )"
    hence "fv t = {}" using ground_imgs x_img(2) by auto
    hence "t  $\neq$  Var x" by auto
    show "t  $\in$  subst_range  $\vartheta$ "
    proof (cases "t =  $\delta \ x$ ")
      case True thus ?thesis using subst_imgI  $\vartheta$  (t  $\neq$  Var x) by metis
    next
      case False
      hence "t  $\in$  subst_range  $\sigma$ " using t by auto
      then obtain y where " $\sigma \ y \in$  subst_range  $\sigma$ " "t =  $\sigma \ y$ " by auto

```

```

hence "y ∈ subst_domain σ" "t ≠ Var y"
  using ground_subst_dom_iff_img[OF ground_imgs(1)]
  by (auto simp add: subst_domain_def simp del: subst_range.simps)
hence "x ≠ y" using x_dom by auto
hence "∅ y = σ y" unfolding ∅ by auto
thus ?thesis using (t ≠ Var y) (t = σ y) subst_imgI[of ∅ y] by auto
qed
qed
qed
hence ∅_ground_img: "ground (subst_range ∅)" using ground_imgs x_img by auto

have "subst_domain ∅ = insert x S" using ∅_dom σ(1) by auto
moreover have "bij_betw ∅ (subst_domain ∅) (subst_range ∅)"
proof (intro bij_betwI')
  fix y z assume *: "y ∈ subst_domain ∅" "z ∈ subst_domain ∅"
  hence "fv (∅ y) = {}" "fv (∅ z) = {}" using ∅_ground_img by auto
  { assume "∅ y = ∅ z" hence "y = z"
  proof (cases "∅ y ∈ subst_range σ ∧ ∅ z ∈ subst_range σ")
    case True
    hence **: "y ∈ subst_domain σ" "z ∈ subst_domain σ"
    using ∅ ∅_dom x_img(2) δ(3) True
    by (metis (no_types) *(1) DiffE Un_upper2 insertE subsetCE,
        metis (no_types) *(2) DiffE Un_upper2 insertE subsetCE)
    hence "y ≠ x" "z ≠ x" using x_dom by auto
    hence "∅ y = σ y" "∅ z = σ z" using ∅ by auto
    thus ?thesis using (∅ y = ∅ z) σ(2) ** unfolding bij_betw_def inj_on_def by auto
  qed (metis ∅ * (∅ y = ∅ z) ∅_dom ground_imgs(1) ground_subst_dom_iff_img insertE)
  }
  thus "(∅ y = ∅ z) = (y = z)" by auto
next
  fix y assume "y ∈ subst_domain ∅" thus "∅ y ∈ subst_range ∅" by auto
next
  fix t assume "t ∈ subst_range ∅" thus "∃ z ∈ subst_domain ∅. t = ∅ z" by auto
qed
moreover have "subst_range ∅ ⊆ (λc. Fun c []) ' Cpub - T"
  using σ(3) δ(3) ∅ by (auto simp add: subst_domain_def)
moreover have "wtsubst ∅" using σ(4) δ(4) ∅ unfolding wtsubst_def by auto
moreover have "wftrms (subst_range ∅)"
  using ∅ σ(5) δ(5) wf_trm_subst_range_iff[of δ]
  wf_trm_subst_range_iff[of σ] wf_trm_subst_range_iff[of ∅]
  by presburger
ultimately show ?case by blast
qed

theorem wt_sat_if_simple:
  assumes "simple S" "wfconstr S ∅" "wtsubst ∅" "wftrms (subst_range ∅)" "wftrms (trmsst S)"
  and I': "∀ X F. Inequality X F ∈ set S ⟶ ineq_model I' X F"
  "ground (subst_range I)"
  "subst_domain I' = {x ∈ varsst S. ∃ X F. Inequality X F ∈ set S ∧ x ∈ fvpairs F - set X}"
  and tfr_stp_all: "list_all tfr_stp S"
  shows "∃ I. interpretationsubst I ∧ (I ⊨c ⟨S, ∅⟩) ∧ wtsubst I ∧ wftrms (subst_range I)"
proof -
  from (wfconstr S ∅) have "wfst {} S" "subst_idem ∅" and S_∅_disj: "∀ v ∈ varsst S. ∅ v = Var v"
  using subst_idemI[of ∅] unfolding wfconstr_def wfsubst_def by force+

  obtain I::('fun, 'var) subst"
  where I: "interpretationsubst I" "wtsubst I" "subst_range I ⊆ public_ground_wf_terms"
  using wt_interpretation_exists by blast
  hence I_deduct: "∧ x M. M ⊢c I x" and I_wf_trm: "wftrms (subst_range I)"
  using pgwt_deducible pgwt_wellformed by fastforce+

  let ?P = "λδ X. subst_domain δ = set X ∧ ground (subst_range δ)"
  let ?Sineqvars = "{x ∈ varsst S. ∃ X F. Inequality X F ∈ set S ∧ x ∈ fvpairs F ∧ x ∉ set X}"

```

```

let ?Strms = "subtermsset (trmsst S)"

have finite_vars: "finite ?Sineqvars" "finite ?Strms" "wftrms ?Strms"
  using wf_trm_subtermeq assms(5) by fastforce+

define Q1 where "Q1 = (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀x ∈ fvpairs F - set X. ∃a. Γ (Var x) = TAtom a)"

define Q2 where "Q2 = (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀f T. Fun f T ∈ subtermsset (trmspairs F) → T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X))"

define Q1' where "Q1' = (λ(t::('fun,'var) term) (t'::('fun,'var) term) X.
  ∀x ∈ (fv t ∪ fv t') - set X. ∃a. Γ (Var x) = TAtom a)"

define Q2' where "Q2' = (λ(t::('fun,'var) term) (t'::('fun,'var) term) X.
  ∀f T. Fun f T ∈ subterms t ∪ subterms t' → T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X))"

have ex_P: "∀X. ∃δ. ?P δ X" using interpretation_subst_exists' by blast

have tfr_ineq: "∀X F. Inequality X F ∈ set S → Q1 F X ∨ Q2 F X"
  using tfr_stp_all Q1_def Q2_def tfr_stp_list_all_alt_def[of S] by blast

have S_fv_bvars_disj: "fvst S ∩ bvarsst S = {}" using ⟨wfconstr S ∅⟩ unfolding wfconstr_def by metis
hence ineqs_vars_not_bound: "∀X F x. Inequality X F ∈ set S → x ∈ ?Sineqvars → x ∉ set X"
  using strand_fv_bvars_disjoint_unfold by blast

have ∅_vars_S_bvars_disj: "(subst_domain ∅ ∪ range_vars ∅) ∩ set X = {}"
  when "Inequality X F ∈ set S" for F X
  using wfconstr_bvars_disj[OF ⟨wfconstr S ∅⟩]
    strand_fv_bvars_disjointD(1)[OF S_fv_bvars_disj that]
  by blast

obtain σ::('fun,'var) subst
  where σ_fv_dom: "subst_domain σ = ?Sineqvars"
  and σ_subterm_inj: "subterm_inj_on σ (subst_domain σ)"
  and σ_fresh_pub_img: "subtermsset (subst_range σ) ⊆ {t. {} ⊢c t} - ?Strms"
  and σ_wt: "wtsubst σ"
  and σ_wf_trm: "wftrms (subst_range σ)"
  using wt_bij_finite_subst_exists[OF finite_vars]
    subst_inj_on_is_bij_betw_subterm_inj_on_alt_def'
  by moura

have σ_bij_dom_img: "bij_betw σ (subst_domain σ) (subst_range σ)"
  by (metis σ_subterm_inj subst_inj_on_is_bij_betw_subterm_inj_on_alt_def)

have "finite (subst_domain σ)" by (metis σ_fv_dom finite_vars(1))
hence σ_finite_img: "finite (subst_range σ)" using σ_bij_dom_img bij_betw_finite by blast

have σ_img_subterms: "∀s ∈ subst_range σ. ∀u ∈ subst_range σ. (∃v. v ⊆ s ∧ v ⊆ u) → s = u"
  by (metis σ_subterm_inj subterm_inj_on_alt_def')

have "subst_range σ ⊆ subtermsset (subst_range σ)" by auto
hence "subst_range σ ⊆ public_ground_wf_terms - ?Strms"
  and σ_pgwt_img:
    "subst_range σ ⊆ public_ground_wf_terms"
    "subtermsset (subst_range σ) ⊆ public_ground_wf_terms"
  using σ_fresh_pub_img pgwt_is_empty_synth by blast+

have σ_img_ground: "ground (subst_range σ)"
  using σ_pgwt_img pgwt_ground by auto
hence σ_inj: "inj σ"
  using σ_bij_dom_img subst_inj_is_bij_betw_dom_img_if_ground_img by auto

```



```

have  $\sigma_{\text{ineqs\_fv\_dom}}$ : " $\bigwedge X F$ . Inequality  $X F \in \text{set } S \implies \text{fv}_{\text{pairs}} F - \text{set } X \subseteq \text{subst\_domain } \sigma$ "
  using  $\sigma_{\text{fv\_dom}}$  by fastforce

have  $\sigma_{\text{dom\_bvars\_disj}}$ : " $\forall X F$ . Inequality  $X F \in \text{set } S \longrightarrow \text{subst\_domain } \sigma \cap \text{set } X = \{\}$ "
  using  $\text{ineqs\_vars\_not\_bound } \sigma_{\text{fv\_dom}}$  by fastforce

have  $\mathcal{I}'1$ : " $\forall X F \delta$ . Inequality  $X F \in \text{set } S \longrightarrow \text{fv}_{\text{pairs}} F - \text{set } X \subseteq \text{subst\_domain } \mathcal{I}'$ "
  using  $\mathcal{I}'(3)$   $\text{ineqs\_vars\_not\_bound}$  by fastforce

have  $\mathcal{I}'2$ : " $\forall X F$ . Inequality  $X F \in \text{set } S \longrightarrow \text{subst\_domain } \mathcal{I}' \cap \text{set } X = \{\}$ "
  using  $\mathcal{I}'(3)$   $\text{ineqs\_vars\_not\_bound}$  by blast

have  $\text{doms\_eq}$ : " $\text{subst\_domain } \mathcal{I}' = \text{subst\_domain } \sigma$ " using  $\mathcal{I}'(3)$   $\sigma_{\text{fv\_dom}}$  by simp

have  $\sigma_{\text{ineqs\_neq}}$ : " $\text{ineq\_model } \sigma X F$ " when " $\text{Inequality } X F \in \text{set } S$ " for  $X F$ 
proof -
  obtain  $a$ : "'fun" where  $a$ : " $a \notin \bigcup (\text{funs\_term } ' \text{subterms}_{\text{set}} (\text{subst\_range } \sigma))$ "
    using  $\text{exists\_fun\_notin\_funs\_terms}[OF \text{subterms\_union\_finite}[OF \sigma_{\text{finite\_img}}]]$ 
    by moura
  hence  $a'$ : " $\bigwedge T$ .  $\text{Fun } a T \notin \text{subterms}_{\text{set}} (\text{subst\_range } \sigma)$ "
    " $\bigwedge S$ .  $\text{Fun } a [] \in \text{set } (\text{Fun } a [] \# S)$ " " $\text{Fun } a [] \notin \text{Var } ' \text{set } X$ "
    by ( $\text{meson } a \text{ UN\_I term.set\_intros}(1)$ , auto)

  define  $t$  where " $t \equiv \text{Fun } a (\text{Fun } a [] \# \text{map fst } F)$ "
  define  $t'$  where " $t' \equiv \text{Fun } a (\text{Fun } a [] \# \text{map snd } F)$ "

  note  $F_{\text{in}} = \text{that}$ 

  have  $t_{\text{fv}}$ : " $\text{fv } t \cup \text{fv } t' \subseteq \text{fv}_{\text{pairs}} F$ "
    unfolding  $t_{\text{def}}$   $t'_{\text{def}}$  by force

  have  $t_{\text{subterms}}$ : " $\text{subterms } t \cup \text{subterms } t' \subseteq \text{subterms}_{\text{set}} (\text{trms}_{\text{pairs}} F) \cup \{t, t', \text{Fun } a []\}$ "
    unfolding  $t_{\text{def}}$   $t'_{\text{def}}$  by force

  have " $t \cdot \delta \cdot \sigma \neq t' \cdot \delta \cdot \sigma$ " when " $?P \delta X$ " for  $\delta$ 
  proof -
    have  $t_{\text{fr\_assms}}$ : " $Q1 F X \vee Q2 F X$ " using  $t_{\text{fr\_ineq}} F_{\text{in}}$  by metis

    have " $Q1 F X \implies \forall x \in \text{fv}_{\text{pairs}} F - \text{set } X. \exists c. \sigma x = \text{Fun } c []$ "
    proof
      fix  $x$  assume " $Q1 F X$ " and  $x$ : " $x \in \text{fv}_{\text{pairs}} F - \text{set } X$ "
      then obtain  $a$  where " $\Gamma (\text{Var } x) = T_{\text{Atom}} a$ " unfolding  $Q1_{\text{def}}$  by moura
      hence  $a$ : " $\Gamma (\sigma x) = T_{\text{Atom}} a$ " using  $\sigma_{\text{wt}}$  unfolding  $\text{wt}_{\text{subst\_def}}$  by simp

      have " $x \in \text{subst\_domain } \sigma$ " using  $\sigma_{\text{ineqs\_fv\_dom}}$   $x F_{\text{in}}$  by auto
      then obtain  $f T$  where  $fT$ : " $\sigma x = \text{Fun } f T$ " by ( $\text{meson } \sigma_{\text{img\_ground ground\_img\_obtain\_fun}}$ )
      hence " $T = []$ " using  $\sigma_{\text{wf\_trm}}$   $a T_{\text{Atom\_term\_cases}}$  by fastforce
      thus " $\exists c. \sigma x = \text{Fun } c []$ " using  $fT$  by metis
    qed

    hence 1: " $Q1 F X \implies \forall x \in (\text{fv } t \cup \text{fv } t') - \text{set } X. \exists c. \sigma x = \text{Fun } c []$ "
      using  $t_{\text{fv}}$  by auto

    have 2: " $\neg Q1 F X \implies Q2 F X$ " by (metis  $t_{\text{fr\_assms}}$ )

    have 3: " $\text{subst\_domain } \sigma \cap \text{set } X = \{\}$ " using  $\sigma_{\text{dom\_bvars\_disj}}$   $F_{\text{in}}$  by auto

    have 4: " $\text{subterms}_{\text{set}} (\text{subst\_range } \sigma) \cap (\text{subterms } t \cup \text{subterms } t') = \{\}$ "
    proof -
      define  $M1$  where " $M1 \equiv \{t, t', \text{Fun } a []\}$ "
      define  $M2$  where " $M2 \equiv ?\text{Strms}$ "

      have " $\text{subterms}_{\text{set}} (\text{trms}_{\text{pairs}} F) \subseteq M2$ "
        using  $F_{\text{in}}$  unfolding  $M2_{\text{def}}$  by force

```

```

moreover have "subterms t  $\cup$  subterms t'  $\subseteq$  subtermsset (trmspairs F)  $\cup$  M1"
  using t_subterms unfolding M1_def by blast
ultimately have *: "subterms t  $\cup$  subterms t'  $\subseteq$  M2  $\cup$  M1"
  by auto

have "subtermsset (subst_range  $\sigma$ )  $\cap$  M1 = {}"
  "subtermsset (subst_range  $\sigma$ )  $\cap$  M2 = {}"
  using a'  $\sigma$ _fresh_pub_img
  unfolding t_def t'_def M1_def M2_def
  by blast+
thus ?thesis using * by blast
qed

have 5: "(fv t  $\cup$  fv t') - subst_domain  $\sigma \subseteq$  set X"
  using  $\sigma$ _ineqs_fv_dom[OF F_in] t_fv
  by auto

have 6: " $\forall \delta. ?P \delta X \longrightarrow t \cdot \delta \cdot \mathcal{I}' \neq t' \cdot \delta \cdot \mathcal{I}'$ "
  by (metis t_def t'_def  $\mathcal{I}'$ (1) F_in ineq_model_singleE ineq_model_single_iff)

have 7: "fv t  $\cup$  fv t' - set X  $\subseteq$  subst_domain  $\mathcal{I}'$ " using  $\mathcal{I}'$ 1 F_in t_fv by force

have 8: "subst_domain  $\mathcal{I}' \cap$  set X = {}" using  $\mathcal{I}'$ 2 F_in by auto

have 9: "Q1' t t' X" when "Q1 F X"
  using that t_fv
  unfolding Q1_def Q1'_def t_def t'_def
  by blast

have 10: "Q2' t t' X" when "Q2 F X" unfolding Q2'_def
proof (intro allI impI)
  fix f T assume "Fun f T  $\in$  subterms t  $\cup$  subterms t'"
  moreover {
    assume "Fun f T  $\in$  subtermsset (trmspairs F)"
    hence "T = []  $\vee$  ( $\exists s \in$  set T. s  $\notin$  Var ' set X)" by (metis Q2_def that)
  } moreover {
    assume "Fun f T = t" hence "T = []  $\vee$  ( $\exists s \in$  set T. s  $\notin$  Var ' set X)"
    unfolding t_def using a'(2,3) by simp
  } moreover {
    assume "Fun f T = t'" hence "T = []  $\vee$  ( $\exists s \in$  set T. s  $\notin$  Var ' set X)"
    unfolding t'_def using a'(2,3) by simp
  } moreover {
    assume "Fun f T = Fun a []" hence "T = []  $\vee$  ( $\exists s \in$  set T. s  $\notin$  Var ' set X)" by simp
  } ultimately show "T = []  $\vee$  ( $\exists s \in$  set T. s  $\notin$  Var ' set X)" using t_subterms by blast
qed

note 11 =  $\sigma$ _subterm_inj  $\sigma$ _img_ground 3 4 5

note 12 = 6 7 8  $\mathcal{I}'$ (2) doms_eq

show "t  $\cdot \delta \cdot \sigma \neq t' \cdot \delta \cdot \sigma$ "
  using 1 2 9 10 that sat_ineq_subterm_inj_subst[OF 11 _ 12]
  unfolding Q1'_def Q2'_def by metis
qed
thus ?thesis by (metis t_def t'_def ineq_model_singleI ineq_model_single_iff)
qed

have  $\sigma$ _ineqs_fv_dom': "fvpairs (F  $\cdot$ pairs  $\delta$ )  $\subseteq$  subst_domain  $\sigma$ "
  when "Inequality X F  $\in$  set S" and "?P  $\delta$  X" for F  $\delta$  X
  using  $\sigma$ _ineqs_fv_dom[OF that(1)]
proof (induction F)
  case (Cons g G)
  obtain t t' where g: "g = (t, t')" by (metis surj_pair)

```

```

hence "fv_pairs (g#G .pairs δ) = fv (t . δ) ∪ fv (t' . δ) ∪ fv_pairs (G .pairs δ)"
      "fv_pairs (g#G) = fv t ∪ fv t' ∪ fv_pairs G"
  by (simp_all add: subst_apply_pairs_def)
moreover have "fv (t . δ) = fv t - subst_domain δ" "fv (t' . δ) = fv t' - subst_domain δ"
  using g that(2) by (simp_all add: subst_fv_unfold_ground_img range_vars_alt_def)
moreover have "fv_pairs (G .pairs δ) ⊆ subst_domain σ" using Cons by auto
ultimately show ?case using Cons.prem1s that(2) by auto
qed (simp add: subst_apply_pairs_def)

have σ_ineqs_ground: "fv_pairs ((F .pairs δ) .pairs σ) = {}"
  when "Inequality X F ∈ set S" and "?P δ X" for F δ X
  using σ_ineqs_fv_dom'[OF that]
proof (induction F)
  case (Cons g G)
  obtain t t' where g: "g = (t,t')" by (metis surj_pair)
  hence "fv (t . δ) ⊆ subst_domain σ" "fv (t' . δ) ⊆ subst_domain σ"
    using Cons.prem1s by (auto simp add: subst_apply_pairs_def)
  hence "fv (t . δ . σ) = {}" "fv (t' . δ . σ) = {}"
    using subst_fv_dom_ground_if_ground_img[OF _ σ_img_ground] by metis+
  thus ?case using g Cons by (auto simp add: subst_apply_pairs_def)
qed (simp add: subst_apply_pairs_def)

from σ_pgwt_img σ_ineqs_neq have σ_deduct: "M ⊢c σ x" when "x ∈ subst_domain σ" for x M
  using that pgwt_deducible by fastforce

{ fix M: "('fun,'var) terms"
  have "[M; S]c (∅ ∘s σ ∘s I)"
    using (wfst {} S) (simple S) S_∅_disj σ_ineqs_neq σ_ineqs_fv_dom' ∅_vars_S_bvars_disj
  proof (induction S arbitrary: M rule: wfst_simple_induct)
    case (ConsSnd v S)
    hence S_sat: "[M; S]c (∅ ∘s σ ∘s I)" and "∅ v = Var v" by auto
    hence "ΛM. M ⊢c Var v . (∅ ∘s σ ∘s I)"
      using I_deduct σ_deduct
      by (metis ideduct_synth_subst_apply subst_apply_term.simps(1)
        subst_subst_compose trm_subst_ident')
    thus ?case using strand_sem_append(1)[OF S_sat] by (metis strand_sem_c.simps(1,2))
  next
    case (ConsIneq X F S)
    have dom_disj: "subst_domain ∅ ∩ fv_pairs F = {}"
      using ConsIneq.prem1s(1) subst_dom_vars_in_subst
      by force
    hence *: "F .pairs ∅ = F" by blast

    have **: "ineq_model σ X F" by (meson ConsIneq.prem1s(2) in_set_conv_decomp)

    have "Λx. x ∈ varsst S ⇒ x ∈ varsst (S@[Inequality X F])"
      "Λx. x ∈ set S ⇒ x ∈ set (S@[Inequality X F])" by auto
    hence IH: "[M; S]c (∅ ∘s σ ∘s I)" by (metis ConsIneq.IH ConsIneq.prem1s(1,2,3,4))

    have "ineq_model (σ ∘s I) X F"
    proof -
      have "fv_pairs (F .pairs δ) ⊆ subst_domain σ" when "?P δ X" for δ
        using ConsIneq.prem1s(3)[OF _ that] by simp
      hence "fv_pairs F - set X ⊆ subst_domain σ"
        using fv_pairs_subst_subset ex_P
        by (metis Diff_subset_conv Un_commute)
      thus ?thesis by (metis ineq_model_ground_subst[OF _ σ_img_ground **])
    qed
    hence "ineq_model (∅ ∘s σ ∘s I) X F"
      using * ineq_model_subst' subst_compose_assoc ConsIneq.prem1s(4)
      by (metis UnCI list.set_intros(1) set_append)
    thus ?case using IH by (auto simp add: ineq_model_def)
  qed auto
}

```

```

}
moreover have "wt_subst ( $\vartheta \circ_s \sigma \circ_s \mathcal{I}$ )" "wf_trms (subst_range ( $\vartheta \circ_s \sigma \circ_s \mathcal{I}$ ))"
  by (metis wt_subst_compose <wt_subst  $\vartheta$ > <wt_subst  $\sigma$ > <wt_subst  $\mathcal{I}$ >,
      metis assms(4)  $\mathcal{I}$ _wf_trm  $\sigma$ _wf_trm wf_trm_subst subst_img_comp_subset')
ultimately show ?thesis
  using interpretation_comp(1)[OF <interpretation_subst  $\mathcal{I}$ >, of " $\vartheta \circ_s \sigma$ "]
      subst_idem_support[OF <subst_idem  $\vartheta$ >, of " $\sigma \circ_s \mathcal{I}$ "] subst_compose_assoc
      unfolding constr_sem_c_def by metis
qed
end

```

### Theorem: Type-flaw resistant constraints are well-typed satisfiable (composition-only)

There exists well-typed models of satisfiable type-flaw resistant constraints in the semantics where the intruder is limited to composition only (i.e., he cannot perform decomposition/analysis of deducible messages).

**theorem** *wt\_attack\_if\_tfr\_attack*:

```

assumes "interpretation_subst  $\mathcal{I}$ "
  and " $\mathcal{I} \models_c \langle S, \vartheta \rangle$ "
  and " $\text{wf}_{constr} S \vartheta$ "
  and " $\text{wt}_{subst} \vartheta$ "
  and " $\text{tfr}_{st} S$ "
  and " $\text{wf}_{trms} (\text{trms}_{st} S)$ "
  and " $\text{wf}_{trms} (\text{subst\_range } \vartheta)$ "
obtains  $\mathcal{I}_\tau$  where "interpretation_subst  $\mathcal{I}_\tau$ "
  and " $\mathcal{I}_\tau \models_c \langle S, \vartheta \rangle$ "
  and " $\text{wt}_{subst} \mathcal{I}_\tau$ "
  and " $\text{wf}_{trms} (\text{subst\_range } \mathcal{I}_\tau)$ "

```

**proof** -

```

have tfr: "tfr_set (trms_st S)" "wf_trms (trms_st S)" "list_all tfr_stp S"
  using assms(5,6) unfolding tfr_st_def by metis+
obtain  $S' \vartheta'$  where *: "simple  $S'$ " "( $S, \vartheta$ )  $\rightsquigarrow^*$  ( $S', \vartheta'$ )" "[[{}];  $S'$ ]c  $\mathcal{I}$ "
  using LI_completeness[OF assms(3,2)] unfolding constr_sem_c_def
  by (meson term.order_refl)
have **: "wf_constr  $S' \vartheta'$ " "wt_subst  $\vartheta'$ " "list_all tfr_stp  $S'$ " "wf_trms (trms_st  $S'$ )" "wf_trms (subst_range
 $\vartheta'$ )"
  using LI_preserves_welltypedness[OF *(2) assms(3,4,7) tfr]
      LI_preserves_wellformedness[OF *(2) assms(3)]
      LI_preserves_tfr[OF *(2) assms(3,4,7) tfr]
  by metis+

```

```

define A where "A  $\equiv \{x \in \text{vars}_{st} S'. \exists X F. \text{Inequality } X F \in \text{set } S' \wedge x \in \text{fv}_{pairs} F \wedge x \notin \text{set } X\}$ "
define B where "B  $\equiv \text{UNIV} - A$ "

```

```

let ? $\mathcal{I}$  = "rm_vars B  $\mathcal{I}$ "

```

```

have gr $\mathcal{I}$ : "ground (subst_range  $\mathcal{I}$ )" "ground (subst_range ? $\mathcal{I}$ )"
  using assms(1) rm_vars_img_subset[of B  $\mathcal{I}$ ] by (auto simp add: subst_domain_def)

```

```

{ fix X F
  assume "Inequality X F  $\in \text{set } S'$ "
  hence *: "ineq_model  $\mathcal{I}$  X F"
    using strand_sem_c_imp_ineq_model[OF *(3)]
    by (auto simp del: subst_range.simps)
  hence "ineq_model ? $\mathcal{I}$  X F"
  proof -
    { fix  $\delta$ 
      assume 1: "subst_domain  $\delta$  = set X" "ground (subst_range  $\delta$ )"
        and 2: "list_ex ( $\lambda f. \text{fst } f \cdot \delta \circ_s \mathcal{I} \neq \text{snd } f \cdot \delta \circ_s \mathcal{I}$ ) F"
      have "list_ex ( $\lambda f. \text{fst } f \cdot \delta \circ_s \text{rm\_vars } B \mathcal{I} \neq \text{snd } f \cdot \delta \circ_s \text{rm\_vars } B \mathcal{I}$ ) F" using 2
      proof (induction F)
        case (Cons g G)
        obtain t t' where g: "g = (t, t')" by (metis surj_pair)

```

```

      thus ?case
        using Cons Unifier_ground_rm_vars[OF grI(1), of "t · δ" B "t' · δ"]
        by auto
    qed simp
  } thus ?thesis using * unfolding ineq_model_def by simp
qed
} moreover have "subst_domain I = UNIV" using assms(1) by metis
hence "subst_domain ?I = A" using rm_vars_dom[of B I] B_def by blast
ultimately obtain Iτ where
  "interpretationsubst Iτ" "Iτ ⊨c ⟨S', ϑ'⟩" "wtsubst Iτ" "wftrms (subst_range Iτ)"
  using wt_sat_if_simple[OF *(1) *(1,2,5,4) _ grI(2) _ *(3)] A_def
  by (auto simp del: subst_range.simps)
thus ?thesis using that LI_soundness[OF assms(3) *(2)] by metis
qed

```

Contra-positive version: if a type-flaw resistant constraint does not have a well-typed model then it is unsatisfiable

```

corollary secure_if_wt_secure:
  assumes "¬(∃ Iτ. interpretationsubst Iτ ∧ (Iτ ⊨c ⟨S, ϑ⟩) ∧ wtsubst Iτ)"
  and      "wfconstr S ϑ" "wtsubst ϑ" "tfrst S"
  and      "wftrms (trmsst S)" "wftrms (subst_range ϑ)"
  shows "¬(∃ I. interpretationsubst I ∧ (I ⊨c ⟨S, ϑ⟩))"
using wt_attack_if_tfr_attack[OF _ _ assms(2,3,4,5,6)] assms(1) by metis

end

```

### 3.4.2 Lifting the Composition-Only Typing Result to the Full Intruder Model

```

context typed_model
begin

```

#### Analysis Invariance

```

definition (in typed_model) Ana_invar_subst where
  "Ana_invar_subst M ≡
    (∀ f T K M δ. Fun f T ∈ (subtermsset M) ⟶
      Ana (Fun f T) = (K, M) ⟶ Ana (Fun f T · δ) = (K ·list δ, M ·list δ))"

```

```

lemma (in typed_model) Ana_invar_subst_subset:
  assumes "Ana_invar_subst M" "N ⊆ M"
  shows "Ana_invar_subst N"
using assms unfolding Ana_invar_subst_def by blast

```

```

lemma (in typed_model) Ana_invar_substD:
  assumes "Ana_invar_subst M"
  and "Fun f T ∈ subtermsset M" "Ana (Fun f T) = (K, M)"
  shows "Ana (Fun f T · I) = (K ·list I, M ·list I)"
using assms Ana_invar_subst_def by blast

```

end

#### Preliminary Definitions

Strands extended with "decomposition steps"

```

datatype (funestp: 'a, varsestp: 'b) extstrand_step =
  Step "'a,'b) strand_step"
| Decomp "'a,'b) term"

```

```

context typed_model
begin

```

```

context
begin

```

```

private fun trms_estp where
  "trms_estp (Step x) = trms_stp x"
  | "trms_estp (Decomp t) = {t}"

private abbreviation trms_est where "trms_est S  $\equiv$   $\bigcup$  (trms_estp ' set S)"

private type_synonym ('a,'b) extstrand = "('a,'b) extstrand_step list"
private type_synonym ('a,'b) extstrands = "('a,'b) extstrand set"

private definition decomp:: "('fun,'var) term  $\Rightarrow$  ('fun,'var) strand" where
  "decomp t  $\equiv$  (case (Ana t) of (K,T)  $\Rightarrow$  send<t>_st#map Send K@map Receive T)"

private fun to_st where
  "to_st [] = []"
  | "to_st (Step x#S) = x#(to_st S)"
  | "to_st (Decomp t#S) = (decomp t)@(to_st S)"

private fun to_est where
  "to_est [] = []"
  | "to_est (x#S) = Step x#to_est S"

private abbreviation "ik_est A  $\equiv$  ik_st (to_st A)"
private abbreviation "wf_est V A  $\equiv$  wf_st V (to_st A)"
private abbreviation "assignment_rhs_est A  $\equiv$  assignment_rhs_st (to_st A)"
private abbreviation "vars_est A  $\equiv$  vars_st (to_st A)"
private abbreviation "wfrestrictedvars_est A  $\equiv$  wfrestrictedvars_st (to_st A)"
private abbreviation "bvars_est A  $\equiv$  bvars_st (to_st A)"
private abbreviation "fv_est A  $\equiv$  fv_st (to_st A)"
private abbreviation "funs_est A  $\equiv$  funs_st (to_st A)"

private definition wf_sts':: "('fun,'var) strands  $\Rightarrow$  ('fun,'var) extstrand  $\Rightarrow$  bool" where
  "wf_sts' S A  $\equiv$  ( $\forall S \in \mathcal{S}. wf_st (wfrestrictedvars_est A) (dual_st S)$ )  $\wedge$ 
    ( $\forall S \in \mathcal{S}. \forall S' \in \mathcal{S}. fv_st S \cap bvars_st S' = \{\}$ )  $\wedge$ 
    ( $\forall S \in \mathcal{S}. fv_st S \cap bvars_est A = \{\}$ )  $\wedge$ 
    ( $\forall S \in \mathcal{S}. fv_st (to_st A) \cap bvars_st S = \{\}$ )"

private definition wf_sts:: "('fun,'var) strands  $\Rightarrow$  bool" where
  "wf_sts S  $\equiv$  ( $\forall S \in \mathcal{S}. wf_st \{\}$  (dual_st S))  $\wedge$  ( $\forall S \in \mathcal{S}. \forall S' \in \mathcal{S}. fv_st S \cap bvars_st S' = \{\}$ )"

private inductive well_analyzed:: "('fun,'var) extstrand  $\Rightarrow$  bool" where
  Nil[simp]: "well_analyzed []"
  | Step: "well_analyzed A  $\Rightarrow$  well_analyzed (A@[Step x])"
  | Decomp: "[well_analyzed A; t  $\in$  subterms_set (ik_est A  $\cup$  assignment_rhs_est A) - (Var ' V)]"
     $\Rightarrow$  well_analyzed (A@[Decomp t])"

private fun subst_apply_extstrandstep (infix ".estp" 51) where
  "subst_apply_extstrandstep (Step x)  $\vartheta$  = Step (x .stp  $\vartheta$ )"
  | "subst_apply_extstrandstep (Decomp t)  $\vartheta$  = Decomp (t .  $\vartheta$ )"

private lemma subst_apply_extstrandstep'_simps[simp]:
  "(Step (send<t>_st)) .estp  $\vartheta$  = Step (send<t .  $\vartheta\vartheta$  = Step (receive<t .  $\vartheta\doteq$  t'>_st)) .estp  $\vartheta$  = Step (<a: (t .  $\vartheta$ )  $\doteq$  (t' .  $\vartheta$ )>_st)"
  "(Step ( $\forall X(\forall \neq: F)$ _st)) .estp  $\vartheta$  = Step ( $\forall X(\forall \neq: (F \cdot_{pairs} rm\_vars (set X) \vartheta)$ _st)"
by simp_all

private lemma vars_estp_subst_apply_simps[simp]:
  "vars_estp ((Step (send<t>_st)) .estp  $\vartheta$ ) = fv (t .  $\vartheta$ )"
  "vars_estp ((Step (receive<t>_st)) .estp  $\vartheta$ ) = fv (t .  $\vartheta$ )"
  "vars_estp ((Step (<a: t  $\doteq$  t'>_st)) .estp  $\vartheta$ ) = fv (t .  $\vartheta$ )  $\cup$  fv (t' .  $\vartheta$ )"
  "vars_estp ((Step ( $\forall X(\forall \neq: F)$ _st)) .estp  $\vartheta$ ) = set X  $\cup$  fv_pairs (F .pairs rm_vars (set X)  $\vartheta$ )"
by auto

```

```

private definition subst_apply_extstrand (infix ".est" 51) where "S .est  $\vartheta \equiv \text{map } (\lambda x. x .estp \vartheta) S"$ 

private abbreviation updatest:: "('fun,'var) strands  $\Rightarrow$  ('fun,'var) strand  $\Rightarrow$  ('fun,'var) strands"
where
  "updatest S S  $\equiv$  (case S of Nil  $\Rightarrow$  S - {S} | Cons _ S'  $\Rightarrow$  insert S' (S - {S}))"

private inductive_set decompest::
  "('fun,'var) terms  $\Rightarrow$  ('fun,'var) terms  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  ('fun,'var) extstrands"

for  $\mathcal{M}$  and  $\mathcal{N}$  and  $\mathcal{I}$  where
  Nil: "[ ]  $\in$  decompest  $\mathcal{M}$   $\mathcal{N}$   $\mathcal{I}$ "
  | Decompose: "[D  $\in$  decompest  $\mathcal{M}$   $\mathcal{N}$   $\mathcal{I}$ ; Fun f T  $\in$  subtermsset ( $\mathcal{M} \cup \mathcal{N}$ );
    Ana (Fun f T) = (K,M); M  $\neq$  [ ];
    ( $\mathcal{M} \cup \text{ik}_{est}$  D) .set  $\mathcal{I} \vdash_c$  Fun f T .  $\mathcal{I}$ ;
     $\bigwedge k. k \in \text{set } K \Rightarrow (\mathcal{M} \cup \text{ik}_{est}$  D) .set  $\mathcal{I} \vdash_c k . \mathcal{I}$ ]
     $\Rightarrow$  D@[Decomp (Fun f T)]  $\in$  decompest  $\mathcal{M}$   $\mathcal{N}$   $\mathcal{I}$ "

private fun decomp_rmest:: "('fun,'var) extstrand  $\Rightarrow$  ('fun,'var) extstrand" where
  "decomp_rmest [ ] = [ ]"
  | "decomp_rmest (Decomp t#S) = decomp_rmest S"
  | "decomp_rmest (Step x#S) = Step x#(decomp_rmest S)"

private inductive semest-d:: "('fun,'var) terms  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  ('fun,'var) extstrand  $\Rightarrow$  bool"
where
  Nil[simp]: "semest-d M0  $\mathcal{I}$  [ ]"
  | Send: "semest-d M0  $\mathcal{I}$  S  $\Rightarrow$  (ikest S  $\cup$  M0) .set  $\mathcal{I} \vdash t . \mathcal{I} \Rightarrow \text{sem}_{est-d}$  M0  $\mathcal{I}$  (S@[Step (send<t>st)])"
  | Receive: "semest-d M0  $\mathcal{I}$  S  $\Rightarrow$  semest-d M0  $\mathcal{I}$  (S@[Step (receive<t>st)])"
  | Equality: "semest-d M0  $\mathcal{I}$  S  $\Rightarrow$  t .  $\mathcal{I} = t' . \mathcal{I} \Rightarrow \text{sem}_{est-d}$  M0  $\mathcal{I}$  (S@[Step (<a: t  $\doteq$  t'>st)])"
  | Inequality: "semest-d M0  $\mathcal{I}$  S
     $\Rightarrow$  ineq_model  $\mathcal{I}$  X F
     $\Rightarrow$  semest-d M0  $\mathcal{I}$  (S@[Step (< $\forall X \langle \forall \neq: F \rangle_{st}$ >)])"
  | Decompose: "semest-d M0  $\mathcal{I}$  S  $\Rightarrow$  (ikest S  $\cup$  M0) .set  $\mathcal{I} \vdash t . \mathcal{I} \Rightarrow \text{Ana } t = (K, M)$ 
     $\Rightarrow$  ( $\bigwedge k. k \in \text{set } K \Rightarrow (\text{ik}_{est}$  S  $\cup$  M0) .set  $\mathcal{I} \vdash k . \mathcal{I} \Rightarrow \text{sem}_{est-d}$  M0  $\mathcal{I}$  (S@[Decomp t]))"

private inductive semest-c:: "('fun,'var) terms  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  ('fun,'var) extstrand  $\Rightarrow$  bool"
where
  Nil[simp]: "semest-c M0  $\mathcal{I}$  [ ]"
  | Send: "semest-c M0  $\mathcal{I}$  S  $\Rightarrow$  (ikest S  $\cup$  M0) .set  $\mathcal{I} \vdash_c t . \mathcal{I} \Rightarrow \text{sem}_{est-c}$  M0  $\mathcal{I}$  (S@[Step (send<t>st)])"
  | Receive: "semest-c M0  $\mathcal{I}$  S  $\Rightarrow$  semest-c M0  $\mathcal{I}$  (S@[Step (receive<t>st)])"
  | Equality: "semest-c M0  $\mathcal{I}$  S  $\Rightarrow$  t .  $\mathcal{I} = t' . \mathcal{I} \Rightarrow \text{sem}_{est-c}$  M0  $\mathcal{I}$  (S@[Step (<a: t  $\doteq$  t'>st)])"
  | Inequality: "semest-c M0  $\mathcal{I}$  S
     $\Rightarrow$  ineq_model  $\mathcal{I}$  X F
     $\Rightarrow$  semest-c M0  $\mathcal{I}$  (S@[Step (< $\forall X \langle \forall \neq: F \rangle_{st}$ >)])"
  | Decompose: "semest-c M0  $\mathcal{I}$  S  $\Rightarrow$  (ikest S  $\cup$  M0) .set  $\mathcal{I} \vdash_c t . \mathcal{I} \Rightarrow \text{Ana } t = (K, M)$ 
     $\Rightarrow$  ( $\bigwedge k. k \in \text{set } K \Rightarrow (\text{ik}_{est}$  S  $\cup$  M0) .set  $\mathcal{I} \vdash_c k . \mathcal{I} \Rightarrow \text{sem}_{est-c}$  M0  $\mathcal{I}$  (S@[Decomp t]))"

```

### Preliminary Lemmata

```

private lemma wfsts-wfsts':
  "wfsts S = wfsts' S [ ]"
by (simp add: wfsts-def wfsts'-def)

private lemma decomp_ik:
  assumes "Ana t = (K,M)"
  shows "ikst (decomp t) = set M"
using ik_rcv_map[of _ M] ik_rcv_map'[of _ M]
by (auto simp add: decomp_def inv_def assms)

private lemma decomp_assignment_rhs_empty:
  assumes "Ana t = (K,M)"
  shows "assignment_rhsst (decomp t) = {}"
by (auto simp add: decomp_def inv_def assms)

```

```

private lemma decomp_tfrstp:
  "list_all tfrstp (decomp t)"
by (auto simp add: decomp_def list_all_def)

private lemma trmsest_ikI:
  "t ∈ ikest A ⇒ t ∈ subtermsset (trmsest A)"
proof (induction A rule: to_st.induct)
  case (2 x S) thus ?case by (cases x) auto
next
  case (3 t' A)
  obtain K M where Ana: "Ana t' = (K,M)" by (metis surj_pair)
  show ?case using 3 decomp_ik[OF Ana] Ana_subterm[OF Ana] by auto
qed simp

private lemma trmsest_ik_assignment_rhsI:
  "t ∈ ikest A ∪ assignment_rhsest A ⇒ t ∈ subtermsset (trmsest A)"
proof (induction A rule: to_st.induct)
  case (2 x S) thus ?case
  proof (cases x)
    case (Equality ac t t') thus ?thesis using 2 by (cases ac) auto
  qed auto
next
  case (3 t' A)
  obtain K M where Ana: "Ana t' = (K,M)" by (metis surj_pair)
  show ?case
  using 3 decomp_ik[OF Ana] decomp_assignment_rhs_empty[OF Ana] Ana_subterm[OF Ana]
  by auto
qed simp

private lemma trmsest_ik_subtermsI:
  assumes "t ∈ subtermsset (ikest A)"
  shows "t ∈ subtermsset (trmsest A)"
proof -
  obtain t' where "t' ∈ ikest A" "t ⊆ t'" using trmsest_ikI assms by auto
  thus ?thesis by (meson contra_subsetD in_subterms_subset_Union trmsest_ikI)
qed

private lemma trmsestD:
  assumes "t ∈ trmsest A"
  shows "t ∈ trmsst (to_st A)"
using assms
proof (induction A)
  case (Cons a A)
  obtain K M where Ana: "Ana t = (K,M)" by (metis surj_pair)
  hence "t ∈ trmsst (decomp t)" unfolding decomp_def by force
  thus ?case using Cons.IH Cons.prem by (cases a) auto
qed simp

private lemma subst_apply_extstrand_nil[simp]:
  "[] ·est ∅ = []"
by (simp add: subst_apply_extstrand_def)

private lemma subst_apply_extstrand_singleton[simp]:
  "[Step (receive⟨t⟩st)] ·est ∅ = [Step (Receive (t · ∅))]"
  "[Step (send⟨t⟩st)] ·est ∅ = [Step (Send (t · ∅))]"
  "[Step (⟨a: t ≐ t'⟩st)] ·est ∅ = [Step (Equality a (t · ∅) (t' · ∅))]"
  "[Decomp t] ·est ∅ = [Decomp (t · ∅)]"
unfolding subst_apply_extstrand_def by auto

private lemma extstrand_subst_hom:
  "(S@S') ·est ∅ = (S ·est ∅)@(S' ·est ∅)" "(x#S) ·est ∅ = (x ·estp ∅)#(S ·est ∅)"
unfolding subst_apply_extstrand_def by auto

```



```

private lemma decomp_vars:
  "wfrestrictedvarsst (decomp t) = fv t" "varsst (decomp t) = fv t" "bvarsst (decomp t) = {}"
  "fvst (decomp t) = fv t"
proof -
  obtain K M where Ana: "Ana t = (K,M)" by (metis surj_pair)
  hence "decomp t = send⟨t⟩st#map Send K@map Receive M"
    unfolding decomp_def by simp
  moreover have "⋃ (set (map fv K)) = fvset (set K)" "⋃ (set (map fv M)) = fvset (set M)" by auto
  moreover have "fvset (set K) ⊆ fv t" "fvset (set M) ⊆ fv t"
    using Ana_subterm[OF Ana(1)] Ana_keys_fv[OF Ana(1)]
    by (simp_all add: UN_least psubsetD subtermeq_vars_subset)
  ultimately show
    "wfrestrictedvarsst (decomp t) = fv t" "varsst (decomp t) = fv t" "bvarsst (decomp t) = {}"
    "fvst (decomp t) = fv t"
  by auto
qed

private lemma bvarsest_cons: "bvarsest (x#X) = bvarsest [x] ∪ bvarsest X"
by (cases x) auto

private lemma bvarsest_append: "bvarsest (A@B) = bvarsest A ∪ bvarsest B"
proof (induction A)
  case (Cons x A) thus ?case using bvarsest_cons[of x "A@B"] bvarsest_cons[of x A] by force
qed simp

private lemma fvest_cons: "fvest (x#X) = fvest [x] ∪ fvest X"
by (cases x) auto

private lemma fvest_append: "fvest (A@B) = fvest A ∪ fvest B"
proof (induction A)
  case (Cons x A) thus ?case using fvest_cons[of x "A@B"] fvest_cons[of x A] by auto
qed simp

private lemma bvarsdecomp: "bvarsest (A@[Decomp t]) = bvarsest A" "bvarsest (Decomp t#A) = bvarsest A"
using bvarsest_append decomp_vars(3) by fastforce+

private lemma bvarsdecomp_rm: "bvarsest (decomp_rmest A) = bvarsest A"
using bvarsdecomp by (induct A rule: decomp_rmest.induct) simp_all+

private lemma fvdecomp_rm: "fvest (decomp_rmest A) ⊆ fvest A"
by (induct A rule: decomp_rmest.induct) auto

private lemma ik_assignment_rhs_decomp_fv:
  assumes "t ∈ subtermsset (ikest A ∪ assignment_rhsest A)"
  shows "fvest (A@[Decomp t]) = fvest A"
proof -
  have "fvest (A@[Decomp t]) = fvest A ∪ fv t" using fvest_append decomp_vars by simp
  moreover have "fvset (ikest A ∪ assignment_rhsest A) ⊆ fvest A" by force
  moreover have "fv t ⊆ fvset (ikest A ∪ assignment_rhsest A)"
    using fv_subset_subterms[OF assms(1)] by simp
  ultimately show ?thesis by blast
qed

private lemma wfrestrictedvarsest_decomp_rmest_subset:
  "wfrestrictedvarsest (decomp_rmest A) ⊆ wfrestrictedvarsest A"
by (induct A rule: decomp_rmest.induct) auto+

private lemma wfrestrictedvarsest_eq_wfrestrictedvarsst:
  "wfrestrictedvarsest A = wfrestrictedvarsst (tost A)"
by simp

private lemma decomp_set_unfold:

```

```

  assumes "Ana t = (K, M)"
  shows "set (decomp t) = {send⟨t⟩st} ∪ (Send ' set K) ∪ (Receive ' set M)"
  using assms unfolding decomp_def by auto

private lemma ikest_finite: "finite (ikest A)"
by (rule finite_ikst)

private lemma assignment_rhsest_finite: "finite (assignment_rhsest A)"
by (rule finite_assignment_rhsst)

private lemma to_est_append: "to_est (A@B) = to_est A@to_est B"
by (induct A rule: to_est.induct) auto

private lemma to_st_to_est_inv: "to_st (to_est A) = A"
by (induct A rule: to_est.induct) auto

private lemma to_st_append: "to_st (A@B) = (to_st A)@(to_st B)"
by (induct A rule: to_st.induct) auto

private lemma to_st_cons: "to_st (a#B) = (to_st [a])@(to_st B)"
using to_st_append[of "[a]" B] by simp

private lemma wfrestrictedvarsest_split:
  "wfrestrictedvarsest (x#S) = wfrestrictedvarsest [x] ∪ wfrestrictedvarsest S"
  "wfrestrictedvarsest (S@S') = wfrestrictedvarsest S ∪ wfrestrictedvarsest S'"
using to_st_cons[of x S] to_st_append[of S S'] by auto

private lemma ikest_append: "ikest (A@B) = ikest A ∪ ikest B"
by (metis ik_append to_st_append)

private lemma assignment_rhsest_append:
  "assignment_rhsest (A@B) = assignment_rhsest A ∪ assignment_rhsest B"
by (metis assignment_rhs_append to_st_append)

private lemma ikest_cons: "ikest (a#A) = ikest [a] ∪ ikest A"
by (metis ik_append to_st_cons)

private lemma ikest_append_subst:
  "ikest (A@B ·est ∅) = ikest (A ·est ∅) ∪ ikest (B ·est ∅)"
  "ikest (A@B) ·set ∅ = (ikest A ·set ∅) ∪ (ikest B ·set ∅)"
by (metis ikest_append extstrand_subst_hom(1), simp add: image_Un to_st_append)

private lemma assignment_rhsest_append_subst:
  "assignment_rhsest (A@B ·est ∅) = assignment_rhsest (A ·est ∅) ∪ assignment_rhsest (B ·est ∅)"
  "assignment_rhsest (A@B) ·set ∅ = (assignment_rhsest A ·set ∅) ∪ (assignment_rhsest B ·set ∅)"
by (metis assignment_rhsest_append extstrand_subst_hom(1), use assignment_rhsest_append in blast)

private lemma ikest_cons_subst:
  "ikest (a#A ·est ∅) = ikest ([a ·estp ∅]) ∪ ikest (A ·est ∅)"
  "ikest (a#A) ·set ∅ = (ikest [a] ·set ∅) ∪ (ikest A ·set ∅)"
by (metis ikest_cons extstrand_subst_hom(2), metis image_Un ikest_cons)

private lemma decomp_rmest_append: "decomp_rmest (S@S') = (decomp_rmest S)@(decomp_rmest S')"
by (induct S rule: decomp_rmest.induct) auto

private lemma decomp_rmest_single[simp]:
  "decomp_rmest [Step (send⟨t⟩st)] = [Step (send⟨t⟩st)]"
  "decomp_rmest [Step (receive⟨t⟩st)] = [Step (receive⟨t⟩st)]"
  "decomp_rmest [Decomp t] = []"
by auto

private lemma decomp_rmest_ik_subset: "ikest (decomp_rmest S) ⊆ ikest S"
proof (induction S rule: decomp_rmest.induct)

```

```

case (3 x S) thus ?case by (cases x) auto
qed auto

private lemma decompest_ik_subset: "D ∈ decompest M N  $\mathcal{I}$  ⇒ ikest D ⊆ subtermsset (M ∪ N)"
proof (induction D rule: decompest.induct)
  case (Decomp D f T K M')
  have "ikst (decomp (Fun f T)) ⊆ subterms (Fun f T)"
    "ikst (decomp (Fun f T)) = ikest [Decomp (Fun f T)]"
    using decomp_ik[OF Decomp.hyps(3)] Ana_subterm[OF Decomp.hyps(3)]
    by auto
  hence "ikst (tost [Decomp (Fun f T)]) ⊆ subtermsset (M ∪ N)"
    using in_subterms_subset_Union[OF Decomp.hyps(2)]
    by blast
  thus ?case using ikest_append[of D "[Decomp (Fun f T)]"] using Decomp.IH by auto
qed simp

private lemma decompest_decomp_rmest_empty: "D ∈ decompest M N  $\mathcal{I}$  ⇒ decomp_rmest D = []"
by (induct D rule: decompest.induct) (auto simp add: decomp_rmest_append)

private lemma decompest_append:
  assumes "A ∈ decompest S N  $\mathcal{I}$ " "B ∈ decompest S N  $\mathcal{I}$ "
  shows "A@B ∈ decompest S N  $\mathcal{I}$ "
using assms(2)
proof (induction B rule: decompest.induct)
  case Nil show ?case using assms(1) by simp
next
  case (Decomp B f X K T)
  hence "S ∪ ikest B ·set  $\mathcal{I}$  ⊆ S ∪ ikest (A@B) ·set  $\mathcal{I}$ " using ikest_append by auto
  thus ?case
    using decompest.Decomp[OF Decomp.IH(1) Decomp.hyps(2,3,4)]
    ideduct_synth_mono[OF Decomp.hyps(5)]
    ideduct_synth_mono[OF Decomp.hyps(6)]
    by auto
qed

private lemma decompest_subterms:
  assumes "A' ∈ decompest M N  $\mathcal{I}$ "
  shows "subtermsset (ikest A') ⊆ subtermsset (M ∪ N)"
using assms
proof (induction A' rule: decompest.induct)
  case (Decomp D f X K T)
  hence "Fun f X ∈ subtermsset (M ∪ N)" by auto
  hence "subtermsset (set X) ⊆ subtermsset (M ∪ N)"
    using in_subterms_subset_Union[of "Fun f X" "M ∪ N"] params_subterms_Union[of X f]
    by blast
  moreover have "ikst (tost [Decomp (Fun f X)]) = set T" using Decomp.hyps(3) decomp_ik by simp
  hence "subtermsset (ikst (tost [Decomp (Fun f X)])) ⊆ subtermsset (set X)"
    using Ana_fun_subterm[OF Decomp.hyps(3)] by auto
  ultimately show ?case
    using ikest_append[of D "[Decomp (Fun f X)]"] Decomp.IH
    by auto
qed simp

private lemma decompest_assignment_rhs_empty:
  assumes "A' ∈ decompest M N  $\mathcal{I}$ "
  shows "assignment_rhsest A' = {}"
using assms
by (induction A' rule: decompest.induct)
  (simp_all add: decomp_assignment_rhs_empty assignment_rhsest_append)

private lemma decompest_finite_ik_append:
  assumes "finite M" "M ⊆ decompest A N  $\mathcal{I}$ "
  shows "∃ D ∈ decompest A N  $\mathcal{I}$ . ikest D = (⋃ m ∈ M. ikest m)"

```

```

using assms
proof (induction M rule: finite_induct)
  case empty
    moreover have "[ ] ∈ decompest A N I" "ikst (tost [ ]) = {}" using decompest.Nil by auto
    ultimately show ?case by blast
  next
    case (insert m M)
    then obtain D where "D ∈ decompest A N I" "ikest D = (⋃m∈M. ikst (tost m))" by moura
    moreover have "m ∈ decompest A N I" using insert.prem1 by blast
    ultimately show ?case using decompest.append[of D A N I m] ikest.append[of D m] by blast
qed

private lemma decomp_snd_exists[simp]: "∃D. decomp t = send⟨t⟩st#D"
by (metis (mono_tags, lifting) decomp_def prod.case surj_pair)

private lemma decomp_nonnil[simp]: "decomp t ≠ [ ]"
using decomp_snd_exists[of t] by fastforce

private lemma to_st_nil_inv[dest]: "tost A = [ ] ⇒ A = [ ]"
by (induct A rule: tost.induct) auto

private lemma well_analyzedD:
  assumes "well_analyzed A" "Decomp t ∈ set A"
  shows "∃f T. t = Fun f T"
using assms
proof (induction A rule: well_analyzed.induct)
  case (Decomp A t')
  hence "∃f T. t' = Fun f T" by (cases t') auto
  moreover have "Decomp t ∈ set A ∨ t = t'" using Decomp by auto
  ultimately show ?case using Decomp.IH by auto
qed auto

private lemma well_analyzed_inv:
  assumes "well_analyzed (A@[Decomp t])"
  shows "t ∈ subtermsset (ikest A ∪ assignment_rhsest A) - (Var ' V)"
using assms well_analyzed.cases[of "A@[Decomp t]"] by fastforce

private lemma well_analyzed_split_left_single: "well_analyzed (A@[a]) ⇒ well_analyzed A"
by (induction "A@[a]" rule: well_analyzed.induct) auto

private lemma well_analyzed_split_left: "well_analyzed (A@B) ⇒ well_analyzed A"
proof (induction B rule: List.rev_induct)
  case (snoc b B) thus ?case using well_analyzed_split_left_single[of "A@B" b] by simp
qed simp

private lemma well_analyzed_append:
  assumes "well_analyzed A" "well_analyzed B"
  shows "well_analyzed (A@B)"
using assms(2,1)
proof (induction B rule: well_analyzed.induct)
  case (Step B x) show ?case using well_analyzed.Step[OF Step.IH[OF Step.prem1]] by simp
next
  case (Decomp B t) thus ?case
    using well_analyzed.Decomp[OF Decomp.IH[OF Decomp.prem1]] ikest.append assignment_rhsest.append
    by auto
qed simp_all

private lemma well_analyzed_singleton:
  "well_analyzed [Step (send⟨t⟩st)]" "well_analyzed [Step (receive⟨t⟩st)]"
  "well_analyzed [Step (⟨a: t ≐ t'⟩st)]" "well_analyzed [Step (∀X(∀≠: F)⟨t⟩st)]"
  "¬well_analyzed [Decomp t]"
proof -
  show "well_analyzed [Step (send⟨t⟩st)]" "well_analyzed [Step (receive⟨t⟩st)]"

```

```

"well_analyzed [Step (<a: t ÷ t'>_st)]" "well_analyzed [Step (<X<V≠: F>_st)]"
using well_analyzed.Step[OF well_analyzed.Nil]
by simp_all

show "¬well_analyzed [Decomp t]" using well_analyzed.cases[of "[Decomp t]"] by auto
qed

private lemma well_analyzed_decomp_rm_est_fv: "well_analyzed A ⇒ fv_est (decomp_rm_est A) = fv_est A"
proof
  assume "well_analyzed A" thus "fv_est A ⊆ fv_est (decomp_rm_est A)"
  proof (induction A rule: well_analyzed.induct)
    case Decompose thus ?case using ik_assignment_rhs_decomp_fv decomp_rm_est_append by auto
  next
    case (Step A x)
    have "fv_est (A@[Step x]) = fv_est A ∪ fv_stp x"
      "fv_est (decomp_rm_est (A@[Step x])) = fv_est (decomp_rm_est A) ∪ fv_stp x"
      using fv_est_append decomp_rm_est_append by auto
    thus ?case using Step by auto
  qed simp
qed (rule fv_decomp_rm)

private lemma sem_est_d_split_left: assumes "sem_est_d M₀ I (A@A′)" shows "sem_est_d M₀ I A"
using assms sem_est_d.cases by (induction A′ rule: List.rev_induct) fastforce+

private lemma sem_est_d_eq_sem_st: "sem_est_d M₀ I A = ⟦M₀; to_st A⟧_d' I"
proof
  show "⟦M₀; to_st A⟧_d' I ⇒ sem_est_d M₀ I A"
  proof (induction A arbitrary: M₀ rule: List.rev_induct)
    case Nil show ?case using to_st_nil_inv by simp
  next
    case (snoc a A)
    hence IH: "sem_est_d M₀ I A" and *: "⟦ik_est A ∪ M₀; to_st [a]⟧_d' I"
      using to_st_append by (auto simp add: sup commute)
    thus ?case using snoc
    proof (cases a)
      case (Step b) thus ?thesis
      proof (cases b)
        case (Send t) thus ?thesis using sem_est_d.Send[OF IH] * Step by auto
      next
        case (Receive t) thus ?thesis using sem_est_d.Receive[OF IH] Step by auto
      next
        case (Equality a t t′) thus ?thesis using sem_est_d.Equality[OF IH] * Step by auto
      next
        case (Inequality X F) thus ?thesis using sem_est_d.Inequality[OF IH] * Step by auto
      qed
    next
      case (Decomp t)
      obtain K M where Ana: "Ana t = (K,M)" by maura
      have "to_st [a] = decomp t" using Decompose by auto
      hence "to_st [a] = send⟨t⟩_st#map Send K@map Receive M"
        using Ana unfolding decomp_def by auto
      hence **: "ik_est A ∪ M₀ ·_set I ⊢ t · I" and "⟦ik_est A ∪ M₀; map Send K⟧_d' I"
        using * by auto
      hence "⋀k. k ∈ set K ⇒ ik_est A ∪ M₀ ·_set I ⊢ k · I"
        using *
        by (metis (full_types) strand_sem_d.simps(2) strand_sem_eq_defs(2) strand_sem_Send_split(2))
      thus ?thesis using Decompose sem_est_d.Decompose[OF IH ** Ana] by metis
    qed
  qed
qed

show "sem_est_d M₀ I A ⇒ ⟦M₀; to_st A⟧_d' I"
proof (induction rule: sem_est_d.induct)
  case Nil thus ?case by simp

```

```

next
  case (Send  $M_0 \mathcal{I} \mathcal{A} t$ ) thus ?case
    using strand_sem_append'[of  $M_0$  "to_st  $\mathcal{A}$ "  $\mathcal{I}$  "[send< $t$ > $_{st}$ ]]"]
      to_st_append[of  $\mathcal{A}$  "[Step (send< $t$ > $_{st}$ )]]"]
    by (simp add: sup commute)
next
  case (Receive  $M_0 \mathcal{I} \mathcal{A} t$ ) thus ?case
    using strand_sem_append'[of  $M_0$  "to_st  $\mathcal{A}$ "  $\mathcal{I}$  "[receive< $t$ > $_{st}$ ]]"]
      to_st_append[of  $\mathcal{A}$  "[Step (receive< $t$ > $_{st}$ )]]"]
    by (simp add: sup commute)
next
  case (Equality  $M_0 \mathcal{I} \mathcal{A} t t' a$ ) thus ?case
    using strand_sem_append'[of  $M_0$  "to_st  $\mathcal{A}$ "  $\mathcal{I}$  "[< $a: t \doteq t'$ > $_{st}$ ]]"]
      to_st_append[of  $\mathcal{A}$  "[Step (< $a: t \doteq t'$ > $_{st}$ )]]"]
    by (simp add: sup commute)
next
  case (Inequality  $M_0 \mathcal{I} \mathcal{A} X F$ ) thus ?case
    using strand_sem_append'[of  $M_0$  "to_st  $\mathcal{A}$ "  $\mathcal{I}$  "[ $\forall X \langle \forall \neq: F \rangle_{st}$ ]]"]
      to_st_append[of  $\mathcal{A}$  "[Step ( $\forall X \langle \forall \neq: F \rangle_{st}$ )]]"]
    by (simp add: sup commute)
next
  case (Decompose  $M_0 \mathcal{I} \mathcal{A} t K M$ )
  have "[ $M_0 \cup ik_{st} (to\_st \mathcal{A}); decomp t$ ] $_d$ '  $\mathcal{I}$ "
  proof -
    have "[ $M_0 \cup ik_{st} (to\_st \mathcal{A}); [send< $t$ > $_{st}$ ]] $_d$ '  $\mathcal{I}$ "
      using Decompose.hyps(2) by (auto simp add: sup commute)
    moreover have " $\bigwedge k. k \in set K \implies M_0 \cup ik_{st} (to\_st \mathcal{A}) \cdot_{set} \mathcal{I} \vdash k \cdot \mathcal{I}$ "
      using Decompose by (metis sup commute)
    hence " $\bigwedge k. k \in set K \implies [M_0 \cup ik_{st} (to\_st \mathcal{A}); [Send k]]_d$ '  $\mathcal{I}$ " by auto
    hence "[ $M_0 \cup ik_{st} (to\_st \mathcal{A}); map Send K$ ] $_d$ '  $\mathcal{I}$ "
      using strand_sem_Send_map(2)[of  $K$ , of " $M_0 \cup ik_{st} (to\_st \mathcal{A}) \cdot_{set} \mathcal{I}$ "  $\mathcal{I}$ ] strand_sem_eq_defs(2)
      by auto
    moreover have "[ $M_0 \cup ik_{st} (to\_st \mathcal{A}); map Receive M$ ] $_d$ '  $\mathcal{I}$ "
      by (metis strand_sem_Receive_map(2) strand_sem_eq_defs(2))
    ultimately have
      "[ $M_0 \cup ik_{st} (to\_st \mathcal{A}); send< $t$ > $_{st}$ #map Send K@map Receive M$ ] $_d$ '  $\mathcal{I}$ "
      by auto
    thus ?thesis using Decompose.hyps(3) unfolding decomp_def by auto
  qed
  hence "[ $M_0; to\_st \mathcal{A}@decomp t$ ] $_d$ '  $\mathcal{I}$ "
    using strand_sem_append'[of  $M_0$  "to_st  $\mathcal{A}$ "  $\mathcal{I}$  "decomp  $t$ "] Decompose.IH
    by simp
  thus ?case using to_st_append[of  $\mathcal{A}$  "[Decomp  $t$ ]]" by simp
qed
qed

private lemma sem_est_c_eq_sem_st: "sem_est_c  $M_0 \mathcal{I} \mathcal{A} = [M_0; to\_st \mathcal{A}]_c$ '  $\mathcal{I}$ "
proof
  show "[ $M_0; to\_st \mathcal{A}$ ] $_c$ '  $\mathcal{I} \implies sem\_est\_c M_0 \mathcal{I} \mathcal{A}$ "
  proof (induction  $\mathcal{A}$  arbitrary:  $M_0$  rule: List.rev_induct)
    case Nil show ?case using to_st_nil_inv by simp
  next
    case (snoc  $a \mathcal{A}$ )
    hence IH: "sem_est_c  $M_0 \mathcal{I} \mathcal{A}$ " and *: "[ $ik_{est} \mathcal{A} \cup M_0; to\_st [a]$ ] $_c$ '  $\mathcal{I}$ "
      using to_st_append
      by (auto simp add: sup commute)
    thus ?case using snoc
    proof (cases  $a$ )
      case (Step  $b$ ) thus ?thesis
      proof (cases  $b$ )
        case (Send  $t$ ) thus ?thesis using sem_est_c.Send[OF IH] * Step by auto
      next
        case (Receive  $t$ ) thus ?thesis using sem_est_c.Receive[OF IH] Step by auto
      end
    end
  end
end$ 
```

```

next
  case (Equality t) thus ?thesis using sem_est_c.Equality[OF IH] * Step by auto
next
  case (Inequality t) thus ?thesis using sem_est_c.Inequality[OF IH] * Step by auto
qed
next
  case (Decomp t)
  obtain K M where Ana: "Ana t = (K,M)" by moura
  have "to_st [a] = decomp t" using Decomp by auto
  hence "to_st [a] = send⟨t⟩st#map Send K@map Receive M"
    using Ana unfolding decomp_def by auto
  hence **: "ikest A ∪ M0 ·set I ⊢c t · I" and "[[ikest A ∪ M0; map Send K]c' I"
    using * by auto
  hence "∧k. k ∈ set K ⇒ ikest A ∪ M0 ·set I ⊢c k · I"
    using * strand_sem_Send_split(1) strand_sem_eq_defs(1)
    by auto
  thus ?thesis using Decomp sem_est_c.Decompose[OF IH ** Ana] by metis
qed
qed

show "sem_est_c M0 I A ⇒ [[M0; to_st A]c' I"
proof (induction rule: sem_est_c.induct)
  case Nil thus ?case by simp
next
  case (Send M0 I A t) thus ?case
    using strand_sem_append'[of M0 "to_st A" I "[send⟨t⟩st]"
      to_st_append[of A "[Step (send⟨t⟩st)]"
    by (simp add: sup commute)
next
  case (Receive M0 I A t) thus ?case
    using strand_sem_append'[of M0 "to_st A" I "[receive⟨t⟩st]"
      to_st_append[of A "[Step (receive⟨t⟩st)]"
    by (simp add: sup commute)
next
  case (Equality M0 I A t t' a) thus ?case
    using strand_sem_append'[of M0 "to_st A" I "[⟨a: t ≐ t'⟩st]"
      to_st_append[of A "[Step (⟨a: t ≐ t'⟩st)]"
    by (simp add: sup commute)
next
  case (Inequality M0 I A X F) thus ?case
    using strand_sem_append'[of M0 "to_st A" I "[∀X⟨∀≠: F⟩st]"
      to_st_append[of A "[Step (∀X⟨∀≠: F⟩st)]"
    by (auto simp add: sup commute)
next
  case (Decompose M0 I A t K M)
  have "[[M0 ∪ ikst (to_st A); decomp t]c' I"
  proof -
    have "[[M0 ∪ ikst (to_st A); [send⟨t⟩st]]c' I"
      using Decompose.hyps(2) by (auto simp add: sup commute)
    moreover have "∧k. k ∈ set K ⇒ M0 ∪ ikst (to_st A) ·set I ⊢c k · I"
      using Decompose by (metis sup commute)
    hence "∧k. k ∈ set K ⇒ [[M0 ∪ ikst (to_st A); [Send k]]c' I" by auto
    hence "[[M0 ∪ ikst (to_st A); map Send K]c' I"
      using strand_sem_Send_map(1)[of K, of "M0 ∪ ikst (to_st A) ·set I" I]
      strand_sem_eq_defs(1)
      by auto
    moreover have "[[M0 ∪ ikst (to_st A); map Receive M]c' I"
      by (metis strand_sem_Receive_map(1) strand_sem_eq_defs(1))
    ultimately have
      "[[M0 ∪ ikst (to_st A); send⟨t⟩st#map Send K@map Receive M]c' I"
      by auto
    thus ?thesis using Decompose.hyps(3) unfolding decomp_def by auto
  qed
qed

```

```

    hence "[M0; to_st A@decomp t]c' I"
    using strand_sem_append'[of M0 "to_st A" I "decomp t"] Decompose.IH
    by simp
    thus ?case using to_st_append[of A "[Decomp t]"] by simp
qed
qed

private lemma sem_est_c_decomp_rm_est_deduct_aux:
  assumes "sem_est_c M0 I A" "t ∈ ikest A ·set I" "t ∉ ikest (decomp_rmest A) ·set I"
  shows "ikest (decomp_rmest A) ∪ M0 ·set I ⊢ t"
using assms
proof (induction M0 I A arbitrary: t rule: sem_est_c.induct)
  case (Send M0 I A t') thus ?case using decomp_rm_est_append ik_est_append by auto
next
  case (Receive M0 I A t')
  hence "t ∈ ikest A ·set I" "t ∉ ikest (decomp_rmest A) ·set I"
  using decomp_rm_est_append ik_est_append by auto
  hence IH: "ikest (decomp_rmest A) ∪ M0 ·set I ⊢ t" using Receive.IH by auto
  show ?case using ideduct_mono[OF IH] decomp_rm_est_append ik_est_append by auto
next
  case (Equality M0 I A t') thus ?case using decomp_rm_est_append ik_est_append by auto
next
  case (Inequality M0 I A t') thus ?case using decomp_rm_est_append ik_est_append by auto
next
  case (Decompose M0 I A t' K M t)
  have *: "ikest (decomp_rmest A) ∪ M0 ·set I ⊢ t' · I" using Decompose.hyps(2)
  proof (induction rule: intruder_synth_induct)
    case (AxiomC t'')
    moreover {
      assume "t'' ∈ ikest A ·set I" "t'' ∉ ikest (decomp_rmest A) ·set I"
      hence ?case using Decompose.IH by auto
    }
    ultimately show ?case by force
  qed simp

  { fix k assume "k ∈ set K"
    hence "ikest A ∪ M0 ·set I ⊢c k · I" using Decompose.hyps by auto
    hence "ikest (decomp_rmest A) ∪ M0 ·set I ⊢ k · I"
    proof (induction rule: intruder_synth_induct)
      case (AxiomC t'')
      moreover {
        assume "t'' ∈ ikest A ·set I" "t'' ∉ ikest (decomp_rmest A) ·set I"
        hence ?case using Decompose.IH by auto
      }
      ultimately show ?case by force
    qed simp
  }
}
hence **: "∧k. k ∈ set (K ·list I) ⇒ ikest (decomp_rmest A) ∪ M0 ·set I ⊢ k" by auto

show ?case
proof (cases "t ∈ ikest A ·set I")
  case True thus ?thesis using Decompose.IH Decompose.prem(2) decomp_rm_est_append by auto
next
  case False
  hence "t ∈ ikst (decomp t') ·set I" using Decompose.prem(1) ik_est_append by auto
  hence ***: "t ∈ set (M ·list I)" using Decompose.hyps(3) decomp_ik by auto
  hence "M ≠ []" by auto
  hence ****: "Ana (t' · I) = (K ·list I, M ·list I)" using Ana_subst[OF Decompose.hyps(3)] by auto

  have "ikest (decomp_rmest A) ∪ M0 ·set I ⊢ t" by (rule intruder_deduct.Decompose[OF * **** **])
  thus ?thesis using ideduct_mono decomp_rm_est_append by auto
qed

```



qed simp

```
private lemma sem_est_c_decomp_rm_est_deduct:
  assumes "sem_est_c M0 I A" "ik_est A ∪ M0 ·set I ⊢c t"
  shows "ik_est (decomp_rm_est A) ∪ M0 ·set I ⊢ t"
using assms(2)
proof (induction t rule: intruder_synth_induct)
  case (AxiomC t)
  hence "t ∈ ik_est A ·set I ∨ t ∈ M0 ·set I" by auto
  moreover {
    assume "t ∈ ik_est A ·set I" "t ∈ ik_est (decomp_rm_est A) ·set I"
    hence ?case using ideduct_mono[OF intruder_deduct.Axiom] by auto
  }
  moreover {
    assume "t ∈ ik_est A ·set I" "t ∉ ik_est (decomp_rm_est A) ·set I"
    hence ?case using sem_est_c_decomp_rm_est_deduct_aux[OF assms(1)] by auto
  }
  ultimately show ?case by auto
qed simp
```

```
private lemma sem_est_d_decomp_rm_est_if_sem_est_c: "sem_est_c M0 I A ⟹ sem_est_d M0 I (decomp_rm_est A)"
proof (induction M0 I A rule: sem_est_c.induct)
  case (Send M0 I A t)
  thus ?case using decomp_rm_est_append sem_est_d.Send[OF Send.IH] sem_est_c_decomp_rm_est_deduct by auto
next
  case (Receive t) thus ?case using decomp_rm_est_append sem_est_d.Receive by auto
next
  case (Equality M0 I A t)
  thus ?case
    using decomp_rm_est_append sem_est_d.Equality[OF Equality.IH] sem_est_c_decomp_rm_est_deduct
    by auto
next
  case (Inequality M0 I A t)
  thus ?case
    using decomp_rm_est_append sem_est_d.Inequality[OF Inequality.IH] sem_est_c_decomp_rm_est_deduct
    by auto
next
  case Decompose thus ?case using decomp_rm_est_append by auto
qed auto
```

```
private lemma sem_est_c_decomps_est_append:
  assumes "sem_est_c {} I A" "D ∈ decomp_est (ik_est A) (assignment_rhs_est A) I"
  shows "sem_est_c {} I (A @ D)"
using assms(2,1)
proof (induction D rule: decomp_est.induct)
  case (Decomp D f T K M)
  hence *: "sem_est_c {} I (A @ D)" "ik_est (A @ D) ∪ {} ·set I ⊢c Fun f T · I"
    "∧ k. k ∈ set K ⟹ ik_est (A @ D) ∪ {} ·set I ⊢c k · I"
    using ik_est_append by auto
  show ?case using sem_est_c.Decompose[OF *(1,2) Decomp.hyps(3) *(3)] by simp
qed auto
```

```
private lemma decomp_est_preserves_wf:
  assumes "D ∈ decomp_est (ik_est A) (assignment_rhs_est A) I" "wf_est V A"
  shows "wf_est V (A @ D)"
using assms
proof (induction D rule: decomp_est.induct)
  case (Decomp D f T K M)
  have "wfrestrictedvarsst (decomp (Fun f T)) ⊆ fvset (ik_est A ∪ assignment_rhs_est A)"
    using decomp_vars fv_subset_subterms[OF Decomp.hyps(2)] by fast
  hence "wfrestrictedvarsst (decomp (Fun f T)) ⊆ wfrestrictedvarsest A"
    using ik_st_assignment_rhs_st_wfrestrictedvars_subset[of "to_st A"] by blast
  hence "wfrestrictedvarsst (decomp (Fun f T)) ⊆ wfrestrictedvarsst (to_st (A @ D)) ∪ V"
```

```

using to_st_append[of A D] strand_vars_split(2)[of "to_st A" "to_st D"]
by (metis le_supI1)
thus ?case
  using wf_append_suffix[OF Decomp.IH[OF Decomp.prem], of "decomp (Fun f T)"]
    to_st_append[of "A@D" "[Decomp (Fun f T)]"]
  by auto
qed auto

private lemma decompest_preserves_model_c:
  assumes "D ∈ decompest (ikest A) (assignment_rhsest A) I" "semest-c M0 I A"
  shows "semest-c M0 I (A@D)"
using assms
proof (induction D rule: decompest.induct)
  case (Decomp D f T K M) show ?case
    using semest-c.Decompose[OF Decomp.IH[OF Decomp.prem] _ Decomp.hyps(3)]
      Decomp.hyps(5,6) ideduct_synth_mono ikest_append
    by (metis (mono_tags, lifting) List.append_assoc image_Un sup_ge1)
qed auto

private lemma decompest_exist_aux:
  assumes "D ∈ decompest M N I" "M ∪ ikest D ⊢ t" "¬(M ∪ (ikest D) ⊢c t)"
  obtains D' where
    "D@D' ∈ decompest M N I" "M ∪ ikest (D@D') ⊢ t" "M ∪ ikest D ⊂ M ∪ ikest (D@D')"
proof -
  have "∃D' ∈ decompest M N I. M ∪ ikest D' ⊢ t" using assms(2)
  proof (induction t rule: intruder_deduct_induct)
    case (Compose X f)
    from Compose.IH have "∃D ∈ decompest M N I. ∀x ∈ set X. M ∪ ikest D ⊢c x"
    proof (induction X)
      case (Cons t X)
      then obtain D' D'' where
        D': "D' ∈ decompest M N I" "M ∪ ikest D' ⊢c t" and
        D'': "D'' ∈ decompest M N I" "∀x ∈ set X. M ∪ ikest D'' ⊢c x"
      by maura
      hence "M ∪ ikest (D'@D'') ⊢c t" "∀x ∈ set X. M ∪ ikest (D'@D'') ⊢c x"
      by (auto intro: ideduct_synth_mono simp add: ikest_append)
      thus ?case using decompest_append[OF D'(1) D''(1)] by (metis set_ConsD)
    qed (auto intro: decompest.Nil)
    thus ?case using intruder_synth.ComposeC[OF Compose.hyps(1,2)] by metis
  next
    case (Decompose t K T ti)
    have "∃D ∈ decompest M N I. ∀k ∈ set K. M ∪ ikest D ⊢c k" using Decompose.IH
    proof (induction K)
      case (Cons t X)
      then obtain D' D'' where
        D': "D' ∈ decompest M N I" "M ∪ ikest D' ⊢c t" and
        D'': "D'' ∈ decompest M N I" "∀x ∈ set X. M ∪ ikest D'' ⊢c x"
      using assms(1) by maura
      hence "M ∪ ikest (D'@D'') ⊢c t" "∀x ∈ set X. M ∪ ikest (D'@D'') ⊢c x"
      by (auto intro: ideduct_synth_mono simp add: ikest_append)
      thus ?case using decompest_append[OF D'(1) D''(1)] by auto
    qed auto
    then obtain D' where D': "D' ∈ decompest M N I" "∧k. k ∈ set K ⇒ M ∪ ikest D' ⊢c k" by
    metis
    obtain D'' where D'': "D'' ∈ decompest M N I" "M ∪ ikest D'' ⊢c t" by (metis Decompose.IH(1))
    obtain f X where fX: "t = Fun f X" "ti ∈ set X"
    using Decompose.hyps(2,4) by (cases t) (auto dest: Ana_fun_subterm)

  from decompest_append[OF D'(1) D''(1)] D'(2) D''(2) have *:
    "D'@D'' ∈ decompest M N I" "∧k. k ∈ set K ⇒ M ∪ ikest (D'@D'') ⊢c k"
    "M ∪ ikest (D'@D'') ⊢c t"
  by (auto intro: ideduct_synth_mono simp add: ikest_append)
  hence **: "∧k. k ∈ set K ⇒ M ∪ ikest (D'@D'') .set I ⊢c k . I"

```

```

using ideduct_synth_subst by auto

have "ti ∈ ikst (decomp t)" using Decompose.hyps(2,4) ik_rcv_map unfolding decomp_def by auto
with *(3) fX(1) Decompose.hyps(2) show ?case
proof (induction t rule: intruder_synth_induct)
  case (AxiomC t)
  hence t_in_subterms: "t ∈ subtermsset (M ∪ N)"
    using decompest_ik_subset[OF *(1)] subset_subterms_Union
    by auto
  have "M ∪ ikest (D'@D') .set I ⊢c t . I"
    using ideduct_synth_subst[OF intruder_synth.AxiomC[OF AxiomC.hyps(1)]] by metis
  moreover have "T ≠ []" using decomp_ik[OF ⟨Ana t = (K,T)⟩] ⟨ti ∈ ikst (decomp t)⟩ by auto
  ultimately have "D'@D'@[Decomp (Fun f X)] ∈ decompest M N I"
    using AxiomC decompest.Decomp[OF *(1) _ _ _ **] subset_subterms_Union t_in_subterms
    by (simp add: subset_eq)
  moreover have "decomp t = tost [Decomp (Fun f X)]" using AxiomC.prem(1,2) by auto
  ultimately show ?case
    by (metis AxiomC.prem(3) UnCI intruder_synth.AxiomC ikest_append tost_append)
qed (auto intro!: fX(2) *(1))
qed (fastforce intro: intruder_synth.AxiomC assms(1))
hence "∃D' ∈ decompest M N I. M ∪ ikest (D@D') ⊢c t"
  by (auto intro: ideduct_synth_mono simp add: ikest_append)
thus thesis using that[OF decompest_append[OF assms(1)]] assms ikest_append by maura
qed

private lemma decompest_ik_max_exist:
  assumes "finite A" "finite N"
  shows "∃D ∈ decompest A N I. ∀D' ∈ decompest A N I. ikest D' ⊆ ikest D"
proof -
  let ?IK = "λM. ⋃ D ∈ M. ikest D"
  have "?IK (decompest A N I) ⊆ (⋃ t ∈ A ∪ N. subterms t)" by (auto dest!: decompest_ik_subset)
  hence "finite (?IK (decompest A N I))"
    using subterms_union_finite[OF assms(1)] subterms_union_finite[OF assms(2)] infinite_super
    by auto
  then obtain M where M: "finite M" "M ⊆ decompest A N I" "?IK M = ?IK (decompest A N I)"
    using finite_subset_Union by maura
  show ?thesis using decompest_finite_ik_append[OF M(1,2)] M(3) by auto
qed

private lemma decompest_exist:
  assumes "finite A" "finite N"
  shows "∃D ∈ decompest A N I. ∀t. A ⊢ t → A ∪ ikest D ⊢c t"
proof (rule ccontr)
  assume neg: "¬(∃D ∈ decompest A N I. ∀t. A ⊢ t → A ∪ ikest D ⊢c t)"

  obtain D where D: "D ∈ decompest A N I" "∀D' ∈ decompest A N I. ikest D' ⊆ ikest D"
    using decompest_ik_max_exist[OF assms] by maura
  then obtain t where t: "A ∪ ikest D ⊢ t" "¬(A ∪ ikest D ⊢c t)"
    using neg by (fastforce intro: ideduct_mono)

  obtain D' where D':
    "D@D' ∈ decompest A N I" "A ∪ ikest (D@D') ⊢c t"
    "A ∪ ikest D ⊂ A ∪ ikest (D@D')"
    by (metis decompest_exist_aux t D(1))
  hence "ikest D ⊂ ikest (D@D')" using ikest_append by auto
  moreover have "ikest (D@D') ⊆ ikest D" using D(2) D'(1) by auto
  ultimately show False by simp
qed

private lemma decompest_exist_subst:
  assumes "ikest A .set I ⊢ t . I"
  and "semest-c {} I A" "wfest {} A" "interpretationsubst I"
  and "Ana_invar_subst (ikest A ∪ assignment_rhsest A)"

```

```

and "well_analyzed A"
shows "∃ D ∈ decompest (ikest A) (assignment_rhsest A) I. ikest (A@D) ·set I ⊢c t · I"
proof -
have ik_eq: "ikest (A ·est I) = ikest A ·set I" using assms(5,6)
proof (induction A rule: List.rev_induct)
case (snoc a A)
hence "Ana_invar_subst (ikest A ∪ assignment_rhsest A)"
  using Ana_invar_subst_subset[OF snoc.prem(1)] ikest_append assignment_rhsest_append
  unfolding Ana_invar_subst_def by simp
with snoc have IH:
  "ikest (A@[a] ·est I) = (ikest A ·set I) ∪ ikest ([a] ·est I)"
  "ikest (A@[a]) ·set I = (ikest A ·set I) ∪ (ikest [a] ·set I)"
  using well_analyzed_split_left[OF snoc.prem(2)]
  by (auto simp add: to_st_append ikest_append_subst)

have "ikest [a ·estp I] = ikest [a] ·set I"
proof (cases a)
case (Step b) thus ?thesis by (cases b) auto
next
case (Decomp t)
then obtain f T where t: "t = Fun f T" using well_analyzedD[OF snoc.prem(2)] by force
obtain K M where Ana_t: "Ana (Fun f T) = (K,M)" by (metis surj_pair)
moreover have "Fun f T ∈ subtermsset ((ikest (A@[a]) ∪ assignment_rhsest (A@[a])))"
  using t Decomp snoc.prem(2)
  by (auto dest: well_analyzed_inv simp add: ikest_append assignment_rhsest_append)
hence "Ana (Fun f T · I) = (K ·list I, M ·list I)"
  using Ana_t snoc.prem(1)
  unfolding Ana_invar_subst_def by force
ultimately show ?thesis using Decomp t by (auto simp add: decomp_ik)
qed
thus ?case using IH unfolding subst_apply_extstrand_def by simp
qed simp
moreover have assignment_rhs_eq: "assignment_rhsest (A ·est I) = assignment_rhsest A ·set I"
  using assms(5,6)
proof (induction A rule: List.rev_induct)
case (snoc a A)
hence "Ana_invar_subst (ikest A ∪ assignment_rhsest A)"
  using Ana_invar_subst_subset[OF snoc.prem(1)] ikest_append assignment_rhsest_append
  unfolding Ana_invar_subst_def by simp
hence "assignment_rhsest (A ·est I) = assignment_rhsest A ·set I"
  using snoc.IH well_analyzed_split_left[OF snoc.prem(2)]
  by simp
hence IH:
  "assignment_rhsest (A@[a] ·est I) = (assignment_rhsest A ·set I) ∪ assignment_rhsest ([a] ·est
I)"
  "assignment_rhsest (A@[a]) ·set I = (assignment_rhsest A ·set I) ∪ (assignment_rhsest [a] ·set
I)"
  by (metis assignment_rhsest_append_subst(1), metis assignment_rhsest_append_subst(2))

have "assignment_rhsest [a ·estp I] = assignment_rhsest [a] ·set I"
proof (cases a)
case (Step b) thus ?thesis by (cases b) auto
next
case (Decomp t)
then obtain f T where t: "t = Fun f T" using well_analyzedD[OF snoc.prem(2)] by force
obtain K M where Ana_t: "Ana (Fun f T) = (K,M)" by (metis surj_pair)
moreover have "Fun f T ∈ subtermsset ((ikest (A@[a]) ∪ assignment_rhsest (A@[a])))"
  using t Decomp snoc.prem(2)
  by (auto dest: well_analyzed_inv simp add: ikest_append assignment_rhsest_append)
hence "Ana (Fun f T · I) = (K ·list I, M ·list I)"
  using Ana_t snoc.prem(1) unfolding Ana_invar_subst_def by force
ultimately show ?thesis using Decomp t by (auto simp add: decomp_assignment_rhs_empty)
qed

```

```

    thus ?case using IH unfolding subst_apply_extstrand_def by simp
qed simp
ultimately obtain D where D:
  "D ∈ decompest (ikest A ·set I) (assignment_rhsest A ·set I) Var"
  "(ikest A ·set I) ∪ (ikest D) ⊢c t · I"
using decompest_exist[OF ikest_finite assignment_rhsest_finite, of "A ·est I" "A ·est I"]
  ikest_append assignment_rhsest_append assms(1)
by force

let ?P = "λD D'. ∀t. (ikest A ·set I) ∪ (ikest D) ⊢c t → (ikest A ·set I) ∪ (ikest D' ·set I) ⊢c t"

have "∃D' ∈ decompest (ikest A) (assignment_rhsest A) I. ?P D D'" using D(1)
proof (induction D rule: decompest.induct)
  case Nil
  have "ikest [] = ikest [] ·set I" by auto
  thus ?case by (metis decompest.Nil)
next
  case (Decomp D f T K M)
  obtain D' where D': "D' ∈ decompest (ikest A) (assignment_rhsest A) I" "?P D D'"
  using Decomp.IH by auto
  hence IH: "∧k. k ∈ set K ⇒ (ikest A ·set I) ∪ (ikest D' ·set I) ⊢c k"
    "(ikest A ·set I) ∪ (ikest D' ·set I) ⊢c Fun f T"
  using Decomp.hyps(5,6) by auto

  have D'_ik: "ikest D' ·set I ⊆ subtermsset ((ikest A ∪ assignment_rhsest A) ·set I)"
    "ikest D' ⊆ subtermsset (ikest A ∪ assignment_rhsest A)"
  using decompest_ik_subset[OF D'(1)] by (metis subst_all_mono, metis)

  show ?case using IH(2,1) Decomp.hyps(2,3,4)
  proof (induction "Fun f T" arbitrary: f T K M rule: intruder_synth_induct)
    case (AxiomC f T)
    then obtain s where s: "s ∈ ikest A ∪ ikest D'" "Fun f T = s · I" using AxiomC.prem by blast
    hence fT_s_in: "Fun f T ∈ (subtermsset (ikest A ∪ assignment_rhsest A)) ·set I"
      "s ∈ subtermsset (ikest A ∪ assignment_rhsest A)"
    using AxiomC D'_ik subset_subterms_Union[of "ikest A ∪ assignment_rhsest A"]
      subst_all_mono[OF subset_subterms_Union, of I]
    by (metis (no_types) Un_iff image_eqI subset_Un_eq, metis (no_types) Un_iff subset_Un_eq)
    obtain Ks Ms where Ana_s: "Ana s = (Ks, Ms)" by moura

    have AD'_props: "wfest {} (A@D')" "⌊{ }; tost (A@D')⌋c I"
    using decompest_preserves_model_c[OF D'(1) assms(2)]
      decompest_preserves_wf[OF D'(1) assms(3)]
      semest_c_eq_semst strand_sem_eq_defs(1)
    by auto

    show ?case
    proof (cases s)
      case (Var x)
      — In this case I x (is a subterm of something that) was derived from an "earlier intruder knowledge" because
      A is well-formed and has I as a model. So either the intruder composed Fun f T himself (making Decomp (Fun f T)
      unnecessary) or Fun f T is an instance of something else in the intruder knowledge (in which case the "something" can
      be used in place of Fun f T)
      hence "Var x ∈ ikest (A@D')" "I x = Fun f T" using s ikest_append by auto

      show ?thesis
      proof (cases "∀m ∈ set M. ikest A ∪ ikest D' ·set I ⊢c m")
        case True
        — All terms acquired by decomposing Fun f T are already derivable. Hence there is no need to consider
        decomposition of Fun f T at all.
        have *: "(ikest A ·set I) ∪ ikest (D@Decomp (Fun f T)) = (ikest A ·set I) ∪ ikest D ∪ set
M"
        using decomp_ik[OF (Ana (Fun f T) = (K, M))] ikest_append[of D "[Decomp (Fun f T)]"]
        by auto
      end
    end
  end

```

```

{ fix t' assume "(ikest A ·set I) ∪ ikest D ∪ set M ⊢c t'"
  hence "(ikest A ·set I) ∪ (ikest D' ·set I) ⊢c t'"
  proof (induction t' rule: intruder_synth_induct)
    case (AxiomC t') thus ?case
      proof
        assume "t' ∈ set M"
        moreover have "(ikest A ·set I) ∪ (ikest D' ·set I) = ikest A ∪ ikest D' ·set I" by
auto
          ultimately show ?case using True by auto
        qed (metis D'(2) intruder_synth.AxiomC)
      qed auto
    }
  } thus ?thesis using D'(1) * by metis
next
case False
— Some term acquired by decomposition of Fun f T cannot be derived in ⊢c. Fun f T must therefore be an
instance of something else in the intruder knowledge, because of well-formedness.
then obtain ti where ti: "ti ∈ set T" "¬ikest (A@D') ·set I ⊢c ti"
  using Ana_fun_subterm[OF (Ana (Fun f T) = (K,M))] by (auto simp add: ikest_append)
obtain S where fS:
  "Fun f S ∈ subtermsset (ikest (A@D')) ∨
  Fun f S ∈ subtermsset (assignment_rhsest (A@D'))"
  "I x = Fun f S · I"
  using strand_sem_wf_ik_or_assignment_rhs_fun_subterm[
    OF AD'_props (Var x ∈ ikest (A@D')) _ ti (interpretationsubst I)]
    (I x = Fun f T)
  by moura
hence fS_in: "Fun f S · I ∈ ikest A ∪ ikest D' ·set I"
  "Fun f S ∈ subtermsset (ikest A ∪ assignment_rhsest A)"
  using imageI[OF s(1), of "λx. x · I"] Var
    ikest_append[of A D'] assignment_rhsest_append[of A D']
    decompest_subterms[OF D'(1)] decompest_assignment_rhs_empty[OF D'(1)]
  by auto
obtain KS MS where Ana_fS: "Ana (Fun f S) = (KS, MS)" by moura
hence "K = KS ·list I" "M = MS ·list I"
  using Ana_invar_substD[OF assms(5) fS_in(2)]
    s(2) fS(2) (s = Var x) (Ana (Fun f T) = (K,M))
  by simp_all
hence "MS ≠ []" using (M ≠ []) by simp
have "∧k. k ∈ set KS ⇒ ikest A ∪ ikest D' ·set I ⊢c k · I"
  using AxiomC.prem(1) (K = KS ·list I) by (simp add: image_Un)
hence D'': "D'@[Decomp (Fun f S)] ∈ decompest (ikest A) (assignment_rhsest A) I"
  using decompest.Decomp[OF D'(1) fS_in(2) Ana_fS (MS ≠ [])] AxiomC.prem(1)
    intruder_synth.AxiomC[OF fS_in(1)]
  by simp
moreover {
  fix t' assume "(ikest A ·set I) ∪ ikest (D@[Decomp (Fun f T)]) ⊢c t'"
  hence "(ikest A ·set I) ∪ (ikest (D'@[Decomp (Fun f S)]) ·set I) ⊢c t'"
  proof (induction t' rule: intruder_synth_induct)
    case (AxiomC t')
    hence "t' ∈ (ikest A ·set I) ∪ ikest D ∨ t' ∈ ikest [Decomp (Fun f T)]"
      by (simp add: ikest_append)
    thus ?case
      proof
        assume "t' ∈ ikest [Decomp (Fun f T)]"
        hence "t' ∈ ikest [Decomp (Fun f S)] ·set I"
          using decomp_ik (Ana (Fun f T) = (K,M)) (Ana (Fun f S) = (KS,MS)) (M = MS ·list I)
          by simp
        thus ?case
          using ideduct_synth_mono[
            OF intruder_synth.AxiomC[of t' "ikest [Decomp (Fun f S)] ·set I"],
            of "(ikest A ·set I) ∪ (ikest (D'@[Decomp (Fun f S)]) ·set I)"]

```

```

      by (auto simp add: ikest_append)
next
  assume "t' ∈ (ikest A ·set I) ∪ ikest D"
  hence "(ikest A ·set I) ∪ (ikest D' ·set I) ⊢c t'"
    by (metis D'(2) intruder_synth.AxiomC)
  hence "(ikest A ·set I) ∪ (ikest D' ·set I) ∪ (ikest [Decomp (Fun f S)] ·set I) ⊢c t'"
    by (simp add: ideduct_synth_mono)
  thus ?case
    using ikest_append[of D' "[Decomp (Fun f S)]"]
      image_Un[of "λx. x · I" "ikest D'" "ikest [Decomp (Fun f S)]"]
    by (simp add: sup_aci(2))
  qed
qed auto
}
ultimately show ?thesis using D'' by auto
qed
next
case (Fun g S) — Hence Decomp (Fun f T) can be substituted for Decomp (Fun g S)
hence KM: "K = Ks ·list I" "M = Ms ·list I" "set K = set Ks ·set I" "set M = set Ms ·set I"
  using fT_s_in(2) ⟨Ana (Fun f T) = (K,M)⟩ Ana_s s(2)
  Ana_invar_substD[OF assms(5), of g S]
  by auto
hence Ms_nonempty: "Ms ≠ []" using ⟨M ≠ []⟩ by auto
{ fix t' assume "(ikest A ·set I) ∪ ikest (D@[Decomp (Fun f T)]) ⊢c t'"
  hence "(ikest A ·set I) ∪ (ikest (D'@[Decomp (Fun g S)]) ·set I) ⊢c t'" using AxiomC
  proof (induction t' rule: intruder_synth_induct)
    case (AxiomC t')
    hence "t' ∈ ikest A ·set I ∨ t' ∈ ikest D ∨ t' ∈ set M"
      by (simp add: decomp_ik ikest_append)
    thus ?case
      proof (elim disjE)
        assume "t' ∈ ikest D"
        hence *: "(ikest A ·set I) ∪ (ikest D' ·set I) ⊢c t'" using D'(2) by simp
        show ?case by (auto intro: ideduct_synth_mono[OF *] simp add: ikest_append_subst(2))
      next
        assume "t' ∈ set M"
        hence "t' ∈ ikest [Decomp (Fun g S)] ·set I"
          using KM(2) Fun decomp_ik[OF Ana_s] by auto
        thus ?case by (simp add: image_Un ikest_append)
      qed (simp add: ideduct_synth_mono[OF intruder_synth.AxiomC])
    qed auto
  }
}
thus ?thesis
  using s Fun Ana_s AxiomC.premis(1) KM(3) fT_s_in
    decompest.Decomp[OF D'(1) _ Ms_nonempty, of g S Ks]
  by (metis AxiomC.hyps image_Un image_eqI intruder_synth.AxiomC)
qed
next
case (ComposeC T f)
have *: "∧m. m ∈ set M ⇒ (ikest A ·set I) ∪ (ikest D' ·set I) ⊢c m"
  using Ana_fun_subterm[OF ⟨Ana (Fun f T) = (K, M)⟩] ComposeC.hyps(3)
  by auto
have **: "ikest (D@[Decomp (Fun f T)]) = ikest D ∪ set M"
  using decomp_ik[OF ⟨Ana (Fun f T) = (K, M)⟩] ikest_append by auto
{ fix t' assume "(ikest A ·set I) ∪ ikest (D@[Decomp (Fun f T)]) ⊢c t'"
  hence "(ikest A ·set I) ∪ (ikest D' ·set I) ⊢c t'"
    by (induct rule: intruder_synth_induct) (auto simp add: D'(2) * **)
}
thus ?case using D'(1) by auto
qed
qed

```

thus ?thesis using D(2) assms(1) by (auto simp add: ik<sub>est</sub>\_append\_subst(2))  
qed

private lemma wf<sub>sts</sub>'\_update<sub>st</sub>\_nil: assumes "wf<sub>sts</sub>' S A" shows "wf<sub>sts</sub>' (update<sub>st</sub> S []) A"  
using assms unfolding wf<sub>sts</sub>'\_def by auto

private lemma wf<sub>sts</sub>'\_update<sub>st</sub>\_snd:  
assumes "wf<sub>sts</sub>' S A" "send⟨t⟩<sub>st</sub>#S ∈ S"  
shows "wf<sub>sts</sub>' (update<sub>st</sub> S (send⟨t⟩<sub>st</sub>#S)) (A@[Step (receive⟨t⟩<sub>st</sub>)])"  
unfolding wf<sub>sts</sub>'\_def

proof (intro conjI)

let ?S = "send⟨t⟩<sub>st</sub>#S"

let ?A = "A@[Step (receive⟨t⟩<sub>st</sub>)]"

have S: "∀S'. S' ∈ update<sub>st</sub> S ?S ⇒ S' = S ∨ S' ∈ S" by auto

have 1: "∀S ∈ S. wf<sub>st</sub> (wfrestrictedvars<sub>est</sub> A) (dual<sub>st</sub> S)" using assms unfolding wf<sub>sts</sub>'\_def by auto

moreover have 2: "wfrestrictedvars<sub>est</sub> ?A = wfrestrictedvars<sub>est</sub> A ∪ fv t"

using wfrestrictedvars<sub>est</sub>\_split(2) by (auto simp add: Un\_assoc)

ultimately have 3: "∀S ∈ S. wf<sub>st</sub> (wfrestrictedvars<sub>est</sub> ?A) (dual<sub>st</sub> S)" by (metis wf\_vars\_mono)

have 4: "∀S ∈ S. ∀S' ∈ S. fv<sub>st</sub> S ∩ bvars<sub>st</sub> S' = {}" using assms unfolding wf<sub>sts</sub>'\_def by simp

have "wf<sub>st</sub> (wfrestrictedvars<sub>est</sub> ?A) (dual<sub>st</sub> S)" using 1 2 3 assms(2) by auto

thus "∀S ∈ update<sub>st</sub> S ?S. wf<sub>st</sub> (wfrestrictedvars<sub>est</sub> ?A) (dual<sub>st</sub> S)" by (metis 3 S)

have "fv<sub>st</sub> S ∩ bvars<sub>st</sub> S = {}"

"∀S' ∈ S. fv<sub>st</sub> S ∩ bvars<sub>st</sub> S' = {}"

"∀S' ∈ S. fv<sub>st</sub> S' ∩ bvars<sub>st</sub> S = {}"

using 4 assms(2) unfolding wf<sub>sts</sub>'\_def by force+

thus "∀S ∈ update<sub>st</sub> S ?S. ∀S' ∈ update<sub>st</sub> S ?S. fv<sub>st</sub> S ∩ bvars<sub>st</sub> S' = {}" by (metis 4 S)

have "∀S' ∈ S. fv<sub>st</sub> ?S ∩ bvars<sub>st</sub> S' = {}" "∀S' ∈ S. fv<sub>st</sub> S' ∩ bvars<sub>st</sub> ?S = {}"

using assms unfolding wf<sub>sts</sub>'\_def by metis+

hence 5: "fv<sub>est</sub> ?A = fv<sub>est</sub> A ∪ fv t" "bvars<sub>est</sub> ?A = bvars<sub>est</sub> A" "∀S' ∈ S. fv t ∩ bvars<sub>st</sub> S' = {}"

using to\_st\_append by fastforce+

have \*: "∀S ∈ S. fv<sub>st</sub> S ∩ bvars<sub>est</sub> ?A = {}"

using 5 assms(1) unfolding wf<sub>sts</sub>'\_def by fast

hence "fv<sub>st</sub> ?S ∩ bvars<sub>est</sub> ?A = {}" using assms(2) by metis

hence "fv<sub>st</sub> S ∩ bvars<sub>est</sub> ?A = {}" by auto

thus "∀S ∈ update<sub>st</sub> S ?S. fv<sub>st</sub> S ∩ bvars<sub>est</sub> ?A = {}" by (metis \* S)

have \*\*: "∀S ∈ S. fv<sub>est</sub> ?A ∩ bvars<sub>st</sub> S = {}"

using 5 assms(1) unfolding wf<sub>sts</sub>'\_def by fast

hence "fv<sub>est</sub> ?A ∩ bvars<sub>st</sub> ?S = {}" using assms(2) by metis

hence "fv<sub>est</sub> ?A ∩ bvars<sub>st</sub> S = {}" by fastforce

thus "∀S ∈ update<sub>st</sub> S ?S. fv<sub>est</sub> ?A ∩ bvars<sub>st</sub> S = {}" by (metis \*\* S)

qed

private lemma wf<sub>sts</sub>'\_update<sub>st</sub>\_rcv:  
assumes "wf<sub>sts</sub>' S A" "receive⟨t⟩<sub>st</sub>#S ∈ S"  
shows "wf<sub>sts</sub>' (update<sub>st</sub> S (receive⟨t⟩<sub>st</sub>#S)) (A@[Step (send⟨t⟩<sub>st</sub>)])"  
unfolding wf<sub>sts</sub>'\_def

proof (intro conjI)

let ?S = "receive⟨t⟩<sub>st</sub>#S"

let ?A = "A@[Step (send⟨t⟩<sub>st</sub>)]"

have S: "∀S'. S' ∈ update<sub>st</sub> S ?S ⇒ S' = S ∨ S' ∈ S" by auto

have 1: "∀S ∈ S. wf<sub>st</sub> (wfrestrictedvars<sub>est</sub> A) (dual<sub>st</sub> S)" using assms unfolding wf<sub>sts</sub>'\_def by auto

moreover have 2: "wfrestrictedvars<sub>est</sub> ?A = wfrestrictedvars<sub>est</sub> A ∪ fv t"

using wfrestrictedvars<sub>est</sub>\_split(2) by (auto simp add: Un\_assoc)



ultimately have 3: " $\forall S \in \mathcal{S}. wf_{st} (wf_{restrictedvars}_{est} ?A) (dual_{st} S)$ " by (metis wf\_vars\_mono)

have 4: " $\forall S \in \mathcal{S}. \forall S' \in \mathcal{S}. fv_{st} S \cap bvars_{st} S' = \{\}$ " using assms unfolding wf\_{sts}'\_def by simp

have " $wf_{st} (wf_{restrictedvars}_{est} ?A) (dual_{st} S)$ " using 1 2 3 assms(2) by auto

thus " $\forall S \in update_{st} \mathcal{S} ?S. wf_{st} (wf_{restrictedvars}_{est} ?A) (dual_{st} S)$ " by (metis 3 S)

have " $fv_{st} S \cap bvars_{st} S = \{\}$ "

" $\forall S' \in \mathcal{S}. fv_{st} S \cap bvars_{st} S' = \{\}$ "

" $\forall S' \in \mathcal{S}. fv_{st} S' \cap bvars_{st} S = \{\}$ "

using 4 assms(2) unfolding wf\_{sts}'\_def by force+

thus " $\forall S \in update_{st} \mathcal{S} ?S. \forall S' \in update_{st} \mathcal{S} ?S. fv_{st} S \cap bvars_{st} S' = \{\}$ " by (metis 4 S)

have " $\forall S' \in \mathcal{S}. fv_{st} ?S \cap bvars_{st} S' = \{\}$ " " $\forall S' \in \mathcal{S}. fv_{st} S' \cap bvars_{st} ?S = \{\}$ "

using assms unfolding wf\_{sts}'\_def by metis+

hence 5: " $fv_{est} ?A = fv_{est} \mathcal{A} \cup fv \ t$ " " $bvars_{est} ?A = bvars_{est} \mathcal{A}$ " " $\forall S' \in \mathcal{S}. fv \ t \cap bvars_{st} S' = \{\}$ "

using to\_st\_append by fastforce+

have \*: " $\forall S \in \mathcal{S}. fv_{st} S \cap bvars_{est} ?A = \{\}$ "

using 5 assms(1) unfolding wf\_{sts}'\_def by fast

hence " $fv_{st} ?S \cap bvars_{est} ?A = \{\}$ " using assms(2) by metis

hence " $fv_{st} S \cap bvars_{est} ?A = \{\}$ " by auto

thus " $\forall S \in update_{st} \mathcal{S} ?S. fv_{st} S \cap bvars_{est} ?A = \{\}$ " by (metis \* S)

have \*\*: " $\forall S \in \mathcal{S}. fv_{est} ?A \cap bvars_{st} S = \{\}$ "

using 5 assms(1) unfolding wf\_{sts}'\_def by fast

hence " $fv_{est} ?A \cap bvars_{st} ?S = \{\}$ " using assms(2) by metis

hence " $fv_{est} ?A \cap bvars_{st} S = \{\}$ " by fastforce

thus " $\forall S \in update_{st} \mathcal{S} ?S. fv_{est} ?A \cap bvars_{st} S = \{\}$ " by (metis \*\* S)

qed

private lemma wf\_{sts}'\_update\_{st}\_eq:

assumes " $wf_{sts}' \ S \ \mathcal{A}$ " " $\langle a: t \doteq t' \rangle_{st} \# S \in \mathcal{S}$ "

shows " $wf_{sts}' (update_{st} \mathcal{S} (\langle a: t \doteq t' \rangle_{st} \# S)) (\mathcal{A}@[Step (\langle a: t \doteq t' \rangle_{st})])$ "

unfolding wf\_{sts}'\_def

proof (intro conjI)

let ?S = " $\langle a: t \doteq t' \rangle_{st} \# S$ "

let ?A = " $\mathcal{A}@[Step (\langle a: t \doteq t' \rangle_{st})]$ "

have S: " $\bigwedge S'. S' \in update_{st} \mathcal{S} ?S \implies S' = S \vee S' \in \mathcal{S}$ " by auto

have 1: " $\forall S \in \mathcal{S}. wf_{st} (wf_{restrictedvars}_{est} \mathcal{A}) (dual_{st} S)$ " using assms unfolding wf\_{sts}'\_def by auto

moreover have 2:

"a = Assign  $\implies wf_{restrictedvars}_{est} ?A = wf_{restrictedvars}_{est} \mathcal{A} \cup fv \ t \cup fv \ t'$ "

"a = Check  $\implies wf_{restrictedvars}_{est} ?A = wf_{restrictedvars}_{est} \mathcal{A}$ "

using wf\_{restrictedvars}\_{est}\_split(2) by (auto simp add: Un\_assoc)

ultimately have 3: " $\forall S \in \mathcal{S}. wf_{st} (wf_{restrictedvars}_{est} ?A) (dual_{st} S)$ "

by (cases a) (metis wf\_vars\_mono, metis)

have 4: " $\forall S \in \mathcal{S}. \forall S' \in \mathcal{S}. fv_{st} S \cap bvars_{st} S' = \{\}$ " using assms unfolding wf\_{sts}'\_def by simp

have " $wf_{st} (wf_{restrictedvars}_{est} ?A) (dual_{st} S)$ " using 1 2 3 assms(2) by (cases a) auto

thus " $\forall S \in update_{st} \mathcal{S} ?S. wf_{st} (wf_{restrictedvars}_{est} ?A) (dual_{st} S)$ " by (metis 3 S)

have " $fv_{st} S \cap bvars_{st} S = \{\}$ "

" $\forall S' \in \mathcal{S}. fv_{st} S \cap bvars_{st} S' = \{\}$ "

" $\forall S' \in \mathcal{S}. fv_{st} S' \cap bvars_{st} S = \{\}$ "

using 4 assms(2) unfolding wf\_{sts}'\_def by force+

thus " $\forall S \in update_{st} \mathcal{S} ?S. \forall S' \in update_{st} \mathcal{S} ?S. fv_{st} S \cap bvars_{st} S' = \{\}$ " by (metis 4 S)

have " $\forall S' \in \mathcal{S}. fv_{st} ?S \cap bvars_{st} S' = \{\}$ " " $\forall S' \in \mathcal{S}. fv_{st} S' \cap bvars_{st} ?S = \{\}$ "

using assms unfolding wf\_{sts}'\_def by metis+

hence 5: " $fv_{est} ?A = fv_{est} \mathcal{A} \cup fv \ t \cup fv \ t'$ " " $bvars_{est} ?A = bvars_{est} \mathcal{A}$ "

```

    "∀S' ∈ S. fv t ∩ bvarsst S' = {}" "∀S' ∈ S. fv t' ∩ bvarsst S' = {}"
    using to_st_append by fastforce+

  have *: "∀S ∈ S. fvst S ∩ bvarsest ?A = {}"
    using 5 assms(1) unfolding wfsts'_def by fast
  hence "fvst ?S ∩ bvarsest ?A = {}" using assms(2) by metis
  hence "fvst S ∩ bvarsest ?A = {}" by auto
  thus "∀S ∈ updatest S ?S. fvst S ∩ bvarsest ?A = {}" by (metis * S)

  have **: "∀S ∈ S. fvest ?A ∩ bvarsst S = {}"
    using 5 assms(1) unfolding wfsts'_def by fast
  hence "fvest ?A ∩ bvarsst ?S = {}" using assms(2) by metis
  hence "fvest ?A ∩ bvarsst S = {}" by fastforce
  thus "∀S ∈ updatest S ?S. fvest ?A ∩ bvarsst S = {}" by (metis ** S)
qed

private lemma wfsts'_updatest_ineq:
  assumes "wfsts' S A" "∀X(∀≠: F)st#S ∈ S"
  shows "wfsts' (updatest S (∀X(∀≠: F)st#S)) (A@[Step (∀X(∀≠: F)st)])"
unfolding wfsts'_def
proof (intro conjI)
  let ?S = "∀X(∀≠: F)st#S"
  let ?A = "A@[Step (∀X(∀≠: F)st)]"

  have S: "∀S'. S' ∈ updatest S ?S ⇒ S' = S ∨ S' ∈ S" by auto

  have 1: "∀S ∈ S. wfst (wfrestrictedvarsest A) (dualst S)" using assms unfolding wfsts'_def by auto
  moreover have 2: "wfrestrictedvarsest ?A = wfrestrictedvarsest A"
    using wfrestrictedvarsest_split(2) by (auto simp add: Un_assoc)
  ultimately have 3: "∀S ∈ S. wfst (wfrestrictedvarsest ?A) (dualst S)" by metis

  have 4: "∀S ∈ S. ∀S' ∈ S. fvst S ∩ bvarsst S' = {}" using assms unfolding wfsts'_def by simp

  have "wfst (wfrestrictedvarsest ?A) (dualst S)" using 1 2 3 assms(2) by auto
  thus "∀S ∈ updatest S ?S. wfst (wfrestrictedvarsest ?A) (dualst S)" by (metis 3 S)

  have "fvst S ∩ bvarsst S = {}"
    "∀S' ∈ S. fvst S ∩ bvarsst S' = {}"
    "∀S' ∈ S. fvst S' ∩ bvarsst S = {}"
    using 4 assms(2) unfolding wfsts'_def by force+
  thus "∀S ∈ updatest S ?S. ∀S' ∈ updatest S ?S. fvst S ∩ bvarsst S' = {}" by (metis 4 S)

  have "∀S' ∈ S. fvst ?S ∩ bvarsst S' = {}" "∀S' ∈ S. fvst S' ∩ bvarsst ?S = {}"
    using assms unfolding wfsts'_def by metis+
  moreover have "fvpairs F - set X ⊆ fvst (∀X(∀≠: F)st # S)" by auto
  ultimately have 5:
    "∀S' ∈ S. (fvpairs F - set X) ∩ bvarsst S' = {}"
    "fvest ?A = fvest A ∪ (fvpairs F - set X)" "bvarsest ?A = set X ∪ bvarsest A"
    "∀S ∈ S. fvst S ∩ set X = {}"
    using to_st_append
    by (blast, force, force, force)

  have *: "∀S ∈ S. fvst S ∩ bvarsest ?A = {}" using 5(3,4) assms(1) unfolding wfsts'_def by blast
  hence "fvst ?S ∩ bvarsest ?A = {}" using assms(2) by metis
  hence "fvst S ∩ bvarsest ?A = {}" by auto
  thus "∀S ∈ updatest S ?S. fvst S ∩ bvarsest ?A = {}" by (metis * S)

  have **: "∀S ∈ S. fvest ?A ∩ bvarsst S = {}"
    using 5(1,2) assms(1) unfolding wfsts'_def by fast
  hence "fvest ?A ∩ bvarsst ?S = {}" using assms(2) by metis
  hence "fvest ?A ∩ bvarsst S = {}" by auto
  thus "∀S ∈ updatest S ?S. fvest ?A ∩ bvarsst S = {}" by (metis ** S)
qed

```

```

private lemma trmsst_updatest_eq:
  assumes "x#S ∈ S"
  shows "⋃ (trmsst ' S) ∪ trmsstp x = ⋃ (trmsst ' S)" (is "?A = ?B")
proof
  show "?B ⊆ ?A"
  proof
    have "trmsstp x ⊆ trmsst (x#S)" by auto
    hence "⋀ t'. t' ∈ ?B ⇒ t' ∈ trmsstp x ⇒ t' ∈ ?A" by simp
    moreover {
      fix t' assume t': "t' ∈ ?B" "t' ∉ trmsstp x"
      then obtain S' where S': "t' ∈ trmsst S'" "S' ∈ S" by auto
      hence "S' = x#S ∨ S' ∈ updatest S (x#S)" by auto
      moreover {
        assume "S' = x#S"
        hence "t' ∈ trmsst S" using S' t' by simp
        hence "t' ∈ ?A" by auto
      }
      ultimately have "t' ∈ ?A" using t' S' by auto
    }
    ultimately show "⋀ t'. t' ∈ ?B ⇒ t' ∈ ?A" by metis
  qed

  show "?A ⊆ ?B"
  proof
    have "⋀ t'. t' ∈ ?A ⇒ t' ∈ trmsstp x ⇒ trmsstp x ⊆ ?B"
      using assms by force+
    moreover {
      fix t' assume t': "t' ∈ ?A" "t' ∉ trmsstp x"
      then obtain S' where "t' ∈ trmsst S'" "S' ∈ updatest S (x#S)" by auto
      hence "S' = S ∨ S' ∈ S" by auto
      moreover have "trmsst S ⊆ ?B" using assms trmsst_cons[of x S] by blast
      ultimately have "t' ∈ ?B" using t' by fastforce
    }
    ultimately show "⋀ t'. t' ∈ ?A ⇒ t' ∈ ?B" by blast
  qed
qed

private lemma trmsst_updatest_eq_snd:
  assumes "send⟨t⟩st#S ∈ S" "S' = updatest S (send⟨t⟩st#S)" "A' = A@[Step (receive⟨t⟩st)]"
  shows "(⋃ (trmsst ' S)) ∪ (trmsest A) = (⋃ (trmsst ' S')) ∪ (trmsest A')"
proof -
  have "(trmsest A') = (trmsest A) ∪ {t}" "⋃ (trmsst ' S') ∪ {t} = ⋃ (trmsst ' S)"
    using to_st_append trmsst_updatest_eq[OF assms(1)] assms(2,3) by auto
  thus ?thesis
    by (metis (no_types, lifting) Un_insert_left Un_insert_right sup_bot.right_neutral)
qed

private lemma trmsst_updatest_eq_rcv:
  assumes "receive⟨t⟩st#S ∈ S" "S' = updatest S (receive⟨t⟩st#S)" "A' = A@[Step (send⟨t⟩st)]"
  shows "(⋃ (trmsst ' S)) ∪ (trmsest A) = (⋃ (trmsst ' S')) ∪ (trmsest A')"
proof -
  have "(trmsest A') = (trmsest A) ∪ {t}" "⋃ (trmsst ' S') ∪ {t} = ⋃ (trmsst ' S)"
    using to_st_append trmsst_updatest_eq[OF assms(1)] assms(2,3) by auto
  thus ?thesis
    by (metis (no_types, lifting) Un_insert_left Un_insert_right sup_bot.right_neutral)
qed

private lemma trmsst_updatest_eq_eq:
  assumes "⟨a: t ≐ t'⟩st#S ∈ S" "S' = updatest S (⟨a: t ≐ t'⟩st#S)" "A' = A@[Step (⟨a: t ≐ t'⟩st)]"
  shows "(⋃ (trmsst ' S)) ∪ (trmsest A) = (⋃ (trmsst ' S')) ∪ (trmsest A')"
proof -

```

```

have "(trmsest A') = (trmsest A) ∪ {t, t'}" "⋃(trmsst ' S') ∪ {t, t'} = ⋃(trmsst ' S)"
  using to_st_append trmsst_updatest_eq[OF assms(1)] assms(2,3) by auto
thus ?thesis
  by (metis (no_types, lifting) Un_insert_left Un_insert_right sup_bot.right_neutral)
qed

private lemma trmsst_updatest_eq_ineq:
  assumes "∀X⟨V≠: F⟩st#S ∈ S" "S' = updatest S (∀X⟨V≠: F⟩st#S)" "A' = A@[Step (∀X⟨V≠: F⟩st)]"
  shows "(⋃(trmsst ' S)) ∪ (trmsest A) = (⋃(trmsst ' S')) ∪ (trmsest A')"
proof -
  have "(trmsest A') = (trmsest A) ∪ trmspairs F" "⋃(trmsst ' S') ∪ trmspairs F = ⋃(trmsst ' S)"
    using to_st_append trmsst_updatest_eq[OF assms(1)] assms(2,3) by auto
  thus ?thesis by (simp add: Un_commute sup_left_commute)
qed

private lemma ikst_updatest_subset:
  assumes "x#S ∈ S"
  shows "⋃(ikst'dualst ' (updatest S (x#S))) ⊆ ⋃(ikst'dualst ' S)" (is ?A)
    "⋃(assignment_rhsst ' (updatest S (x#S))) ⊆ ⋃(assignment_rhsst ' S)" (is ?B)
proof -
  { fix t assume "t ∈ ⋃(ikst'dualst ' (updatest S (x#S)))"
    then obtain S' where S': "S' ∈ updatest S (x#S)" "t ∈ ikst (dualst S'" by auto

    have *: "ikst (dualst S) ⊆ ikst (dualst (x#S))"
      using ik_append[of "dualst [x]" "dualst S"] dualst_append[of "[x]" S]
      by auto

    hence "t ∈ ⋃(ikst'dualst ' S)"
    proof (cases "S' = S")
      case True thus ?thesis using * assms S' by auto
    next
      case False thus ?thesis using S' by auto
    qed
  }
  moreover
  { fix t assume "t ∈ ⋃(assignment_rhsst ' (updatest S (x#S)))"
    then obtain S' where S': "S' ∈ updatest S (x#S)" "t ∈ assignment_rhsst S'" by auto

    have "assignment_rhsst S ⊆ assignment_rhsst (x#S)"
      using assignment_rhs_append[of "[x]" S] by simp
    hence "t ∈ ⋃(assignment_rhsst ' S)"
      using assms S' by (cases "S' = S") auto
  }
  ultimately show ?A ?B by (metis subsetI)+
qed

private lemma ikst_updatest_subset_snd:
  assumes "send⟨t⟩st#S ∈ S"
    "S' = updatest S (send⟨t⟩st#S)"
    "A' = A@[Step (receive⟨t⟩st)]"
  shows "(⋃(ikst ' dualst ' S')) ∪ (ikest A') ⊆
    (⋃(ikst ' dualst ' S)) ∪ (ikest A)" (is ?A)
    "(⋃(assignment_rhsst ' S')) ∪ (assignment_rhsest A') ⊆
    (⋃(assignment_rhsst ' S)) ∪ (assignment_rhsest A)" (is ?B)
proof -
  { fix t' assume t'_in: "t' ∈ (⋃(ikst'dualst ' S')) ∪ (ikest A'"
    hence "t' ∈ (⋃(ikst'dualst ' S')) ∪ (ikest A) ∪ {t}" using assms ikest_append by auto
    moreover have "t ∈ ⋃(ikst'dualst ' S)" using assms(1) by force
    ultimately have "t' ∈ (⋃(ikst'dualst ' S)) ∪ (ikest A)"
      using ikst_updatest_subset[OF assms(1)] assms(2) by auto
  }
  moreover
  { fix t' assume t'_in: "t' ∈ (⋃(assignment_rhsst ' S')) ∪ (assignment_rhsest A'"

```

```

    hence "t' ∈ (⋃ (assignment_rhs_st ' S')) ∪ (assignment_rhs_est A)"
      using assms assignment_rhs_est_append by auto
    hence "t' ∈ (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A)"
      using ik_st_update_st_subset[OF assms(1)] assms(2) by auto
  }
  ultimately show ?A ?B by (metis subsetI)+
qed

private lemma ik_st_update_st_subset_rcv:
  assumes "receive⟨t⟩_st # S ∈ S"
    "S' = update_st S (receive⟨t⟩_st # S)"
    "A' = A@[Step (send⟨t⟩_st)]"
  shows "(⋃ (ik_st ' dual_st ' S')) ∪ (ik_est A') ⊆
    (⋃ (ik_st ' dual_st ' S)) ∪ (ik_est A)" (is ?A)
    "(⋃ (assignment_rhs_st ' S')) ∪ (assignment_rhs_est A') ⊆
    (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A)" (is ?B)
proof -
  { fix t' assume t'_in: "t' ∈ (⋃ (ik_st ' dual_st ' S')) ∪ (ik_est A)"
    hence "t' ∈ (⋃ (ik_st ' dual_st ' S)) ∪ (ik_est A)" using assms ik_est_append by auto
    hence "t' ∈ (⋃ (ik_st ' dual_st ' S)) ∪ (ik_est A)"
      using ik_st_update_st_subset[OF assms(1)] assms(2) by auto
  }
  moreover
  { fix t' assume t'_in: "t' ∈ (⋃ (assignment_rhs_st ' S')) ∪ (assignment_rhs_est A)"
    hence "t' ∈ (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A)"
      using assms assignment_rhs_est_append by auto
    hence "t' ∈ (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A)"
      using ik_st_update_st_subset[OF assms(1)] assms(2) by auto
  }
  ultimately show ?A ?B by (metis subsetI)+
qed

private lemma ik_st_update_st_subset_eq:
  assumes "⟨a: t ≐ t'⟩_st # S ∈ S"
    "S' = update_st S (⟨a: t ≐ t'⟩_st # S)"
    "A' = A@[Step (⟨a: t ≐ t'⟩_st)]"
  shows "(⋃ (ik_st ' dual_st ' S')) ∪ (ik_est A') ⊆
    (⋃ (ik_st ' dual_st ' S)) ∪ (ik_est A)" (is ?A)
    "(⋃ (assignment_rhs_st ' S')) ∪ (assignment_rhs_est A') ⊆
    (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A)" (is ?B)
proof -
  have 1: "t' ∈ (⋃ (ik_st ' dual_st ' S)) ∪ (ik_est A)"
    when "t' ∈ (⋃ (ik_st ' dual_st ' S)) ∪ (ik_est A)"
    for t'
  proof -
    have "t' ∈ (⋃ (ik_st ' dual_st ' S)) ∪ (ik_est A)" using that assms ik_est_append by auto
    thus ?thesis using ik_st_update_st_subset[OF assms(1)] assms(2) by auto
  qed

  have 2: "t'' ∈ (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A)"
    when "t'' ∈ (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A)" "a = Assign"
    for t''
  proof -
    have "t'' ∈ (⋃ (assignment_rhs_st ' S)) ∪ (assignment_rhs_est A) ∪ {t'}"
      using that assms assignment_rhs_est_append by auto
    moreover have "t' ∈ ⋃ (assignment_rhs_st ' S)" using assms(1) that by force
    ultimately show ?thesis using ik_st_update_st_subset[OF assms(1)] assms(2) that by auto
  qed

  have 3: "assignment_rhs_est A' = assignment_rhs_est A" (is ?C)
    "(⋃ (assignment_rhs_st ' S')) ⊆ (⋃ (assignment_rhs_st ' S))" (is ?D)
    when "a = Check"
  proof -

```

```

show ?C using that assms(2,3) by (simp add: assignment_rhs_est_append)
show ?D using assms(1,2,3) ik_st_update_st_subset(2) by auto
qed

show ?A using 1 2 by (metis subsetI)
show ?B using 1 2 3 by (cases a) blast+
qed

private lemma ik_st_update_st_subset_ineq:
  assumes "\X<\neq: F>_st#S \in S"
    "S' = update_st S (\X<\neq: F>_st#S)"
    "A' = A@[Step (\X<\neq: F>_st)]"
  shows "( \ (ik_st 'dual_st ' S')) \cup (ik_est A') \subseteq
    ( \ (ik_st 'dual_st ' S)) \cup (ik_est A)" (is ?A)
    "( \ (assignment_rhs_st ' S')) \cup (assignment_rhs_est A') \subseteq
    ( \ (assignment_rhs_st ' S)) \cup (assignment_rhs_est A)" (is ?B)
proof -
  { fix t' assume t'_in: "t' \in ( \ (ik_st 'dual_st ' S')) \cup (ik_est A)"
    hence "t' \in ( \ (ik_st 'dual_st ' S)) \cup (ik_est A)" using assms ik_est_append by auto
    hence "t' \in ( \ (ik_st 'dual_st ' S)) \cup (ik_est A)"
      using ik_st_update_st_subset[OF assms(1)] assms(2) by auto
  }
  moreover
  { fix t' assume t'_in: "t' \in ( \ (assignment_rhs_st ' S')) \cup (assignment_rhs_est A)"
    hence "t' \in ( \ (assignment_rhs_st ' S)) \cup (assignment_rhs_est A)"
      using assms assignment_rhs_est_append by auto
    hence "t' \in ( \ (assignment_rhs_st ' S)) \cup (assignment_rhs_est A)"
      using ik_st_update_st_subset[OF assms(1)] assms(2) by auto
  }
  ultimately show ?A ?B by (metis subsetI)+
qed

```

### Transition Systems Definitions

```

inductive pts_symbolic::
  "((\fun,\var) strands \times (\fun,\var) strand) \Rightarrow
    ((\fun,\var) strands \times (\fun,\var) strand) \Rightarrow bool"
(infix "\Rightarrow^*" 50) where
  Nil[simp]:      "[ ] \in S \Longrightarrow (S,A) \Rightarrow^* (update_st S [ ],A)"
| Send[simp]:     "send<t>_st#S \in S \Longrightarrow (S,A) \Rightarrow^* (update_st S (send<t>_st#S),A@[receive<t>_st])"
| Receive[simp]:  "receive<t>_st#S \in S \Longrightarrow (S,A) \Rightarrow^* (update_st S (receive<t>_st#S),A@[send<t>_st])"
| Equality[simp]: "\langle a: t \doteq t' \rangle_st#S \in S \Longrightarrow (S,A) \Rightarrow^* (update_st S (\langle a: t \doteq t' \rangle_st#S),A@[ \langle a: t \doteq t' \rangle_st])"
| Inequality[simp]: "\X<\neq: F>_st#S \in S \Longrightarrow (S,A) \Rightarrow^* (update_st S (\X<\neq: F>_st#S),A@[ \X<\neq: F>_st])"

private inductive pts_symbolic_c::
  "((\fun,\var) strands \times (\fun,\var) extstrand) \Rightarrow
    ((\fun,\var) strands \times (\fun,\var) extstrand) \Rightarrow bool"
(infix "\Rightarrow^*_c" 50) where
  Nil[simp]:      "[ ] \in S \Longrightarrow (S,A) \Rightarrow^*_c (update_st S [ ],A)"
| Send[simp]:     "send<t>_st#S \in S \Longrightarrow (S,A) \Rightarrow^*_c (update_st S (send<t>_st#S),A@[Step
(receive<t>_st)])"
| Receive[simp]:  "receive<t>_st#S \in S \Longrightarrow (S,A) \Rightarrow^*_c (update_st S (receive<t>_st#S),A@[Step
(send<t>_st)])"
| Equality[simp]:  "\langle a: t \doteq t' \rangle_st#S \in S \Longrightarrow (S,A) \Rightarrow^*_c (update_st S (\langle a: t \doteq t' \rangle_st#S),A@[Step
(\langle a: t \doteq t' \rangle_st)])"
| Inequality[simp]: "\X<\neq: F>_st#S \in S \Longrightarrow (S,A) \Rightarrow^*_c (update_st S (\X<\neq: F>_st#S),A@[Step
(\X<\neq: F>_st)])"
| Decompose[simp]: "Fun f T \in subterms_set (ik_est A \cup assignment_rhs_est A)
\Longrightarrow (S,A) \Rightarrow^*_c (S,A@[Decomp (Fun f T)])"

```

abbreviation pts\_symbolic\_rtrancl (infix "\Rightarrow^{\*\*}" 50) where "a \Rightarrow^{\*\*} b \equiv pts\_symbolic^{\*\*} a b"

```

private abbreviation pts_symbolic_c_rtrancl (infix "⇒•c" 50) where "a ⇒•c b ≡ pts_symbolic_c** a b"

lemma pts_symbolic_induct[consumes 1, case_names Nil Send Receive Equality Inequality]:
  assumes "(S, A) ⇒• (S', A')"
  and "[[] ∈ S; S' = updatest S []; A' = A] ⇒ P"
  and "⋀t S. [send⟨t⟩st#S ∈ S; S' = updatest S (send⟨t⟩st#S); A' = A@[receive⟨t⟩st]] ⇒ P"
  and "⋀t S. [receive⟨t⟩st#S ∈ S; S' = updatest S (receive⟨t⟩st#S); A' = A@[send⟨t⟩st]] ⇒ P"
  and "⋀a t t' S. [(a: t ≐ t')st#S ∈ S; S' = updatest S ((a: t ≐ t')st#S); A' = A@[a: t ≐ t']st]] ⇒ P"
  and "⋀X F S. [∀X⟨V≠: F⟩st#S ∈ S; S' = updatest S (∀X⟨V≠: F⟩st#S); A' = A@[∀X⟨V≠: F⟩st]] ⇒ P"
  shows "P"
apply (rule pts_symbolic.cases[OF assms(1)])
using assms(2,3,4,5,6) by simp_all

private lemma pts_symbolic_c_induct[consumes 1, case_names Nil Send Receive Equality Inequality Decompose]:
  assumes "(S, A) ⇒•c (S', A')"
  and "[[] ∈ S; S' = updatest S []; A' = A] ⇒ P"
  and "⋀t S. [send⟨t⟩st#S ∈ S; S' = updatest S (send⟨t⟩st#S); A' = A@[Step (receive⟨t⟩st)]] ⇒ P"
  and "⋀t S. [receive⟨t⟩st#S ∈ S; S' = updatest S (receive⟨t⟩st#S); A' = A@[Step (send⟨t⟩st)]] ⇒ P"
  and "⋀a t t' S. [(a: t ≐ t')st#S ∈ S; S' = updatest S ((a: t ≐ t')st#S); A' = A@[Step ((a: t ≐ t')st)]] ⇒ P"
  and "⋀X F S. [∀X⟨V≠: F⟩st#S ∈ S; S' = updatest S (∀X⟨V≠: F⟩st#S); A' = A@[Step (∀X⟨V≠: F⟩st)]] ⇒ P"
  and "⋀f T. [Fun f T ∈ subtermsset (ikest A ∪ assignment_rhsest A); S' = S; A' = A@[Decomp (Fun f T)]] ⇒ P"
  shows "P"
apply (rule pts_symbolic_c.cases[OF assms(1)])
using assms(2,3,4,5,6,7) by simp_all

private lemma pts_symbolic_c_preserves_wf_prot:
  assumes "(S, A) ⇒•c (S', A'" "wfsts' S A"
  shows "wfsts' S' A'"
using assms
proof (induction rule: rtranclp_induct2)
  case (step S1 A1 S2 A2)
  from step.hyps(2) step.IH[OF step.prem] show ?case
  proof (induction rule: pts_symbolic_c_induct)
    case Decompose
    hence "fvest A2 = fvest A1" "bvarsest A2 = bvarsest A1"
    using bvars_decomp ik_assignment_rhs_decomp_fv by metis+
    thus ?case using Decompose unfolding wfsts'_def
    by (metis wf_vars_mono wfrestrictedvarsest_split(2))
  qed (metis wfsts'_updatest_nil, metis wfsts'_updatest_snd,
    metis wfsts'_updatest_rcv, metis wfsts'_updatest_eq,
    metis wfsts'_updatest_ineq)
qed metis

private lemma pts_symbolic_c_preserves_wf_is:
  assumes "(S, A) ⇒•c (S', A'" "wfsts' S A" "wfst V (tost A)"
  shows "wfst V (tost A'"
using assms
proof (induction rule: rtranclp_induct2)
  case (step S1 A1 S2 A2)
  hence "(S, A) ⇒•c (S2, A2)" by auto
  hence *: "wfsts' S1 A1" "wfsts' S2 A2"
    using pts_symbolic_c_preserves_wf_prot[OF _ step.prem(1)] step.hyps(1)
    by auto
  from step.hyps(2) step.IH[OF step.prem] show ?case

```

```

proof (induction rule: pts_symbolic_c_induct)
  case Nil thus ?case by auto
next
  case (Send t S)
  hence "wfst (wfrestrictedvarsest A1) (receive⟨t⟩st#(dualst S))"
    using *(1) unfolding wfsts'_def by fastforce
  hence "fv t ⊆ wfrestrictedvarsst (tost A1) ∪ V"
    using wfrestrictedvarsest=wfrestrictedvarsst by auto
  thus ?case using Send wf_rcv_append''' tost_append by simp
next
  case (Receive t) thus ?case using wf_snd_append tost_append by simp
next
  case (Equality a t t' S)
  hence "wfst (wfrestrictedvarsest A1) (⟨a: t ≐ t'⟩st#(dualst S))"
    using *(1) unfolding wfsts'_def by fastforce
  hence "fv t' ⊆ wfrestrictedvarsst (tost A1) ∪ V" when "a = Assign"
    using wfrestrictedvarsest=wfrestrictedvarsst that by auto
  thus ?case using Equality wf_eq_append''' tost_append by (cases a) auto
next
  case (Inequality t t' S) thus ?case using wf_ineq_append''' tost_append by simp
next
  case (Decompose f T)
  hence "fv (Fun f T) ⊆ wfrestrictedvarsest A1"
    by (metis fv_subterms_set fv_subset subset_trans
      ikst_assignment_rhsst_wfrestrictedvars_subset)
  hence "varsst (decomp (Fun f T)) ⊆ wfrestrictedvarsst (tost A1) ∪ V"
    using decomp_vars[of "Fun f T"] wfrestrictedvarsest=wfrestrictedvarsst[of A1] by auto
  thus ?case
    using tost_append[of A1 "[Decomp (Fun f T)]"]
      wf_append_suffix[OF Decompose.prem] Decompose.hyps(3)
    by (metis append_Nil2 decomp_vars(1,2) tost.simps(1,3))
qed
qed metis

private lemma pts_symbolic_c_preserves_tfrset:
  assumes "(S, A) ⇒•c (S', A')"
  and "tfrset ((⋃ (trmsst ' S)) ∪ (trmsest A))"
  and "wftrms ((⋃ (trmsst ' S)) ∪ (trmsest A))"
  shows "tfrset ((⋃ (trmsst ' S')) ∪ (trmsest A')) ∧ wftrms ((⋃ (trmsst ' S')) ∪ (trmsest A'))"
using assms
proof (induction rule: rtrancp_induct2)
  case (step S1 A1 S2 A2)
  from step.hyps(2) step.IH[OF step.prem] show ?case
  proof (induction rule: pts_symbolic_c_induct)
    case Nil
    hence "⋃ (trmsst ' S1) = ⋃ (trmsst ' S2)" by force
    thus ?case using Nil by metis
  next
    case (Decompose f T)
    obtain t where t: "t ∈ ikest A1 ∪ assignment_rhsest A1" "Fun f T ⊆ t"
      using Decompose.hyps(1) by auto
    have t_wf: "wftrm t"
      using Decompose.prem wf_trm_subterm[of _ t]
        trmsest_ik_assignment_rhsI[OF t(1)]
      unfolding tfrset_def
      by (metis UN_E Un_iff)
    have "t ∈ subtermsset (trmsest A1)" using trmsest_ik_assignment_rhsI t by auto
    hence "Fun f T ∈ SMP (trmsest A1)"
      by (metis (no_types) SMP.MP SMP.Subterm UN_E t(2))
    hence "{Fun f T} ⊆ SMP (trmsest A1)" using SMP.Subterm[of "Fun f T"] by auto
    moreover have "trmsest A2 = insert (Fun f T) (trmsest A1)"
      using Decompose.hyps(3) by auto
    ultimately have *: "SMP (trmsest A1) = SMP (trmsest A2)"

```



```

    using SMP_subset_union_eq[of "{Fun f T}"]
    by (simp add: Un_commute)
  hence "SMP (( $\bigcup$  (trmsst ' S1))  $\cup$  (trmsest A1)) = SMP (( $\bigcup$  (trmsst ' S2))  $\cup$  (trmsest A2))"
    using Decompose.hyps(2) SMP_union by auto
  moreover have " $\forall t \in \text{trms}_{est} A1. \text{wf}_{trm} t$ " " $\text{wf}_{trm} (\text{Fun } f \text{ } T)$ "
    using Decompose.premis wf_trm_subterm t(2) t_wf unfolding tfrset_def by auto
  hence " $\forall t \in \text{trms}_{est} A2. \text{wf}_{trm} t$ " by (metis * SMP.MP SMP_wf_trm)
  hence " $\forall t \in (\bigcup (\text{trms}_{st} ' S2)) \cup (\text{trms}_{est} A2). \text{wf}_{trm} t$ "
    using Decompose.premis Decompose.hyps(2) unfolding tfrset_def by force
  ultimately show ?thesis using Decompose.premis unfolding tfrset_def by presburger
qed (metis trmsst_updatest-eq_snd, metis trmsst_updatest-eq_rcv,
    metis trmsst_updatest-eq_eq, metis trmsst_updatest-eq_ineq)
qed metis

private lemma pts_symbolic_c_preserves_tfrstp:
  assumes "(S, A)  $\Rightarrow_c^*$  (S', A)" " $\forall S \in S \cup \{\text{to\_st } A\}. \text{list\_all } \text{tfr}_{stp} S$ "
  shows " $\forall S \in S' \cup \{\text{to\_st } A'\}. \text{list\_all } \text{tfr}_{stp} S$ "
using assms
proof (induction rule: rtrancp_induct2)
  case (step S1 A1 S2 A2)
  from step.hyps(2) step.IH[OF step.premis] show ?case
  proof (induction rule: pts_symbolic_c_induct)
    case Nil
    have 1: " $\forall S \in \{\text{to\_st } A2\}. \text{list\_all } \text{tfr}_{stp} S$ " using Nil by simp
    have 2: " $S2 = S1 - \{[]\}$ " " $\forall S \in S1. \text{list\_all } \text{tfr}_{stp} S$ " using Nil by simp_all
    have " $\forall S \in S2. \text{list\_all } \text{tfr}_{stp} S$ "
    proof
      fix S assume "S  $\in$  S2"
      hence "S  $\in$  S1" using 2(1) by simp
      thus "list_all tfrstp S" using 2(2) by simp
    qed
    thus ?case using 1 by auto
  next
    case (Send t S)
    have 1: " $\forall S \in \{\text{to\_st } A2\}. \text{list\_all } \text{tfr}_{stp} S$ " using Send by (simp add: to_st_append)
    have 2: " $S2 = \text{insert } S (S1 - \{\text{send}(t)_{st}\#S\})$ " " $\forall S \in S1. \text{list\_all } \text{tfr}_{stp} S$ " using Send by
simp_all
    have 3: " $\forall S \in S2. \text{list\_all } \text{tfr}_{stp} S$ "
    proof
      fix S' assume "S'  $\in$  S2"
      hence "S'  $\in$  S1  $\vee$  S' = S" using 2(1) by auto
      moreover have "list_all tfrstp S" using Send.hyps 2(2) by auto
      ultimately show "list_all tfrstp S'" using 2(2) by blast
    qed
    thus ?case using 1 by auto
  next
    case (Receive t S)
    have 1: " $\forall S \in \{\text{to\_st } A2\}. \text{list\_all } \text{tfr}_{stp} S$ " using Receive by (simp add: to_st_append)
    have 2: " $S2 = \text{insert } S (S1 - \{\text{receive}(t)_{st}\#S\})$ " " $\forall S \in S1. \text{list\_all } \text{tfr}_{stp} S$ "
    using Receive by simp_all
    have 3: " $\forall S \in S2. \text{list\_all } \text{tfr}_{stp} S$ "
    proof
      fix S' assume "S'  $\in$  S2"
      hence "S'  $\in$  S1  $\vee$  S' = S" using 2(1) by auto
      moreover have "list_all tfrstp S" using Receive.hyps 2(2) by auto
      ultimately show "list_all tfrstp S'" using 2(2) by blast
    qed
    show ?case using 1 3 by auto
  next
    case (Equality a t t' S)
    have 1: " $\text{to\_st } A2 = \text{to\_st } A1 @ [\langle a: t \doteq t' \rangle_{st}]$ " "list_all tfrstp (to_st A1)"
    using Equality by (simp_all add: to_st_append)
    have 2: "list_all tfrstp [ $\langle a: t \doteq t' \rangle_{st}$ ]" using Equality by fastforce

```

```

have 3: "list_all tfr_stp (to_st A2)"
  using tfr_stp_all_append[of "to_st A1" "[⟨a: t ≐ t'⟩_st]" ] 1 2 by metis
hence 4: "∀S ∈ {to_st A2}. list_all tfr_stp S" using Equality by simp
have 5: "S2 = insert S (S1 - {⟨a: t ≐ t'⟩_st # S})" "∀S ∈ S1. list_all tfr_stp S"
  using Equality by simp_all
have 6: "∀S ∈ S2. list_all tfr_stp S"
proof
  fix S' assume "S' ∈ S2"
  hence "S' ∈ S1 ∨ S' = S" using 5(1) by auto
  moreover have "list_all tfr_stp S" using Equality.hyps 5(2) by auto
  ultimately show "list_all tfr_stp S'" using 5(2) by blast
qed
thus ?case using 4 by auto
next
case (Inequality X F S)
have 1: "to_st A2 = to_st A1@[∀X⟨≠: F⟩_st]" "list_all tfr_stp (to_st A1)"
  using Inequality by (simp_all add: to_st_append)
have "list_all tfr_stp (∀X⟨≠: F⟩_st # S)" using Inequality(1,4) by blast
hence 2: "list_all tfr_stp [∀X⟨≠: F⟩_st]" by simp
have 3: "list_all tfr_stp (to_st A2)"
  using tfr_stp_all_append[of "to_st A1" "[∀X⟨≠: F⟩_st]" ] 1 2 by metis
hence 4: "∀S ∈ {to_st A2}. list_all tfr_stp S" using Inequality by simp
have 5: "S2 = insert S (S1 - {∀X⟨≠: F⟩_st # S})" "∀S ∈ S1. list_all tfr_stp S"
  using Inequality by simp_all
have 6: "∀S ∈ S2. list_all tfr_stp S"
proof
  fix S' assume "S' ∈ S2"
  hence "S' ∈ S1 ∨ S' = S" using 5(1) by auto
  moreover have "list_all tfr_stp S" using Inequality.hyps 5(2) by auto
  ultimately show "list_all tfr_stp S'" using 5(2) by blast
qed
thus ?case using 4 by auto
next
case (Decompose f T)
hence 1: "∀S ∈ S2. list_all tfr_stp S" by blast
have 2: "list_all tfr_stp (to_st A1)" "list_all tfr_stp (to_st [Decomp (Fun f T)])"
  using Decompose.prem1 decomp_tfr_stp by auto
hence "list_all tfr_stp (to_st A1@to_st [Decomp (Fun f T)])" by auto
hence "list_all tfr_stp (to_st A2)"
  using Decompose.hyps(3) to_st_append[of A1 "[Decomp (Fun f T)]"]
  by auto
thus ?case using 1 by blast
qed
qed

private lemma pts_symbolic_c_preserves_well_analyzed:
  assumes "(S, A) ⇒•c (S', A')" "well_analyzed A"
  shows "well_analyzed A'"
using assms
proof (induction rule: rtranc1p_induct2)
  case (step S1 A1 S2 A2)
  from step.hyps(2) step.IH[OF step.prem1] show ?case
  proof (induction rule: pts_symbolic_c_induct)
    case Receive thus ?case by (metis well_analyzed_singleton(1) well_analyzed_append)
  next
    case Send thus ?case by (metis well_analyzed_singleton(2) well_analyzed_append)
  next
    case Equality thus ?case by (metis well_analyzed_singleton(3) well_analyzed_append)
  next
    case Inequality thus ?case by (metis well_analyzed_singleton(4) well_analyzed_append)
  next
    case (Decompose f T)
    hence "Fun f T ∈ subtermsset (ikest A1 ∪ assignment_rhsest A1) - (Var'V)" by auto

```

```

    thus ?case by (metis well_analyzed.Decomp Decompose.prem Decompose.hyps(3))
qed simp
qed metis

private lemma pts_symbolic_c_preserves_Ana_invar_subst:
  assumes "(S, A)  $\Rightarrow^{\bullet}_c$  (S', A')"
  and "Ana_invar_subst (
    ( $\bigcup$  (ikst ' dualst ' S)  $\cup$  (ikest A))  $\cup$ 
    ( $\bigcup$  (assignment_rhsst ' S)  $\cup$  (assignment_rhsest A)))"
  shows "Ana_invar_subst (
    ( $\bigcup$  (ikst ' dualst ' S')  $\cup$  (ikest A'))  $\cup$ 
    ( $\bigcup$  (assignment_rhsst ' S')  $\cup$  (assignment_rhsest A')))"
using assms
proof (induction rule: rtrancIp_induct2)
  case (step S1 A1 S2 A2)
  from step.hyps(2) step.IH[OF step.prem] show ?case
  proof (induction rule: pts_symbolic_c_induct)
    case Nil
    hence " $\bigcup$  (ikst ' dualst ' S1) =  $\bigcup$  (ikst ' dualst ' S2)"
      " $\bigcup$  (assignment_rhsst ' S1) =  $\bigcup$  (assignment_rhsst ' S2)"
      by force+
    thus ?case using Nil by metis
  next
    case Send show ?case
      using ikst_updatest_subset_snd[OF Send.hyps]
        Ana_invar_subst_subset[OF Send.prem]
      by (metis Un_mono)
  next
    case Receive show ?case
      using ikst_updatest_subset_rcv[OF Receive.hyps]
        Ana_invar_subst_subset[OF Receive.prem]
      by (metis Un_mono)
  next
    case Equality show ?case
      using ikst_updatest_subset_eq[OF Equality.hyps]
        Ana_invar_subst_subset[OF Equality.prem]
      by (metis Un_mono)
  next
    case Inequality show ?case
      using ikst_updatest_subset_ineq[OF Inequality.hyps]
        Ana_invar_subst_subset[OF Inequality.prem]
      by (metis Un_mono)
  next
    case (Decompose f T)
    let ?X = " $\bigcup$  (assignment_rhsst ' S2)  $\cup$  assignment_rhsest A2"
    let ?Y = " $\bigcup$  (assignment_rhsst ' S1)  $\cup$  assignment_rhsest A1"
    obtain K M where Ana: "Ana (Fun f T) = (K, M)" by maura
    hence *: "ikest A2 = ikest A1  $\cup$  set M" "assignment_rhsest A2 = assignment_rhsest A1"
      using ikest_append assignment_rhsest_append decomp_ik
        decomp_assignment_rhs_empty Decompose.hyps(3)
      by auto
    { fix g S assume "Fun g S  $\in$  subtermsset ( $\bigcup$  (ikst ' dualst ' S2)  $\cup$  ikest A2  $\cup$  ?X)"
      hence "Fun g S  $\in$  subtermsset ( $\bigcup$  (ikst ' dualst ' S1)  $\cup$  ikest A1  $\cup$  set M  $\cup$  ?X)"
        using * Decompose.hyps(2) by auto
      hence "Fun g S  $\in$  subtermsset ( $\bigcup$  (ikst ' dualst ' S1))"
         $\vee$  Fun g S  $\in$  subtermsset (ikest A1)
         $\vee$  Fun g S  $\in$  subtermsset (set M)
         $\vee$  Fun g S  $\in$  subtermsset ( $\bigcup$  (assignment_rhsst ' S1))
         $\vee$  Fun g S  $\in$  subtermsset (assignment_rhsest A1)"
        using Decompose * Ana_fun_subterm[OF Ana] by auto
    moreover have "Fun f T  $\in$  subtermsset (ikest A1  $\cup$  assignment_rhsest A1)"
      using trmsest_ik_subtermsI Decompose.hyps(1) by auto
    hence "subterms (Fun f T)  $\subseteq$  subtermsset (ikest A1  $\cup$  assignment_rhsest A1)"

```

```

    by (metis in_subterms_subset_Union)
  hence "subtermsset (set M)  $\subseteq$  subtermsset (ikest A1  $\cup$  assignment_rhsest A1)"
    by (meson Un_upper2 Ana_subterm[OF Ana] subterms_subset_set psubsetE subset_trans)
  ultimately have "Fun g S  $\in$  subtermsset ( $\bigcup$  (ikst 'dualst ' S1)  $\cup$  ikest A1  $\cup$  ?Y)"
    by auto
}
thus ?case using Decompose unfolding Ana_invar_subst_def by metis
qed
qed

private lemma pts_symbolic_c_preserves_constr_disj_vars:
  assumes "(S, A)  $\Rightarrow_c^*$  (S', A')" "wfsts ' S A" "fvest A  $\cap$  bvarsest A = {}"
  shows "fvest A'  $\cap$  bvarsest A' = {}"
using assms
proof (induction rule: rtranc1p_induct2)
  case (step S1 A1 S2 A2)
  have *: " $\bigwedge S. S \in S1 \Rightarrow$  fvst S  $\cap$  bvarsest A1 = {}" " $\bigwedge S. S \in S1 \Rightarrow$  fvest A1  $\cap$  bvarsst S = {}"
    using pts_symbolic_c_preserves_wf_prot[OF step.hyps(1) step.prem(1)]
    unfolding wfsts'_def by auto
  from step.hyps(2) step.IH[OF step.prem(2)]
  show ?case
  proof (induction rule: pts_symbolic_c_induct)
    case Nil thus ?case by auto
  next
    case (Send t S)
    hence "fvest A2 = fvest A1  $\cup$  fv t" "bvarsest A2 = bvarsest A1"
      "fvst (send<t>st#S) = fv t  $\cup$  fvst S"
      using fvest_append bvarsest_append by simp+
    thus ?case using *(1)[OF Send(1)] Send(4) by auto
  next
    case (Receive t S)
    hence "fvest A2 = fvest A1  $\cup$  fv t" "bvarsest A2 = bvarsest A1"
      "fvst (receive<t>st#S) = fv t  $\cup$  fvst S"
      using fvest_append bvarsest_append by simp+
    thus ?case using *(1)[OF Receive(1)] Receive(4) by auto
  next
    case (Equality a t t' S)
    hence "fvest A2 = fvest A1  $\cup$  fv t  $\cup$  fv t'" "bvarsest A2 = bvarsest A1"
      "fvst (<a: t  $\doteq$  t'>st#S) = fv t  $\cup$  fv t'  $\cup$  fvst S"
      using fvest_append bvarsest_append by fastforce+
    thus ?case using *(1)[OF Equality(1)] Equality(4) by auto
  next
    case (Inequality X F S)
    hence "fvest A2 = fvest A1  $\cup$  (fvpairs F - set X)" "bvarsest A2 = bvarsest A1  $\cup$  set X"
      "fvst ( $\forall X \langle \forall \neq: F \rangle_{st}\#S$ ) = (fvpairs F - set X)  $\cup$  fvst S"
      using fvest_append bvarsest_append strand_vars_split(3)[of " $\forall X \langle \forall \neq: F \rangle_{st}$ "] S]
      by auto+
    moreover have "fvest A1  $\cap$  set X = {}" using *(2)[OF Inequality(1)] by auto
    ultimately show ?case using *(1)[OF Inequality(1)] Inequality(4) by auto
  next
    case (Decompose f T)
    thus ?case
      using Decompose(3,4) bvars_decomp ik_assignment_rhs_decomp_fv[OF Decompose(1)] by auto
  qed
qed

```

### Theorem: The Typing Result Lifted to the Transition System Level

```

private lemma wfsts'_decomp_rm:
  assumes "well_analyzed A" "wfsts ' S (decomp_rmest A)" shows "wfsts ' S A"
unfolding wfsts'_def
proof (intro conjI)
  show " $\forall S \in S. wf_{st} (wf_{restrictedvars}_{est} A) (dual_{st} S)"$ "

```

```

by (metis (no_types) assms(2) wfsts'_def wfrestrictedvarsest_decomp_rmest_subset
    wf_vars_mono le_iff_sup)

show "∀Sa∈S. ∀S'∈S. fvst Sa ∩ bvarsst S' = {}" by (metis assms(2) wfsts'_def)

show "∀S∈S. fvst S ∩ bvarsest A = {}" by (metis assms(2) wfsts'_def bvars_decomp_rm)

show "∀S∈S. fvest A ∩ bvarsst S = {}" by (metis assms wfsts'_def well_analyzed_decomp_rmest_fv)
qed

private lemma decompest_pts_symbolic_c:
  assumes "D ∈ decompest (ikest A) (assignment_rhsest A) I"
  shows "(S,A) ⇒•c (S,A@D)"
using assms(1)
proof (induction D rule: decompest.induct)
  case (Decomp B f X K T)
  have "subtermsset (ikest A ∪ assignment_rhsest A) ⊆
    subtermsset (ikest (A@B) ∪ assignment_rhsest (A@B))"
  using ikest_append[of A B] assignment_rhsest_append[of A B]
  by auto
  hence "Fun f X ∈ subtermsset (ikest (A@B) ∪ assignment_rhsest (A@B))" using Decomp.hyps by auto
  hence "(S,A@B) ⇒•c (S,A@B@[Decomp (Fun f X)])"
  using pts_symbolic_c.Decompose[of f X "A@B"]
  by simp
  thus ?case
  using Decomp.IH rtrancl_into_rtrancl
    rtranclp_rtrancl_eq[of pts_symbolic_c "(S,A)" "(S,A@B)"]
  by auto
qed simp

private lemma pts_symbolic_to_pts_symbolic_c:
  assumes "(S,tost (decomp_rmest Ad)) ⇒• (S',A') "semest_d {} I (toest A') "semest_c {} I
  Ad"
  and wf: "wfsts' S (decomp_rmest Ad) "wfest {} Ad"
  and tar: "Ana_invar_subst ((⋃ (ikst' dualst' S) ∪ (ikest Ad))
    ∪ (⋃ (assignment_rhsst' S) ∪ (assignment_rhsest Ad)))"
  and wa: "well_analyzed Ad"
  and I: "interpretationsubst I"
  shows "∃Ad'. A' = tost (decomp_rmest Ad') ∧ (S,Ad) ⇒•c (S',Ad') ∧ semest_c {} I Ad'"
using assms(1,2)
proof (induction rule: rtranclp_induct2)
  case refl thus ?case using assms by auto
next
  case (step S1 A1 S2 A2)
  have "semest_d {} I (toest A1)" using step.hyps(2) step.premis
  by (induct rule: pts_symbolic_induct, metis, (metis semest_d_split_left toest_append)+)
  then obtain A1d where
    A1d: "A1 = tost (decomp_rmest A1d)" "(S, Ad) ⇒•c (S1, A1d)" "semest_c {} I A1d"
  using step.IH by moura

  show ?case using step.hyps(2)
proof (induction rule: pts_symbolic_induct)
  case Nil
  hence "(S, Ad) ⇒•c (S2, A1d)" using A1d pts_symbolic_c.Nil[OF Nil.hyps(1), of A1d] by simp
  thus ?case using A1d Nil by auto
next
  case (Send t S)
  hence "semest_c {} I (A1d@[Step (receive⟨t⟩st)])" using semest_c.Receive[OF A1d(3)] by simp
  moreover have "(S1, A1d) ⇒•c (S2, A1d@[Step (receive⟨t⟩st)])"
  using Send.hyps(2) pts_symbolic_c.Send[OF Send.hyps(1), of A1d] by simp
  moreover have "tost (decomp_rmest (A1d@[Step (receive⟨t⟩st)])) = A2"
  using Send.hyps(3) decomp_rmest_append A1d(1) by (simp add: tost_append)
  ultimately show ?case using A1d(2) by auto

```

```

next
  case (Equality a t t' S)
  hence "t · I = t' · I"
    using step.premss semest_d_eq_sem_st[of "{}" I "to_est A2"]
    to_st_append to_est_append to_st_to_est_inv
  by auto
  hence "semest_c {} I (A1d@[Step ((a: t ÷ t')st)])" using semest_c.Equality[OF A1d(3)] by simp
  moreover have "(S1, A1d) ⇒•c (S2, A1d@[Step ((a: t ÷ t')st)])"
    using Equality.hyps(2) pts_symbolic_c.Equality[OF Equality.hyps(1), of A1d] by simp
  moreover have "to_st (decomprmest (A1d@[Step ((a: t ÷ t')st)])) = A2"
    using Equality.hyps(3) decomprmest_append A1d(1) by (simp add: to_st_append)
  ultimately show ?case using A1d(2) by auto
next
  case (Inequality X F S)
  hence "ineq_model I X F"
    using step.premss semest_d_eq_sem_st[of "{}" I "to_est A2"]
    to_st_append to_est_append to_st_to_est_inv
  by auto
  hence "semest_c {} I (A1d@[Step (∀X(∀≠: F)st)])" using semest_c.Inequality[OF A1d(3)] by simp
  moreover have "(S1, A1d) ⇒•c (S2, A1d@[Step (∀X(∀≠: F)st)])"
    using Inequality.hyps(2) pts_symbolic_c.Inequality[OF Inequality.hyps(1), of A1d] by simp
  moreover have "to_st (decomprmest (A1d@[Step (∀X(∀≠: F)st)])) = A2"
    using Inequality.hyps(3) decomprmest_append A1d(1) by (simp add: to_st_append)
  ultimately show ?case using A1d(2) by auto
next
  case (Receive t S)
  hence "ikst A1 ·set I ⊢ t · I"
    using step.premss semest_d_eq_sem_st[of "{}" I "to_est A2"]
    strand_sem_split(4)[of "{}" A1 "[send(t)st]" I]
    to_st_append to_est_append to_st_to_est_inv
  by auto
  moreover have "ikst A1 ·set I ⊆ ikest A1d ·set I" using A1d(1) decomprmest_ik_subset by auto
  ultimately have *: "ikest A1d ·set I ⊢ t · I" using ideduct_mono by auto

  have "wfsts' S Ad" by (rule wfsts'_decomprm[OF wa_assms(4)])
  hence **: "wfest {} A1d" by (rule pts_symbolic_c.preserves_wf_is[OF A1d(2) _ assms(5)])

  have "Ana_invar_subst (⋃ (ikst'dualst'S1) ∪ (ikest A1d) ∪
    (⋃ (assignment_rhsst'S1) ∪ (assignment_rhsest A1d)))"
    using tar A1d(2) pts_symbolic_c.preserves_Ana_invar_subst by metis
  hence "Ana_invar_subst (ikest A1d)" "Ana_invar_subst (assignment_rhsest A1d)"
    using Ana_invar_subst_subset by blast+
  moreover have "well_analyzed A1d"
    using pts_symbolic_c.preserves_well_analyzed[OF A1d(2) wa] by metis
  ultimately obtain D where D:
    "D ∈ decompest (ikest A1d) (assignment_rhsest A1d) I"
    "ikest (A1d@D) ·set I ⊢c t · I"
    using decompest_exist_subst[OF * A1d(3) ** assms(8)] unfolding Ana_invar_subst_def by auto

  have "(S, Ad) ⇒•c* (S1, A1d@D)" using A1d(2) decompest_pts_symbolic_c[OF D(1), of S1] by
auto
  hence "(S, Ad) ⇒•c* (S2, A1d@D@[Step (send(t)st)])"
    using Receive(2) pts_symbolic_c.Receive[OF Receive.hyps(1), of "A1d@D"] by auto
  moreover have "A2 = to_st (decomprmest (A1d@D@[Step (send(t)st)]))"
    using Receive.hyps(3) A1d(1) decompest_decomprmest_empty[OF D(1)]
    decomprmest_append to_st_append
  by auto
  moreover have "semest_c {} I (A1d@D@[Step (send(t)st)])"
    using D(2) semest_c.Send[OF semest_c.decompest_append[OF A1d(3) D(1)]] by simp
  ultimately show ?case by auto
qed
qed

```

```

private lemma pts_symbolic_c_to_pts_symbolic:
  assumes "(S, A)  $\Rightarrow_c^*$  (S', A')" "sem_est_c {} I A'"
  shows "(S, to_st (decomp_rm_est A))  $\Rightarrow^{**}$  (S', to_st (decomp_rm_est A'))"
    "sem_est_d {} I (decomp_rm_est A')"
proof -
  show "(S, to_st (decomp_rm_est A))  $\Rightarrow^{**}$  (S', to_st (decomp_rm_est A'))" using assms(1)
proof (induction rule: rtrancIp_induct2)
  case (step S1 A1 S2 A2) show ?case using step.hyps(2,1) step.IH
  proof (induction rule: pts_symbolic_c_induct)
    case Nil thus ?case
      using pts_symbolic.Nil[OF Nil.hyps(1), of "to_st (decomp_rm_est A1)"] by simp
    next
    case (Send t S) thus ?case
      using pts_symbolic.Send[OF Send.hyps(1), of "to_st (decomp_rm_est A1)"]
      by (simp add: decomp_rm_est_append to_st_append)
    next
    case (Receive t S) thus ?case
      using pts_symbolic.Receive[OF Receive.hyps(1), of "to_st (decomp_rm_est A1)"]
      by (simp add: decomp_rm_est_append to_st_append)
    next
    case (Equality a t t' S) thus ?case
      using pts_symbolic.Equality[OF Equality.hyps(1), of "to_st (decomp_rm_est A1)"]
      by (simp add: decomp_rm_est_append to_st_append)
    next
    case (Inequality t t' S) thus ?case
      using pts_symbolic.Inequality[OF Inequality.hyps(1), of "to_st (decomp_rm_est A1)"]
      by (simp add: decomp_rm_est_append to_st_append)
    next
    case (Decompose t) thus ?case using decomp_rm_est_append by simp
  qed
qed simp
qed (rule sem_est_d_decomp_rm_est_if_sem_est_c[OF assms(2)])

private lemma pts_symbolic_to_pts_symbolic_c_from_initial:
  assumes "(S0, [])  $\Rightarrow_c^*$  (S, A)" "I  $\models \langle A \rangle$ " "wfsts' S0 []"
  and "Ana_invar_subst (⋃ (ikst ' dualst ' S0) ∪ ⋃ (assignment_rhsst ' S0))" "interpretationsubst I"
  shows "∃ Ad. A = to_st (decomp_rm_est Ad) ∧ (S0, [])  $\Rightarrow_c^*$  (S, Ad) ∧ (I  $\models_c \langle to\_st A_d \rangle$ )"
using assms pts_symbolic_to_pts_symbolic_c[of S0 "[]" S A I]
  sem_est_c_eq_sem_st[of "{}" I] sem_est_d_eq_sem_st[of "{}" I]
  to_st_to_est_inv[of A] strand_sem_eq_defs
by (auto simp add: constr_sem_c_def constr_sem_d_def simp del: subst_range.simps)

private lemma pts_symbolic_c_to_pts_symbolic_from_initial:
  assumes "(S0, [])  $\Rightarrow_c^*$  (S, A)" "I  $\models_c \langle to\_st A \rangle$ "
  shows "(S0, [])  $\Rightarrow^{**}$  (S, to_st (decomp_rm_est A))" "I  $\models \langle to\_st (decomp\_rm\_est A) \rangle$ "
using assms pts_symbolic_c_to_pts_symbolic[of S0 "[]" S A I]
  sem_est_c_eq_sem_st[of "{}" I] sem_est_d_eq_sem_st[of "{}" I] strand_sem_eq_defs
by (auto simp add: constr_sem_c_def constr_sem_d_def)

private lemma to_st_trms_wf:
  assumes "wftrms (trmsest A)"
  shows "wftrms (trmsst (to_st A))"
using assms
proof (induction A)
  case (Cons x A)
  hence IH: "∀ t ∈ trmsst (to_st A). wftrm t" by auto
  with Cons show ?case
  proof (cases x)
    case (Decomp t)
    hence "wftrm t" using Cons.prem by auto
    obtain K T where Ana_t: "Ana t = (K, T)" by maura
    hence "trmsst (decomp t) ⊆ {t} ∪ set K ∪ set T" using decomp_set_unfold[OF Ana_t] by force
    moreover have "∀ t ∈ set T. wftrm t" using Ana_subterm[OF Ana_t] ⟨wftrm t⟩ wf_trm_subterm by

```

```

auto
  ultimately have "∀t ∈ trmsst (decomp t). wftrms t" using Ana_keys_wf'[OF Ana_t] (wftrms t) by auto
  thus ?thesis using IH Decomp by auto
qed auto
qed simp

private lemma to_st_trms_SMP_subset: "trmsst (to_st A) ⊆ SMP (trmsest A)"
proof
  fix t assume "t ∈ trmsst (to_st A)" thus "t ∈ SMP (trmsest A)"
  proof (induction A)
    case (Cons x A)
    hence *: "t ∈ trmsst (to_st [x]) ∪ trmsst (to_st A)" using to_st_append[of "[x]" A] by auto
    have **: "trmsst (to_st A) ⊆ trmsst (to_st (x#A))" "trmsest A ⊆ trmsest (x#A)"
      using to_st_append[of "[x]" A] by auto
    show ?case
    proof (cases "t ∈ trmsst (to_st A)")
      case True thus ?thesis using Cons.IH SMP_mono[OF **(2)] by auto
    next
      case False
      hence ***: "t ∈ trmsst (to_st [x])" using * by auto
      thus ?thesis
      proof (cases x)
        case (Decomp t')
        hence ****: "t ∈ trmsst (decomp t')" "t' ∈ trmsest (x#A)" using *** by auto
        obtain K T where Ana_t': "Ana t' = (K,T)" by moura
        hence "t ∈ {t'} ∪ set K ∪ set T" using decomp_set_unfold[OF Ana_t'] ****(1) by force
        moreover
        { assume "t = t'" hence ?thesis using SMP.MP[OF ****(2)] by simp }
        moreover
        { assume "t ∈ set K" hence ?thesis using SMP.Ana[OF SMP.MP[OF ****(2)] Ana_t'] by auto }
        moreover
        { assume "t ∈ set T" "t ≠ t'"
          hence "t ⊆ t'" using Ana_subterm[OF Ana_t'] by blast
          hence ?thesis using SMP.Subterm[OF SMP.MP[OF ****(2)]] by auto
        }
        ultimately show ?thesis using Decomp by auto
      qed auto
    qed
  qed simp
qed

private lemma to_st_trms_tfrset:
  assumes "tfrset (trmsest A)"
  shows "tfrset (trmsst (to_st A))"
proof -
  have *: "trmsst (to_st A) ⊆ SMP (trmsest A)"
  using to_st_trms_wf to_st_trms_SMP_subset assms unfolding tfrset_def by auto
  have "trmsst (to_st A) = trmsst (to_st A) ∪ trmsest A" by (blast dest!: trmsestD)
  hence "SMP (trmsest A) = SMP (trmsst (to_st A))" using SMP_subset_union_eq[OF *] by auto
  thus ?thesis using * assms unfolding tfrset_def by presburger
qed

theorem wt_attack_if_tfr_attack_pts:
  assumes "wfsts S0" "tfrset (⋃ (trmsst ' S0))" "wftrms (⋃ (trmsst ' S0))" "∀S ∈ S0. list_all tfrstp S"
  and "Ana_invar_subst (⋃ (ikst ' dualst ' S0) ∪ ⋃ (assignment_rhsst ' S0))"
  and "(S0, []) ⇒* (S, A)" "interpretationsubst I" "I ⊨ ⟨A, Var⟩"
  shows "∃Iτ. interpretationsubst Iτ ∧ (Iτ ⊨ ⟨A, Var⟩) ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ)"
proof -
  have "(⋃ (trmsst ' S0)) ∪ (trmsest []) = ⋃ (trmsst ' S0)" "to_st [] = []" "list_all tfrstp []"
  using assms by simp_all
  hence *: "tfrset ((⋃ (trmsst ' S0)) ∪ (trmsest []))"
    "wftrms ((⋃ (trmsst ' S0)) ∪ (trmsest []))"

```



```

"wfsts' S0 []" "∀S ∈ S0 ∪ {tost []}. list_all tfrstp S"
using assms wfsts-wfsts' by (metis, metis, metis, simp)

obtain Ad where Ad: "A = tost (decomprmest Ad)" "(S0, []) ⇒•c (S, Ad)" "I ⊨c ⟨tost Ad⟩"
  using pts_symbolic_to_pts_symbolic_c_from_initial assms *(3) by metis
hence "tfrset (⋃ (trmsst ' S) ∪ (trmsest Ad))" "wftrms (⋃ (trmsst ' S) ∪ (trmsest Ad))"
  using pts_symbolic_c_preserves_tfrset[OF _ *(1,2)] by blast+
hence "tfrset (trmsest Ad)" "wftrms (trmsest Ad)"
  unfolding tfrset-def by (metis DiffE DiffI SMP_union UnCI, metis UnCI)
hence "tfrset (trmsst (tost Ad))" "wftrms (trmsst (tost Ad))"
  by (metis tost-trms-tfrset, metis tost-trms-wf)
moreover have "wfconstr (tost Ad) Var"
proof -
  have "wtsubst Var" "wftrms (substrange Var)" "substdomain Var ∩ varsest Ad = {}"
    "range_vars Var ∩ bvarsest Ad = {}"
    by (simp_all add: range_vars_alt_def)
  moreover have "wfest {} Ad"
    using pts_symbolic_c_preserves_wf_is[OF Ad(2) *(3), of "{}"]
    by auto
  moreover have "fvst (tost Ad) ∩ bvarsest Ad = {}"
    using pts_symbolic_c_preserves_constr_disj_vars[OF Ad(2)] assms(1) wfsts-wfsts'
    by fastforce
  ultimately show ?thesis unfolding wfconstr-def wfsubst-def by simp
qed
moreover have "list_all tfrstp (tost Ad)"
  using pts_symbolic_c_preserves_tfrstp[OF Ad(2) *(4)] by blast
moreover have "wtsubst Var" "wftrms (substrange Var)" by simp_all
ultimately obtain Iτ where Iτ:
  "interpretationsubst Iτ" "Iτ ⊨c ⟨tost Ad, Var⟩" "wtsubst Iτ" "wftrms (substrange Iτ)"
  using wt_attack_if_tfr_attack[OF assms(7) Ad(3)]
    ⟨tfrset (trmsst (tost Ad))⟩ ⟨list_all tfrstp (tost Ad)⟩
  unfolding tfrst-def by metis
hence "Iτ ⊨ ⟨A, Var⟩" using pts_symbolic_c_to_pts_symbolic_from_initial Ad by metis
thus ?thesis using Iτ(1,3,4) by metis
qed

```

### Corollary: The Typing Result on the Level of Constraints

There exists well-typed models of satisfiable type-flaw resistant constraints

corollary wt\_attack\_if\_tfr\_attack\_d:

```

assumes "wfst {} A" "fvst A ∩ bvarsst A = {}" "tfrst A" "wftrms (trmsst A)"
and "Ana_invar_subst (ikst A ∪ assignment_rhsst A)"
and "interpretationsubst I" "I ⊨ ⟨A⟩"
shows "∃Iτ. interpretationsubst Iτ ∧ (Iτ ⊨ ⟨A⟩) ∧ wtsubst Iτ ∧ wftrms (substrange Iτ)"

```

proof -

```

{ fix S A have "({S}, A) ⇒• ({}, A@dualst S)"
  proof (induction S arbitrary: A)
    case Nil thus ?case using pts_symbolic.Nil[of "{}"] by auto
  next
    case (Cons x S)
    hence "({S}, A@dualst [x]) ⇒• ({}, A@dualst (x#S))"
      by (metis dualst-append List.append_assoc List.append_Nil List.append_Cons)
    moreover have "({x#S}, A) ⇒• ({S}, A@dualst [x])"
      using pts_symbolic.Send[of _ S "{x#S}"] pts_symbolic.Receive[of _ S "{x#S}"]
        pts_symbolic.Equality[of _ _ S "{x#S}"] pts_symbolic.Inequality[of _ _ S "{x#S}"]
      by (cases x) auto
    ultimately show ?case by simp
  qed
}
hence 0: "({dualst A}, []) ⇒• ({}, A)" using dualst-self_inverse by (metis List.append_Nil)

```

```

have "fvst (dualst A) ∩ bvarsst (dualst A) = {}" using assms(2) dualst-fv dualst-bvars by metis+
hence 1: "wfsts {dualst A}" using assms(1,2) dualst-self_inverse[of A] unfolding wfsts-def by auto

```

```

have "⋃ (trmsst ' {A} ) = trmsst A" "⋃ (trmsst ' {dualst A} ) = trmsst (dualst A)" by auto
hence "tfrset (⋃ (trmsst ' {A} ))" "wftrms (⋃ (trmsst ' {A} ))"
      "(⋃ (trmsst ' {A} )) = ⋃ (trmsst ' {dualst A} )"
      using assms(3,4) unfolding tfrst-def
      by (metis, metis, metis dualst-trms-eq)
hence 2: "tfrset (⋃ (trmsst ' {dualst A} ))" and 3: "wftrms (⋃ (trmsst ' {dualst A} ))" by metis+

have 4: "∀ S ∈ {dualst A}. list_all tfrstp S"
      using dualst-tfrstp assms(3) unfolding tfrst-def by blast

have "assignment_rhsst A = assignment_rhsst (dualst A)"
      by (induct A rule: assignment_rhsst.induct) auto
hence 5: "Ana_invar_subst (⋃ (ikst 'dualst ' {dualst A} ) ∪ ⋃ (assignment_rhsst ' {dualst A} ))"
      using assms(5) dualst-self-inverse[of A] by auto

show ?thesis by (rule wt_attack_if_tfr_attack_pts[OF 1 2 3 4 5 0 assms(6,7)])
qed

end

end

end

```

## 4 The Typing Result for Stateful Protocols

In this chapter, we lift the typing result to stateful protocols. For more details, we refer the reader to [3] and [1, chapter 4].

### 4.1 Stateful Strands (Stateful\_Strands)

```
theory Stateful_Strands
imports Strands_and_Constraints
begin
```

#### 4.1.1 Stateful Constraints

```
datatype (funs sstp: 'a, varssstp: 'b) stateful_strand_step =
  Send (the_msg: "('a,'b) term") ("send⟨_⟩" 80)
| Receive (the_msg: "('a,'b) term") ("receive⟨_⟩" 80)
| Equality (the_check: poscheckvariant) (the_lhs: "('a,'b) term") (the_rhs: "('a,'b) term")
  ("⟨_ : _ ≐ _⟩" [80,80])
| Insert (the_elem_term: "('a,'b) term") (the_set_term: "('a,'b) term") ("insert⟨_,_⟩" 80)
| Delete (the_elem_term: "('a,'b) term") (the_set_term: "('a,'b) term") ("delete⟨_,_⟩" 80)
| InSet (the_check: poscheckvariant) (the_elem_term: "('a,'b) term") (the_set_term: "('a,'b) term")
  ("⟨_ : _ ∈ _⟩" [80,80])
| NegChecks (bvarssstp: "'b list")
  (the_eqs: "((('a,'b) term × ('a,'b) term) list)")
  (the_ins: "((('a,'b) term × ('a,'b) term) list)")
  ("∀_ (∀ ≠: _ ∨ ≠: _)" [80,80])
where
  "bvarssstp (Send _) = []"
| "bvarssstp (Receive _) = []"
| "bvarssstp (Equality _ _ _) = []"
| "bvarssstp (Insert _ _) = []"
| "bvarssstp (Delete _ _) = []"
| "bvarssstp (InSet _ _ _) = []"

type_synonym ('a,'b) stateful_strand = "('a,'b) stateful_strand_step list"
type_synonym ('a,'b) dbstatelist = "((('a,'b) term × ('a,'b) term) list)"
type_synonym ('a,'b) dbstate = "((('a,'b) term × ('a,'b) term) set)"

abbreviation
  "is_Assignment x ≡ (is_Equality x ∨ is_InSet x) ∧ the_check x = Assign"

abbreviation
  "is_Check x ≡ ((is_Equality x ∨ is_InSet x) ∧ the_check x = Check) ∨ is_NegChecks x"

abbreviation
  "is_Update x ≡ is_Insert x ∨ is_Delete x"

abbreviation InSet_select ("select⟨_,_⟩") where "select⟨t,s⟩ ≡ InSet Assign t s"
abbreviation InSet_check ("⟨_ in _⟩") where "⟨t in s⟩ ≡ InSet Check t s"
abbreviation Equality_assign ("⟨_ := _⟩") where "⟨t := s⟩ ≡ Equality Assign t s"
abbreviation Equality_check ("⟨_ == _⟩") where "⟨t == s⟩ ≡ Equality Check t s"

abbreviation NegChecks_Inequality1 ("⟨_ != _⟩") where
  "⟨t != s⟩ ≡ NegChecks [] [(t,s)] []"

abbreviation NegChecks_Inequality2 ("∀_ ⟨_ != _⟩") where
```

" $\forall x \langle t \neq s \rangle \equiv \text{NegChecks } [x] [(t,s)] []$ "

abbreviation *NegChecks\_Inequality3* (" $\forall \_,\_ \langle \_ \neq \_ \rangle$ ") where

" $\forall x,y \langle t \neq s \rangle \equiv \text{NegChecks } [x,y] [(t,s)] []$ "

abbreviation *NegChecks\_Inequality4* (" $\forall \_,\_ \langle \_ \neq \_ \rangle$ ") where

" $\forall x,y,z \langle t \neq s \rangle \equiv \text{NegChecks } [x,y,z] [(t,s)] []$ "

abbreviation *NegChecks\_NotInSet1* (" $\langle \_ \text{ not in } \_ \rangle$ ") where

" $\langle t \text{ not in } s \rangle \equiv \text{NegChecks } [] [] [(t,s)]$ "

abbreviation *NegChecks\_NotInSet2* (" $\forall \_ \langle \_ \text{ not in } \_ \rangle$ ") where

" $\forall x \langle t \text{ not in } s \rangle \equiv \text{NegChecks } [x] [] [(t,s)]$ "

abbreviation *NegChecks\_NotInSet3* (" $\forall \_,\_ \langle \_ \text{ not in } \_ \rangle$ ") where

" $\forall x,y \langle t \text{ not in } s \rangle \equiv \text{NegChecks } [x,y] [] [(t,s)]$ "

abbreviation *NegChecks\_NotInSet4* (" $\forall \_,\_ \langle \_ \text{ not in } \_ \rangle$ ") where

" $\forall x,y,z \langle t \text{ not in } s \rangle \equiv \text{NegChecks } [x,y,z] [] [(t,s)]$ "

fun *trms\_sstp* where

"*trms\_sstp* (Send *t*) = {*t*}"  
 | "*trms\_sstp* (Receive *t*) = {*t*}"  
 | "*trms\_sstp* (Equality *\_ t t'*) = {*t, t'*}"  
 | "*trms\_sstp* (Insert *t t'*) = {*t, t'*}"  
 | "*trms\_sstp* (Delete *t t'*) = {*t, t'*}"  
 | "*trms\_sstp* (InSet *\_ t t'*) = {*t, t'*}"  
 | "*trms\_sstp* (NegChecks *\_ F F'*) = *trms\_pairs F*  $\cup$  *trms\_pairs F'*"

definition *trms\_sst* where "*trms\_sst S*  $\equiv \bigcup (\text{trms\_sstp } ' \text{ set } S)$ "

declare *trms\_sst\_def*[simp]

fun *trms\_list\_sstp* where

"*trms\_list\_sstp* (Send *t*) = [*t*]"  
 | "*trms\_list\_sstp* (Receive *t*) = [*t*]"  
 | "*trms\_list\_sstp* (Equality *\_ t t'*) = [*t, t'*]"  
 | "*trms\_list\_sstp* (Insert *t t'*) = [*t, t'*]"  
 | "*trms\_list\_sstp* (Delete *t t'*) = [*t, t'*]"  
 | "*trms\_list\_sstp* (InSet *\_ t t'*) = [*t, t'*]"  
 | "*trms\_list\_sstp* (NegChecks *\_ F F'*) = concat (map ( $\lambda(t,t'). [t,t']$ ) (*F@F'*))"

definition *trms\_list\_sst* where "*trms\_list\_sst S*  $\equiv \text{remdups } (\text{concat } (\text{map } \text{trms\_list\_sstp } S))$ "

definition *ik\_sst* where "*ik\_sst A*  $\equiv \{t. \text{Receive } t \in \text{set } A\}$ "

definition *bvars\_sst* :: "('a, 'b) stateful\_strand  $\Rightarrow$  'b set" where

"*bvars\_sst S*  $\equiv \bigcup (\text{set } (\text{map } (\text{set } \circ \text{bvars\_sstp}) S))$ "

fun *fv\_sstp* :: "('a, 'b) stateful\_strand\_step  $\Rightarrow$  'b set" where

"*fv\_sstp* (Send *t*) = *fv t*"  
 | "*fv\_sstp* (Receive *t*) = *fv t*"  
 | "*fv\_sstp* (Equality *\_ t t'*) = *fv t*  $\cup$  *fv t'*"  
 | "*fv\_sstp* (Insert *t t'*) = *fv t*  $\cup$  *fv t'*"  
 | "*fv\_sstp* (Delete *t t'*) = *fv t*  $\cup$  *fv t'*"  
 | "*fv\_sstp* (InSet *\_ t t'*) = *fv t*  $\cup$  *fv t'*"  
 | "*fv\_sstp* (NegChecks *X F F'*) = *fv\_pairs F*  $\cup$  *fv\_pairs F'* - set *X*"

definition *fv\_sst* :: "('a, 'b) stateful\_strand  $\Rightarrow$  'b set" where

"*fv\_sst S*  $\equiv \bigcup (\text{set } (\text{map } \text{fv\_sstp } S))$ "

fun *fv\_list\_sstp* where

"*fv\_list\_sstp* (send(*t*)) = *fv\_list t*"  
 | "*fv\_list\_sstp* (receive(*t*)) = *fv\_list t*"

```

| "fv_listsstp ( $\langle \_ : t \dot{=} s \rangle$ ) = fv_list t@fv_list s"
| "fv_listsstp (insert $\langle t, s \rangle$ ) = fv_list t@fv_list s"
| "fv_listsstp (delete $\langle t, s \rangle$ ) = fv_list t@fv_list s"
| "fv_listsstp ( $\langle \_ : t \in s \rangle$ ) = fv_list t@fv_list s"
| "fv_listsstp ( $\forall X \langle \forall \neq : F \vee \notin : F' \rangle$ ) = filter ( $\lambda x. x \notin \text{set } X$ ) (fv_listpairs (F@F'))"

```

**definition** fv\_list<sub>sst</sub> where

```
"fv_listsst S  $\equiv$  remdups (concat (map fv_listsstp S))"
```

**declare** bvars<sub>sst-def</sub>[simp]

**declare** fv<sub>sst-def</sub>[simp]

**definition** vars<sub>sst</sub>::("a, 'b) stateful\_strand  $\Rightarrow$  'b set" where

```
"varssst S  $\equiv$   $\bigcup$  (set (map varssstp S))"
```

**abbreviation** wfrestrictedvars<sub>sstp</sub>::("a, 'b) stateful\_strand\_step  $\Rightarrow$  'b set" where

```

"wfrestrictedvarssstp x  $\equiv$ 
  case x of
    NegChecks _ _  $\Rightarrow$  {}
  | Equality Check _ _  $\Rightarrow$  {}
  | InSet Check _ _  $\Rightarrow$  {}
  | Delete _ _  $\Rightarrow$  {}
  | _  $\Rightarrow$  varssstp x"

```

**definition** wfrestrictedvars<sub>sst</sub>::("a, 'b) stateful\_strand  $\Rightarrow$  'b set" where

```
"wfrestrictedvarssst S  $\equiv$   $\bigcup$  (set (map wfrestrictedvarssstp S))"
```

**abbreviation** wfvarsoccs<sub>sstp</sub> where

```

"wfvarsoccssstp x  $\equiv$ 
  case x of
    Send t  $\Rightarrow$  fv t
  | Equality Assign s t  $\Rightarrow$  fv s
  | InSet Assign s t  $\Rightarrow$  fv s  $\cup$  fv t
  | _  $\Rightarrow$  {}"

```

**definition** wfvarsoccs<sub>sst</sub> where

```
"wfvarsoccssst S  $\equiv$   $\bigcup$  (set (map wfvarsoccssstp S))"
```

**fun** wf'<sub>sst</sub>::"b set  $\Rightarrow$  (a, 'b) stateful\_strand  $\Rightarrow$  bool" where

```

"wf'sst V [] = True"
| "wf'sst V (Receive t#S) = (fv t  $\subseteq$  V  $\wedge$  wf'sst V S)"
| "wf'sst V (Send t#S) = wf'sst (V  $\cup$  fv t) S"
| "wf'sst V (Equality Assign t t'#S) = (fv t'  $\subseteq$  V  $\wedge$  wf'sst (V  $\cup$  fv t) S)"
| "wf'sst V (Equality Check _ _#S) = wf'sst V S"
| "wf'sst V (Insert t s#S) = (fv t  $\subseteq$  V  $\wedge$  fv s  $\subseteq$  V  $\wedge$  wf'sst V S)"
| "wf'sst V (Delete _ _#S) = wf'sst V S"
| "wf'sst V (InSet Assign t s#S) = wf'sst (V  $\cup$  fv t  $\cup$  fv s) S"
| "wf'sst V (InSet Check _ _#S) = wf'sst V S"
| "wf'sst V (NegChecks _ _#S) = wf'sst V S"

```

**abbreviation** "wf<sub>sst</sub> S  $\equiv$  wf'<sub>sst</sub> {} S  $\wedge$  fv<sub>sst</sub> S  $\cap$  bvars<sub>sst</sub> S = {}"

**fun** subst\_apply\_stateful\_strand\_step::

```

("a, 'b) stateful_strand_step  $\Rightarrow$  (a, 'b) subst  $\Rightarrow$  (a, 'b) stateful_strand_step"
(infix ".sstp" 51) where
  "send $\langle t \rangle$  .sstp  $\vartheta$  = send $\langle t \cdot \vartheta \rangle$ "
| "receive $\langle t \rangle$  .sstp  $\vartheta$  = receive $\langle t \cdot \vartheta \rangle$ "
| " $\langle a : t \dot{=} s \rangle$  .sstp  $\vartheta$  =  $\langle a : (t \cdot \vartheta) \dot{=} (s \cdot \vartheta) \rangle$ "
| " $\langle a : t \in s \rangle$  .sstp  $\vartheta$  =  $\langle a : (t \cdot \vartheta) \in (s \cdot \vartheta) \rangle$ "
| "insert $\langle t, s \rangle$  .sstp  $\vartheta$  = insert $\langle t \cdot \vartheta, s \cdot \vartheta \rangle$ "
| "delete $\langle t, s \rangle$  .sstp  $\vartheta$  = delete $\langle t \cdot \vartheta, s \cdot \vartheta \rangle$ "
| " $\forall X \langle \forall \neq : F \vee \notin : G \rangle$  .sstp  $\vartheta$  =  $\forall X \langle \forall \neq : (F \cdot_{pairs} \text{rm\_vars (set } X) \vartheta) \vee \notin : (G \cdot_{pairs} \text{rm\_vars (set } X) \vartheta) \rangle$ "

```

**definition** subst\_apply\_stateful\_strand::

```
"('a,'b) stateful_strand  $\Rightarrow$  ('a,'b) subst  $\Rightarrow$  ('a,'b) stateful_strand"
(infix ".sst" 51) where
"S .sst  $\vartheta \equiv \text{map } (\lambda x. x .sstp \vartheta) S"$ 
```

**fun** dbupd<sub>sst</sub>:: "('f,'v) stateful\_strand  $\Rightarrow$  ('f,'v) subst  $\Rightarrow$  ('f,'v) dbstate  $\Rightarrow$  ('f,'v) dbstate"

**where**

```
"dbupdsst [] I D = D"
| "dbupdsst (Insert t s#A) I D = dbupdsst A I (insert ((t,s) .p I) D)"
| "dbupdsst (Delete t s#A) I D = dbupdsst A I (D - {((t,s) .p I)})"
| "dbupdsst (_#A) I D = dbupdsst A I D"
```

**fun** db'<sub>sst</sub>:: "('f,'v) stateful\_strand  $\Rightarrow$  ('f,'v) subst  $\Rightarrow$  ('f,'v) dbstatelist  $\Rightarrow$  ('f,'v) dbstatelist"

**where**

```
"db'sst [] I D = D"
| "db'sst (Insert t s#A) I D = db'sst A I (List.insert ((t,s) .p I) D)"
| "db'sst (Delete t s#A) I D = db'sst A I (List.removeAll ((t,s) .p I) D)"
| "db'sst (_#A) I D = db'sst A I D"
```

**definition** db<sub>sst</sub> **where**

```
"dbsst S I  $\equiv$  db'sst S I []"
```

**fun** setops<sub>sstp</sub> **where**

```
"setopssstp (Insert t s) = {(t,s)}"
| "setopssstp (Delete t s) = {(t,s)}"
| "setopssstp (InSet _ t s) = {(t,s)}"
| "setopssstp (NegChecks _ _ F') = set F'"
| "setopssstp _ = {}"
```

The set-operations of a stateful strand

**definition** setops<sub>sst</sub> **where**

```
"setopssst S  $\equiv \bigcup$  (setopssstp ' set S)"
```

**fun** setops\_list<sub>sstp</sub> **where**

```
"setops_listsstp (Insert t s) = [(t,s)]"
| "setops_listsstp (Delete t s) = [(t,s)]"
| "setops_listsstp (InSet _ t s) = [(t,s)]"
| "setops_listsstp (NegChecks _ _ F') = F'"
| "setops_listsstp _ = []"
```

The set-operations of a stateful strand (list variant)

**definition** setops\_list<sub>sst</sub> **where**

```
"setops_listsst S  $\equiv \text{remdups } (\text{concat } (\text{map } \text{setops\_list\_sstp } S))"$ 
```

### 4.1.2 Small Lemmata

**lemma** trms\_list<sub>sst</sub>\_is\_trms<sub>sst</sub>: "trms<sub>sst</sub> S = set (trms\_list<sub>sst</sub> S)"

**unfolding** trms<sub>st</sub>\_def trms\_list<sub>sst</sub>\_def

**proof** (induction S)

case (Cons x S) thus ?case by (cases x) auto

**qed simp**

**lemma** setops\_list<sub>sst</sub>\_is\_setops<sub>sst</sub>: "setops<sub>sst</sub> S = set (setops\_list<sub>sst</sub> S)"

**unfolding** setops<sub>sst</sub>\_def setops\_list<sub>sst</sub>\_def

**proof** (induction S)

case (Cons x S) thus ?case by (cases x) auto

**qed simp**

**lemma** fv\_list<sub>sstp</sub>\_is\_fv<sub>sstp</sub>: "fv<sub>sstp</sub> a = set (fv\_list<sub>sstp</sub> a)"

**proof** (cases a)

case (NegChecks X F G) thus ?thesis

using fv<sub>pairs</sub>\_append[of F G] fv\_list<sub>pairs\_append</sub>[of F G]

fv\_list<sub>pairs\_is\_fv\_pairs</sub>[of "F@G"]

```

by auto
qed (simp_all add: fv_list_pairs_is_fvpairs fv_list_is_fv)

lemma fv_list_sst_is_fv_sst: "fv_sst S = set (fv_list_sst S)"
unfolding fv_sst_def fv_list_sst_def by (induct S) (simp_all add: fv_list_sstp_is_fv_sstp)

lemma trms_sstp_finite[simp]: "finite (trms_sstp x)"
by (cases x) auto

lemma trms_sst_finite[simp]: "finite (trms_sst S)"
using trms_sstp_finite unfolding trms_sst_def by (induct S) auto

lemma vars_sstp_finite[simp]: "finite (vars_sstp x)"
by (cases x) auto

lemma vars_sst_finite[simp]: "finite (vars_sst S)"
using vars_sstp_finite unfolding vars_sst_def by (induct S) auto

lemma fv_sstp_finite[simp]: "finite (fv_sstp x)"
by (cases x) auto

lemma fv_sst_finite[simp]: "finite (fv_sst S)"
using fv_sstp_finite unfolding fv_sst_def by (induct S) auto

lemma bvars_sstp_finite[simp]: "finite (set (bvars_sstp x))"
by (rule finite_set)

lemma bvars_sst_finite[simp]: "finite (bvars_sst S)"
using bvars_sstp_finite unfolding bvars_sst_def by (induct S) auto

lemma subst_sst_nil[simp]: "[ ] ·sst δ = [ ]"
by (simp add: subst_apply_stateful_strand_def)

lemma db_sst_nil[simp]: "db_sst [ ] I = [ ]"
by (simp add: db_sst_def)

lemma ik_sst_nil[simp]: "ik_sst [ ] = {}"
by (simp add: ik_sst_def)

lemma ik_sst_append[simp]: "ik_sst (A@B) = ik_sst A ∪ ik_sst B"
by (auto simp add: ik_sst_def)

lemma ik_sst_subst: "ik_sst (A ·sst δ) = ik_sst A ·set δ"
proof (induction A)
  case (Cons a A) thus ?case
    by (cases a) (auto simp add: ik_sst_def subst_apply_stateful_strand_def)
qed simp

lemma db_sst_set_is_dbupd_sst: "set (db'_sst A I D) = dbupd_sst A I (set D)" (is "?A = ?B")
proof
  show "?A ⊆ ?B"
  proof
    fix t s show "(t,s) ∈ ?A ⇒ (t,s) ∈ ?B" by (induct rule: db'_sst.induct) auto
  qed
  show "?B ⊆ ?A"
  proof
    fix t s show "(t,s) ∈ ?B ⇒ (t,s) ∈ ?A" by (induct arbitrary: D rule: dbupd_sst.induct) auto
  qed
qed

lemma dbupd_sst_no_upd:
  assumes "∀ a ∈ set A. ¬is_Insert a ∧ ¬is_Delete a"

```

```

shows "dbupdsst A I D = D"
using assms
proof (induction A)
  case (Cons a A) thus ?case by (cases a) auto
qed simp

lemma dbsst_no_upd:
  assumes "∀ a ∈ set A. ¬is_Insert a ∧ ¬is_Delete a"
  shows "db'sst A I D = D"
using assms
proof (induction A)
  case (Cons a A) thus ?case by (cases a) auto
qed simp

lemma dbsst_no_upd_append:
  assumes "∀ b ∈ set B. ¬is_Insert b ∧ ¬is_Delete b"
  shows "db'sst A = db'sst (A@B)"
using assms
proof (induction A)
  case Nil thus ?case by (simp add: dbsst_no_upd)
next
  case (Cons a A) thus ?case by (cases a) simp_all
qed

lemma dbsst_append:
  "db'sst (A@B) I D = db'sst B I (db'sst A I D)"
proof (induction A arbitrary: D)
  case (Cons a A) thus ?case by (cases a) auto
qed simp

lemma dbsst_in_cases:
  assumes "(t,s) ∈ set (db'sst A I D)"
  shows "(t,s) ∈ set D ∨ (∃ t' s'. insert⟨t',s'⟩ ∈ set A ∧ t = t' · I ∧ s = s' · I)"
using assms
proof (induction A arbitrary: D)
  case (Cons a A) thus ?case by (cases a) fastforce+
qed simp

lemma dbsst_in_cases':
  assumes "(t,s) ∈ set (db'sst A I D)"
  and "(t,s) ∉ set D"
  shows "∃ B C t' s'. A = B@insert⟨t',s'⟩#C ∧ t = t' · I ∧ s = s' · I ∧
    (∀ t'' s''. delete⟨t'',s''⟩ ∈ set C ⟶ t ≠ t'' · I ∨ s ≠ s'' · I)"
using assms(1)
proof (induction A rule: List.rev_induct)
  case (snoc a A)
  note * = snoc dbsst_append[of A "[a]" I D]
  thus ?case
  proof (cases a)
    case (Insert t' s')
    thus ?thesis using * by (cases "(t,s) ∈ set (db'sst A I D)") force+
  next
    case (Delete t' s')
    hence **: "t ≠ t' · I ∨ s ≠ s' · I" using * by simp

    have "(t,s) ∈ set (db'sst A I D)" using * Delete by force
    then obtain B C u v where B:
      "A = B@insert⟨u,v⟩#C" "t = u · I" "s = v · I"
      "∀ t' s'. delete⟨t',s'⟩ ∈ set C ⟶ t ≠ t' · I ∨ s ≠ s' · I"
    using snoc.IH by mouna

    have "A@[a] = B@insert⟨u,v⟩#(C@[a])"
      "∀ t' s'. delete⟨t',s'⟩ ∈ set (C@[a]) ⟶ t ≠ t' · I ∨ s ≠ s' · I"

```



```

using B(1,4) Delete ** by auto
thus ?thesis using B(2,3) by blast
qed force+
qed (simp add: assms(2))

lemma db_sst_filter:
  "db'_sst A I D = db'_sst (filter is_Update A) I D"
by (induct A I D rule: db'_sst.induct) simp_all

lemma subst_sst_cons: "a#A ·sst δ = (a ·sstp δ)#(A ·sst δ)"
by (simp add: subst_apply_stateful_strand_def)

lemma subst_sst_snoc: "A@[a] ·sst δ = (A ·sst δ)@[a ·sstp δ]"
by (simp add: subst_apply_stateful_strand_def)

lemma subst_sst_append[simp]: "A@B ·sst δ = (A ·sst δ)@(B ·sst δ)"
by (simp add: subst_apply_stateful_strand_def)

lemma sst_vars_append_subset:
  "fv_sst A ⊆ fv_sst (A@B)" "bvars_sst A ⊆ bvars_sst (A@B)"
  "fv_sst B ⊆ fv_sst (A@B)" "bvars_sst B ⊆ bvars_sst (A@B)"
by auto

lemma sst_vars_disj_cons[simp]: "fv_sst (a#A) ∩ bvars_sst (a#A) = {} ⟹ fv_sst A ∩ bvars_sst A = {}"
unfolding fv_sst_def bvars_sst_def by auto

lemma fv_sst_cons_subset[simp]: "fv_sst A ⊆ fv_sst (a#A)"
by auto

lemma fv_sstp_subst_cases[simp]:
  "fv_sstp (send⟨t⟩ ·sstp ϑ) = fv (t · ϑ)"
  "fv_sstp (receive⟨t⟩ ·sstp ϑ) = fv (t · ϑ)"
  "fv_sstp (⟨c: t ≐ s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "fv_sstp (insert⟨t,s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "fv_sstp (delete⟨t,s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "fv_sstp (⟨c: t ∈ s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "fv_sstp (∀X(∀≠: F ∨≠: G) ·sstp ϑ) =
    fv_pairs (F ·pairs rm_vars (set X) ϑ) ∪ fv_pairs (G ·pairs rm_vars (set X) ϑ) - set X"
by simp_all

lemma vars_sstp_cases[simp]:
  "vars_sstp (send⟨t⟩) = fv t"
  "vars_sstp (receive⟨t⟩) = fv t"
  "vars_sstp (⟨c: t ≐ s⟩) = fv t ∪ fv s"
  "vars_sstp (insert⟨t,s⟩) = fv t ∪ fv s"
  "vars_sstp (delete⟨t,s⟩) = fv t ∪ fv s"
  "vars_sstp (⟨c: t ∈ s⟩) = fv t ∪ fv s"
  "vars_sstp (∀X(∀≠: F ∨≠: G)) = fv_pairs F ∪ fv_pairs G ∪ set X" (is ?A)
  "vars_sstp (∀X(∀≠: [(t,s)] ∨≠: [])) = fv t ∪ fv s ∪ set X" (is ?B)
  "vars_sstp (∀X(∀≠: [] ∨≠: [(t,s)])) = fv t ∪ fv s ∪ set X" (is ?C)
proof
  show ?A ?B ?C by auto
qed simp_all

lemma vars_sstp_subst_cases[simp]:
  "vars_sstp (send⟨t⟩ ·sstp ϑ) = fv (t · ϑ)"
  "vars_sstp (receive⟨t⟩ ·sstp ϑ) = fv (t · ϑ)"
  "vars_sstp (⟨c: t ≐ s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "vars_sstp (insert⟨t,s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "vars_sstp (delete⟨t,s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "vars_sstp (⟨c: t ∈ s⟩ ·sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)"
  "vars_sstp (∀X(∀≠: F ∨≠: G) ·sstp ϑ) =
    fv_pairs (F ·pairs rm_vars (set X) ϑ) ∪ fv_pairs (G ·pairs rm_vars (set X) ϑ) ∪ set X" (is ?A)

```

```

"varssstp ( $\forall X \langle \forall \neq: [(t,s)] \ \forall \notin: [] \rangle \cdot_{sstp} \vartheta$ ) =
  fv ( $t \cdot rm\_vars$  (set X)  $\vartheta$ )  $\cup$  fv ( $s \cdot rm\_vars$  (set X)  $\vartheta$ )  $\cup$  set X" (is ?B)
"varssstp ( $\forall X \langle \forall \neq: [] \ \forall \notin: [(t,s)] \rangle \cdot_{sstp} \vartheta$ ) =
  fv ( $t \cdot rm\_vars$  (set X)  $\vartheta$ )  $\cup$  fv ( $s \cdot rm\_vars$  (set X)  $\vartheta$ )  $\cup$  set X" (is ?C)
proof
  show ?A ?B ?C by auto
qed simp_all

```

```

lemma bvarssst_cons_subset: "bvarssst A  $\subseteq$  bvarssst (a#A)"
by auto

```

```

lemma bvarssstp_subst: "bvarssstp (a  $\cdot_{sstp}$   $\delta$ ) = bvarssstp a"
by (cases a) auto

```

```

lemma bvarssst_subst: "bvarssst (A  $\cdot_{sst}$   $\delta$ ) = bvarssst A"
using bvarssstp_subst[of _  $\delta$ ]
by (induct A) (simp_all add: subst_apply_stateful_strand_def)

```

```

lemma bvarssstp_set_cases[simp]:
  "set (bvarssstp (send⟨t⟩)) = {}"
  "set (bvarssstp (receive⟨t⟩)) = {}"
  "set (bvarssstp ( $\langle c: t \doteq s \rangle$ )) = {}"
  "set (bvarssstp (insert⟨t,s⟩)) = {}"
  "set (bvarssstp (delete⟨t,s⟩)) = {}"
  "set (bvarssstp ( $\langle c: t \in s \rangle$ )) = {}"
  "set (bvarssstp ( $\forall X \langle \forall \neq: F \ \forall \notin: G \rangle$ )) = set X"
by simp_all

```

```

lemma bvarssstp_NegChecks: " $\neg is\_NegChecks$  a  $\implies$  bvarssstp a = []"
by (cases a) simp_all

```

```

lemma bvarssst_NegChecks: "bvarssst A = bvarssst (filter is_NegChecks A)"
proof (induction A)
  case (Cons a A) thus ?case by (cases a) fastforce+
qed simp

```

```

lemma varssst_append[simp]: "varssst (A@B) = varssst A  $\cup$  varssst B"
by (simp add: varssst_def)

```

```

lemma varssst_Nil[simp]: "varssst [] = {}"
by (simp add: varssst_def)

```

```

lemma varssst_Cons: "varssst (a#A) = varssstp a  $\cup$  varssst A"
by (simp add: varssst_def)

```

```

lemma fvsst_Cons: "fvsst (a#A) = fvsstp a  $\cup$  fvsst A"
unfolding fvsst_def by simp

```

```

lemma bvarssst_Cons: "bvarssst (a#A) = set (bvarssstp a)  $\cup$  bvarssst A"
unfolding bvarssst_def by auto

```

```

lemma varssst_Cons'[simp]:
  "varssst (send⟨t⟩#A) = varssstp (send⟨t⟩)  $\cup$  varssst A"
  "varssst (receive⟨t⟩#A) = varssstp (receive⟨t⟩)  $\cup$  varssst A"
  "varssst ( $\langle a: t \doteq s \rangle$ #A) = varssstp ( $\langle a: t \doteq s \rangle$ )  $\cup$  varssst A"
  "varssst (insert⟨t,s⟩#A) = varssstp (insert⟨t,s⟩)  $\cup$  varssst A"
  "varssst (delete⟨t,s⟩#A) = varssstp (delete⟨t,s⟩)  $\cup$  varssst A"
  "varssst ( $\langle a: t \in s \rangle$ #A) = varssstp ( $\langle a: t \in s \rangle$ )  $\cup$  varssst A"
  "varssst ( $\forall X \langle \forall \neq: F \ \forall \notin: G \rangle$ #A) = varssstp ( $\forall X \langle \forall \neq: F \ \forall \notin: G \rangle$ )  $\cup$  varssst A"
by (simp_all add: varssst_def)

```

```

lemma varssstp_is_fvsstp_bvarssstp:
  fixes x: "('a, 'b) stateful_strand_step"

```

```

shows "varssstp x = fvsstp x ∪ set (bvarssstp x)"
proof (cases x)
  case (NegChecks X F G) thus ?thesis by (induct F) force+
qed simp_all

lemma varssst_is_fvsst_bvarssst:
  fixes S::('a,'b) stateful_strand
  shows "varssst S = fvsst S ∪ bvarssst S"
proof (induction S)
  case (Cons x S) thus ?case
    using varssstp_is_fvsstp_bvarssstp[of x]
    by (auto simp add: varssst_def)
qed simp

lemma varssstp_NegCheck[simp]:
  "varssstp (∀X(∀≠: F ∨≠: G)) = set X ∪ fvpairs F ∪ fvpairs G"
by (simp_all add: sup_commute sup_left_commute varssstp_is_fvsstp_bvarssstp)

lemma bvarssstp_NegCheck[simp]:
  "bvarssstp (∀X(∀≠: F ∨≠: G)) = X"
  "set (bvarssstp (∀ [] (∀≠: F ∨≠: G))) = {}"
by simp_all

lemma fvsstp_NegCheck[simp]:
  "fvsstp (∀X(∀≠: F ∨≠: G)) = fvpairs F ∪ fvpairs G - set X"
  "fvsstp (∀ [] (∀≠: F ∨≠: G)) = fvpairs F ∪ fvpairs G"
  "fvsstp (⟨t != s⟩) = fv t ∪ fv s"
  "fvsstp (⟨t not in s⟩) = fv t ∪ fv s"
by simp_all

lemma fvsst_append[simp]: "fvsst (A@B) = fvsst A ∪ fvsst B"
by simp

lemma bvarssst_append[simp]: "bvarssst (A@B) = bvarssst A ∪ bvarssst B"
by auto

lemma fvsstp_is_subterm_trmssstp:
  assumes "x ∈ fvsstp a"
  shows "Var x ∈ subtermsset (trmssstp a)"
using assms var_is_subterm
proof (cases a)
  case (NegChecks X F F')
  hence "x ∈ fvpairs F ∪ fvpairs F' - set X" using assms by simp
  thus ?thesis using NegChecks var_is_subterm by fastforce
qed force+

lemma fvsst_is_subterm_trmssst: "x ∈ fvsst A ⟹ Var x ∈ subtermsset (trmssst A)"
proof (induction A)
  case (Cons a A) thus ?case using fvsstp_is_subterm_trmssstp by (cases "x ∈ fvsst A") auto
qed simp

lemma var_subterm_trmssstp_is_varssstp:
  assumes "Var x ∈ subtermsset (trmssstp a)"
  shows "x ∈ varssstp a"
using assms vars_iff_subtermeq
proof (cases a)
  case (NegChecks X F F')
  hence "Var x ∈ subtermsset (trmspairs F ∪ trmspairs F'" using assms by simp
  thus ?thesis using NegChecks vars_iff_subtermeq by force
qed force+

lemma var_subterm_trmssst_is_varssst: "Var x ∈ subtermsset (trmssst A) ⟹ x ∈ varssst A"
proof (induction A)

```

```

case (Cons a A)
show ?case
proof (cases "Var x ∈ subtermsset (trmssst A)")
  case True thus ?thesis using Cons.IH by (simp add: varssst_def)
next
  case False thus ?thesis
    using Cons.prems var_subterm_trmssstp_is_varssstp
    by (fastforce simp add: varssst_def)
qed
qed simp

lemma var_trmssst_is_varssst: "Var x ∈ trmssst A ⇒ x ∈ varssst A"
by (meson var_subterm_trmssst_is_varssst UN_I term.order_refl)

lemma iksst_trmssst_subset: "iksst A ⊆ trmssst A"
by (force simp add: iksst_def)

lemma var_subterm_iksst_is_varssst: "Var x ∈ subtermsset (iksst A) ⇒ x ∈ varssst A"
using var_subterm_trmssst_is_varssst iksst_trmssst_subset by fast

lemma var_subterm_iksst_is_fvsst:
  assumes "Var x ∈ subtermsset (iksst A)"
  shows "x ∈ fvsst A"
proof -
  obtain t where t: "Receive t ∈ set A" "Var x ⊆ t" using assms unfolding iksst_def by moura
  hence "fv t ⊆ fvsst A" unfolding fvsst_def by force
  thus ?thesis using t(2) by (meson contra_subsetD subterm_is_var)
qed

lemma fv_iksst_is_fvsst:
  assumes "x ∈ fvset (iksst A)"
  shows "x ∈ fvsst A"
using var_subterm_iksst_is_fvsst assms var_is_subterm by fastforce

lemma fv_trmssst_subset:
  "fvset (trmssst S) ⊆ varssst S"
  "fvsst S ⊆ fvset (trmssst S)"
proof (induction S)
  case (Cons x S)
  have *: "fvset (trmssst (x#S)) = fvset (trmssstp x) ∪ fvset (trmssst S)"
    "fvsst (x#S) = fvsstp x ∪ fvsst S" "varssst (x#S) = varssstp x ∪ varssst S"
    unfolding trmssst_def fvsst_def varssst_def
    by auto
  { case 1
    show ?case using Cons.IH(1)
    proof (cases x)
      case (NegChecks X F G)
      hence "trmssstp x = trmspairs F ∪ trmspairs G"
        "varssstp x = fvpairs F ∪ fvpairs G ∪ set X"
      by (simp, meson varssstp_cases(7))
      hence "fvset (trmssstp x) ⊆ varssstp x"
        using fv_trmspairs_is_fvpairs[of F] fv_trmspairs_is_fvpairs[of G]
        by auto
      thus ?thesis
        using Cons.IH(1) *(1,3)
        by blast
    qed auto
  }
  { case 2
    show ?case using Cons.IH(2)
    proof (cases x)

```

```

case (NegChecks X F G)
hence "trmssstp x = trmspairs F ∪ trmspairs G"
      "fvsstp x = (fvpairs F ∪ fvpairs G) - set X"
  by auto
hence "fvsstp x ⊆ fvset (trmssstp x)"
  using fv_trmspairs_is_fvpairs[of F] fv_trmspairs_is_fvpairs[of G]
  by auto
thus ?thesis
  using Cons.IH(2) *(1,2)
  by blast
qed auto
}
qed simp_all

lemma fv_ik_subset_fv_sst'[simp]: "fvset (iksst S) ⊆ fvsst S"
unfolding iksst_def by (induct S) auto

lemma fv_ik_subset_vars_sst'[simp]: "fvset (iksst S) ⊆ varssst S"
using fv_ik_subset_fv_sst' fv_trmssst_subset by fast

lemma iksst_var_is_fv: "Var x ∈ subtermsset (iksst A) ⇒ x ∈ fvsst A"
by (meson fv_ik_subset_fv_sst'[of A] fv_subset_subterms subsetCE term.set_intros(3))

lemma varssstp_subst_cases':
  assumes x: "x ∈ varssstp (s ·sstp v)"
  shows "x ∈ varssstp s ∨ x ∈ fvset (v ' varssstp s)"
using x vars_term_subst[of _ v] varssstp_cases(1,2,3,4,5,6) varssstp_subst_cases(1,2)[of _ v]
  varssstp_subst_cases(3,6)[of _ _ v] varssstp_subst_cases(4,5)[of _ _ v]
proof (cases s)
  case (NegChecks X F G)
  let ?v' = "rm_vars (set X) v"
  have "x ∈ fvpairs (F ·pairs ?v') ∨ x ∈ fvpairs (G ·pairs ?v') ∨ x ∈ set X"
  using varssstp_subst_cases(7)[of X F G v] x NegChecks by simp
  hence "x ∈ fvset (?v' ' fvpairs F) ∨ x ∈ fvset (?v' ' fvpairs G) ∨ x ∈ set X"
  using fvpairs_subst[of _ ?v'] by blast
  hence "x ∈ fvset (v ' fvpairs F) ∨ x ∈ fvset (v ' fvpairs G) ∨ x ∈ set X"
  using rm_vars_fvset_subst by fast
  thus ?thesis
  using NegChecks varssstp_cases(7)[of X F G]
  by auto
qed simp_all

lemma varssst_subst_cases:
  assumes "x ∈ varssst (S ·sst v)"
  shows "x ∈ varssst S ∨ x ∈ fvset (v ' varssst S)"
  using assms
proof (induction S)
  case (Cons s S) thus ?case
  proof (cases "x ∈ varssst (S ·sst v)")
    case False
    note * = subst_sst_cons[of s S v] varssst_Cons[of "s ·sstp v" "S ·sst v"] varssst_Cons[of s S]
    have **: "x ∈ varssstp (s ·sstp v)" using Cons.prem False * by simp
    show ?thesis using varssstp_subst_cases'[OF **] * by auto
  qed (auto simp add: varssst_def)
qed simp

lemma subset_subst_pairs_diff_exists:
  fixes I:: "('a, 'b) subst" and D D'::: "('a, 'b) dbstate"
  shows "∃ Di. Di ⊆ D ∧ Di ·pset I = (D ·pset I) - D'"
by (metis (no_types, lifting) Diff_subset subset_image_iff)

lemma subset_subst_pairs_diff_exists':
  fixes I:: "('a, 'b) subst" and D:: "('a, 'b) dbstate"

```

```

assumes "finite D"
shows "∃ Di. Di ⊆ D ∧ Di ·pset I ⊆ {d ·p I} ∧ d ·p I ∉ (D - Di) ·pset I"
using assms
proof (induction D rule: finite_induct)
  case (insert d' D)
  then obtain Di where IH: "Di ⊆ D" "Di ·pset I ⊆ {d ·p I}" "d ·p I ∉ (D - Di) ·pset I" by moura
  show ?case
  proof (cases "d' ·p I = d ·p I")
    case True
    hence "insert d' Di ⊆ insert d' D" "insert d' Di ·pset I ⊆ {d ·p I}"
      "d ·p I ∉ (insert d' D - insert d' Di) ·pset I"
      using IH by auto
    thus ?thesis by metis
  next
  case False
  hence "Di ⊆ insert d' D" "Di ·pset I ⊆ {d ·p I}"
      "d ·p I ∉ (insert d' D - Di) ·pset I"
      using IH by auto
  thus ?thesis by metis
qed
qed simp

```

```

lemma stateful_strand_step_subst_inI[intro]:
  "send⟨t⟩ ∈ set A ⟹ send⟨t · v⟩ ∈ set (A ·sst v)"
  "receive⟨t⟩ ∈ set A ⟹ receive⟨t · v⟩ ∈ set (A ·sst v)"
  "⟨c: t ≐ s⟩ ∈ set A ⟹ ⟨c: (t · v) ≐ (s · v)⟩ ∈ set (A ·sst v)"
  "insert⟨t, s⟩ ∈ set A ⟹ insert⟨t · v, s · v⟩ ∈ set (A ·sst v)"
  "delete⟨t, s⟩ ∈ set A ⟹ delete⟨t · v, s · v⟩ ∈ set (A ·sst v)"
  "⟨c: t ∈ s⟩ ∈ set A ⟹ ⟨c: (t · v) ∈ (s · v)⟩ ∈ set (A ·sst v)"
  "∀ X (∀ ≠: F ∀ ≠: G) ∈ set A
    ⟹ ∀ X (∀ ≠: (F ·pairs rm_vars (set X) v) ∨ ≠: (G ·pairs rm_vars (set X) v)) ∈ set (A ·sst v)"
  "⟨t != s⟩ ∈ set A ⟹ ⟨t · v != s · v⟩ ∈ set (A ·sst v)"
  "⟨t not in s⟩ ∈ set A ⟹ ⟨t · v not in s · v⟩ ∈ set (A ·sst v)"
proof (induction A)
  case (Cons a A)
  note * = subst_sst_cons[of a A v]
  { case 1 thus ?case using Cons.IH(1) * by (cases a) auto }
  { case 2 thus ?case using Cons.IH(2) * by (cases a) auto }
  { case 3 thus ?case using Cons.IH(3) * by (cases a) auto }
  { case 4 thus ?case using Cons.IH(4) * by (cases a) auto }
  { case 5 thus ?case using Cons.IH(5) * by (cases a) auto }
  { case 6 thus ?case using Cons.IH(6) * by (cases a) auto }
  { case 7 thus ?case using Cons.IH(7) * by (cases a) auto }
  { case 8 thus ?case using Cons.IH(8) * by (cases a) auto }
  { case 9 thus ?case using Cons.IH(9) * by (cases a) auto }
qed simp_all

```

```

lemma stateful_strand_step_cases_subst:
  "is_Send a = is_Send (a ·sstp v)"
  "is_Receive a = is_Receive (a ·sstp v)"
  "is_Equality a = is_Equality (a ·sstp v)"
  "is_Insert a = is_Insert (a ·sstp v)"
  "is_Delete a = is_Delete (a ·sstp v)"
  "is_InSet a = is_InSet (a ·sstp v)"
  "is_NegChecks a = is_NegChecks (a ·sstp v)"
  "is_Assignment a = is_Assignment (a ·sstp v)"
  "is_Check a = is_Check (a ·sstp v)"
  "is_Update a = is_Update (a ·sstp v)"
by (cases a; simp_all)+

```

```

lemma stateful_strand_step_subst_inv_cases:
  "send⟨t⟩ ∈ set (S ·sst σ) ⟹ ∃ t'. t = t' · σ ∧ send⟨t'⟩ ∈ set S"
  "receive⟨t⟩ ∈ set (S ·sst σ) ⟹ ∃ t'. t = t' · σ ∧ receive⟨t'⟩ ∈ set S"

```

```

"⟨c: t ≐ s⟩ ∈ set (S ·sst σ) ⇒ ∃ t' s'. t = t' · σ ∧ s = s' · σ ∧ ⟨c: t' ≐ s'⟩ ∈ set S"
"insert⟨t,s⟩ ∈ set (S ·sst σ) ⇒ ∃ t' s'. t = t' · σ ∧ s = s' · σ ∧ insert⟨t',s'⟩ ∈ set S"
"delete⟨t,s⟩ ∈ set (S ·sst σ) ⇒ ∃ t' s'. t = t' · σ ∧ s = s' · σ ∧ delete⟨t',s'⟩ ∈ set S"
"⟨c: t ∈ s⟩ ∈ set (S ·sst σ) ⇒ ∃ t' s'. t = t' · σ ∧ s = s' · σ ∧ ⟨c: t' ∈ s'⟩ ∈ set S"
"∀ X (∀ ≠: F ∨ ≠: G) ∈ set (S ·sst σ) ⇒
  ∃ F' G'. F = F' ·pairs rm_vars (set X) σ ∧ G = G' ·pairs rm_vars (set X) σ ∧
  ∀ X (∀ ≠: F' ∨ ≠: G') ∈ set S"

```

proof (induction S)

case (Cons a S)

have \*: "x ∈ set (S ·<sub>sst</sub> σ)"

when "x ∈ set (a#S ·<sub>sst</sub> σ)" "x ≠ a ·<sub>sstp</sub> σ" for x

using that by (simp add: subst\_apply\_stateful\_strand\_def)

```

{ case 1 thus ?case using Cons.IH(1)[OF *] by (cases a) auto }
{ case 2 thus ?case using Cons.IH(2)[OF *] by (cases a) auto }
{ case 3 thus ?case using Cons.IH(3)[OF *] by (cases a) auto }
{ case 4 thus ?case using Cons.IH(4)[OF *] by (cases a) auto }
{ case 5 thus ?case using Cons.IH(5)[OF *] by (cases a) auto }
{ case 6 thus ?case using Cons.IH(6)[OF *] by (cases a) auto }
{ case 7 thus ?case using Cons.IH(7)[OF *] by (cases a) auto }

```

qed simp\_all

lemma stateful\_strand\_step\_fv\_subset\_cases:

"send⟨t⟩ ∈ set S ⇒ fv t ⊆ fv<sub>sst</sub> S"

"receive⟨t⟩ ∈ set S ⇒ fv t ⊆ fv<sub>sst</sub> S"

"⟨c: t ≐ s⟩ ∈ set S ⇒ fv t ∪ fv s ⊆ fv<sub>sst</sub> S"

"insert⟨t,s⟩ ∈ set S ⇒ fv t ∪ fv s ⊆ fv<sub>sst</sub> S"

"delete⟨t,s⟩ ∈ set S ⇒ fv t ∪ fv s ⊆ fv<sub>sst</sub> S"

"⟨c: t ∈ s⟩ ∈ set S ⇒ fv t ∪ fv s ⊆ fv<sub>sst</sub> S"

"∀ X (∀ ≠: F ∨ ≠: G) ∈ set S ⇒ fv<sub>pairs</sub> F ∪ fv<sub>pairs</sub> G - set X ⊆ fv<sub>sst</sub> S"

proof (induction S)

case (Cons a S)

```

{ case 1 thus ?case using Cons.IH(1) by auto }
{ case 2 thus ?case using Cons.IH(2) by auto }
{ case 3 thus ?case using Cons.IH(3) by auto }
{ case 4 thus ?case using Cons.IH(4) by auto }
{ case 5 thus ?case using Cons.IH(5) by auto }
{ case 6 thus ?case using Cons.IH(6) by auto }
{ case 7 thus ?case using Cons.IH(7) by fastforce }

```

qed simp\_all

lemma trms<sub>sst</sub>\_nil[simp]:

"trms<sub>sst</sub> [] = {}"

unfolding trms<sub>sst</sub>\_def by simp

lemma trms<sub>sst</sub>\_mono:

"set M ⊆ set N ⇒ trms<sub>sst</sub> M ⊆ trms<sub>sst</sub> N"

by auto

lemma trms<sub>sst</sub>\_in:

assumes "t ∈ trms<sub>sst</sub> S"

shows "∃ a ∈ set S. t ∈ trms<sub>sstp</sub> a"

using assms unfolding trms<sub>sst</sub>\_def by simp

lemma trms<sub>sst</sub>\_cons: "trms<sub>sst</sub> (a#A) = trms<sub>sstp</sub> a ∪ trms<sub>sst</sub> A"

unfolding trms<sub>sst</sub>\_def by force

lemma trms<sub>sst</sub>\_append[simp]: "trms<sub>sst</sub> (A@B) = trms<sub>sst</sub> A ∪ trms<sub>sst</sub> B"

unfolding trms<sub>sst</sub>\_def by force

lemma trms<sub>sstp</sub>\_subst:

assumes "set (bvars<sub>sstp</sub> a) ∩ subst\_domain ϑ = {}"

shows "trms<sub>sstp</sub> (a ·<sub>sstp</sub> ϑ) = trms<sub>sstp</sub> a ·<sub>set</sub> ϑ"

```

proof (cases a)
  case (NegChecks X F G)
  hence "rm_vars (set X)  $\vartheta$  =  $\vartheta$ " using assms rm_vars_apply' [of  $\vartheta$  "set X"] by auto
  hence "trmssstp (a ·sstp  $\vartheta$ ) = trmspairs (F ·pairs  $\vartheta$ )  $\cup$  trmspairs (G ·pairs  $\vartheta$ )"
    "trmssstp a ·set  $\vartheta$  = (trmspairs F ·set  $\vartheta$ )  $\cup$  (trmspairs G ·set  $\vartheta$ )"
    using NegChecks image_Un by simp_all
  thus ?thesis by (metis trmspairs_subst)
qed simp_all

lemma trmssstp_subst':
  assumes "¬is_NegChecks a"
  shows "trmssstp (a ·sstp  $\vartheta$ ) = trmssstp a ·set  $\vartheta$ "
using assms by (cases a) simp_all

lemma trmssstp_subst'':
  fixes t::('a,'b) term" and  $\delta$ ::('a,'b) subst"
  assumes "t  $\in$  trmssstp (b ·sstp  $\delta$ )"
  shows " $\exists s \in$  trmssstp b. t = s · rm_vars (set (bvarssstp b))  $\delta$ "
proof (cases "is_NegChecks b")
  case True
  then obtain X F G where *: "b = NegChecks X F G" by (cases b) moura+
  thus ?thesis using assms trmspairs_subst [of _ "rm_vars (set X)  $\delta$ "] by auto
next
  case False
  hence "trmssstp (b ·sstp  $\delta$ ) = trmssstp b ·set rm_vars (set (bvarssstp b))  $\delta$ "
    using trmssstp_subst' bvarssstp_NegChecks
    by fastforce
  thus ?thesis using assms by fast
qed

lemma trmssstp_subst''':
  fixes t::('a,'b) term" and  $\delta$   $\vartheta$ ::('a,'b) subst"
  assumes "t  $\in$  trmssstp (b ·sstp  $\delta$ ) ·set  $\vartheta$ "
  shows " $\exists s \in$  trmssstp b. t = s · rm_vars (set (bvarssstp b))  $\delta \circ_s \vartheta$ "
proof -
  obtain s where s: "s  $\in$  trmssstp (b ·sstp  $\delta$ )" "t = s ·  $\vartheta$ " using assms by moura
  show ?thesis using trmssstp_subst'' [OF s(1)] s(2) by auto
qed

lemma trmssst_subst:
  assumes "bvarssst S  $\cap$  subst_domain  $\vartheta$  = {}"
  shows "trmssst (S ·sst  $\vartheta$ ) = trmssst S ·set  $\vartheta$ "
using assms
proof (induction S)
  case (Cons a S)
  hence IH: "trmssst (S ·sst  $\vartheta$ ) = trmssst S ·set  $\vartheta$ " and *: "set (bvarssstp a)  $\cap$  subst_domain  $\vartheta$  = {}"
    by auto
  show ?case using trmssstp_subst [OF *] IH by (auto simp add: subst_apply_stateful_strand_def)
qed simp

lemma trmssst_subst_cons:
  "trmssst (a#A ·sst  $\delta$ ) = trmssstp (a ·sstp  $\delta$ )  $\cup$  trmssst (A ·sst  $\delta$ )"
using subst_sst_cons [of a A  $\delta$ ] trmssst_cons [of a A] trmssst_append by simp

lemma (in intruder_model) wftrms_trmssstp_subst:
  assumes "wftrms (trmssstp a ·set  $\delta$ )"
  shows "wftrms (trmssstp (a ·sstp  $\delta$ ))"
using assms
proof (cases a)
  case (NegChecks X F G)
  hence *: "trmssstp (a ·sstp  $\delta$ ) =
    (trmspairs (F ·pairs rm_vars (set X)  $\delta$ ))  $\cup$  (trmspairs (G ·pairs rm_vars (set X)  $\delta$ ))"
    by simp

```



```

have "trms_sstp a ·set δ = (trms_pairs F ·set δ) ∪ (trms_pairs G ·set δ)"
  using NegChecks image_Un by simp
hence "wf_trms (trms_pairs F ·set δ)" "wf_trms (trms_pairs G ·set δ)" using * assms by auto
hence "wf_trms (trms_pairs F ·set rm_vars (set X) δ)"
  "wf_trms (trms_pairs G ·set rm_vars (set X) δ)"
  using wf_trms_subst_rm_vars[of δ "trms_pairs F" "set X"]
    wf_trms_subst_rm_vars[of δ "trms_pairs G" "set X"]
  by fast+
thus ?thesis
  using * trms_pairs_subst[of _ "rm_vars (set X) δ"]
  by auto
qed auto

lemma trms_sst_fv_vars_sst_subset: "t ∈ trms_sst A ⇒ fv t ⊆ vars_sst A"
proof (induction A)
  case (Cons a A) thus ?case by (cases a) auto
qed simp

lemma trms_sst_fv_subst_subset:
  assumes "t ∈ trms_sst S" "subst_domain ϑ ∩ bvars_sst S = {}"
  shows "fv (t · ϑ) ⊆ vars_sst (S ·sst ϑ)"
using assms
proof (induction S)
  case (Cons s S) show ?case
  proof (cases "t ∈ trms_sst S")
    case True
    hence "fv (t · ϑ) ⊆ vars_sst (S ·sst ϑ)" using Cons.IH Cons.prem by auto
    thus ?thesis using subst_sst_cons[of s S ϑ] unfolding vars_sst_def by auto
  next
    case False
    hence *: "t ∈ trms_sstp s" "subst_domain ϑ ∩ set (bvars_sstp s) = {}" using Cons.prem by auto
    hence "fv (t · ϑ) ⊆ vars_sstp (s ·sst ϑ)"
    proof (cases s)
      case (NegChecks X F G)
      hence **: "t ∈ trms_pairs F ∨ t ∈ trms_pairs G" using *(1) by auto
      have ***: "rm_vars (set X) ϑ = ϑ" using *(2) NegChecks rm_vars_apply' by auto
      have "fv (t · ϑ) ⊆ fv_pairs (F ·pairs rm_vars (set X) ϑ) ∪ fv_pairs (G ·pairs rm_vars (set X) ϑ)"
        using ** *** trms_pairs_fv_subst_subset[of t _ ϑ] by auto
      thus ?thesis using *(2) using NegChecks vars_sstp_subst_cases(7)[of X F G ϑ] by blast
    qed auto
    thus ?thesis using subst_sst_cons[of s S ϑ] unfolding vars_sst_def by auto
  qed
qed simp

lemma trms_sst_fv_subst_subset':
  assumes "t ∈ subterms_set (trms_sst S)" "fv t ∩ bvars_sst S = {}" "fv (t · ϑ) ∩ bvars_sst S = {}"
  shows "fv (t · ϑ) ⊆ fv_sst (S ·sst ϑ)"
using assms
proof (induction S)
  case (Cons s S) show ?case
  proof (cases "t ∈ subterms_set (trms_sst S)")
    case True
    hence "fv (t · ϑ) ⊆ fv_sst (S ·sst ϑ)" using Cons.IH Cons.prem by auto
    thus ?thesis using subst_sst_cons[of s S ϑ] unfolding vars_sst_def by auto
  next
    case False
    hence 0: "t ∈ subterms_set (trms_sstp s)" "fv t ∩ set (bvars_sstp s) = {}"
      "fv (t · ϑ) ∩ set (bvars_sstp s) = {}"
      using Cons.prem by auto
    using Cons.prem by auto
  next
    note 1 = UN_Un UN_insert fv_set.simps subst_apply_fv_subset subst_apply_fv_unfold
      subst_apply_term_empty sup_bot.comm_neutral fv_subterms_set fv_subset[OF 0(1)]

```

```

note 2 = subst_apply_fv_union

have "fv (t ·  $\vartheta$ )  $\subseteq$  fvsstp (s ·sstp  $\vartheta$ )"
proof (cases s)
  case (NegChecks X F G)
  hence 3: "t  $\in$  subtermsset (trmspairs F)  $\vee$  t  $\in$  subtermsset (trmspairs G)" using 0(1) by auto
  have "t · rm_vars (set X)  $\vartheta$  = t ·  $\vartheta$ " using 0(2) NegChecks rm_vars_ident[of t] by auto
  hence "fv (t ·  $\vartheta$ )  $\subseteq$  fvpairs (F ·pairs rm_vars (set X)  $\vartheta$ )  $\cup$  fvpairs (G ·pairs rm_vars (set X)  $\vartheta$ )"
    using 3 trmspairs_fv_subst_subset'[of t _ "rm_vars (set X)  $\vartheta$ "] by fastforce
  thus ?thesis using 0(2,3) NegChecks fvsstp_subst_cases(7)[of X F G  $\vartheta$ ] by auto
qed (metis (no_types, lifting) 1 trmssstp.simps(1) fvsstp_subst_cases(1),
  metis (no_types, lifting) 1 trmssstp.simps(2) fvsstp_subst_cases(2),
  metis (no_types, lifting) 1 2 trmssstp.simps(3) fvsstp_subst_cases(3),
  metis (no_types, lifting) 1 2 trmssstp.simps(4) fvsstp_subst_cases(4),
  metis (no_types, lifting) 1 2 trmssstp.simps(5) fvsstp_subst_cases(5),
  metis (no_types, lifting) 1 2 trmssstp.simps(6) fvsstp_subst_cases(6))
thus ?thesis using subst_sst_cons[of s S  $\vartheta$ ] unfolding fvsst_def by auto
qed
qed simp

lemma trmssstp_funcs_term_cases:
  assumes "t  $\in$  trmssstp (s ·sstp  $\vartheta$ )" "f  $\in$  funcs_term t"
  shows "( $\exists u \in$  trmssstp s. f  $\in$  funcs_term u)  $\vee$  ( $\exists x \in$  fvsstp s. f  $\in$  funcs_term ( $\vartheta$  x))"
  using assms
proof (cases s)
  case (NegChecks X F G)
  hence "t  $\in$  trmspairs (F ·pairs rm_vars (set X)  $\vartheta$ )  $\vee$  t  $\in$  trmspairs (G ·pairs rm_vars (set X)  $\vartheta$ )"
    using assms(1) by auto
  hence "( $\exists u \in$  trmspairs F. f  $\in$  funcs_term u)  $\vee$  ( $\exists x \in$  fvpairs F. f  $\in$  funcs_term (rm_vars (set X)  $\vartheta$  x))  $\vee$ 
    ( $\exists u \in$  trmspairs G. f  $\in$  funcs_term u)  $\vee$  ( $\exists x \in$  fvpairs G. f  $\in$  funcs_term (rm_vars (set X)  $\vartheta$  x))"
    using trmspairs_funcs_term_cases[OF _ assms(2), of _ "rm_vars (set X)  $\vartheta$ "] by meson
  hence "( $\exists u \in$  trmspairs F  $\cup$  trmspairs G. f  $\in$  funcs_term u)  $\vee$ 
    ( $\exists x \in$  fvpairs F  $\cup$  fvpairs G. f  $\in$  funcs_term (rm_vars (set X)  $\vartheta$  x))"
    by blast
  thus ?thesis
proof
  assume " $\exists x \in$  fvpairs F  $\cup$  fvpairs G. f  $\in$  funcs_term (rm_vars (set X)  $\vartheta$  x)"
  then obtain x where x: "x  $\in$  fvpairs F  $\cup$  fvpairs G" "f  $\in$  funcs_term (rm_vars (set X)  $\vartheta$  x)"
    by auto
  hence "x  $\notin$  set X" "rm_vars (set X)  $\vartheta$  x =  $\vartheta$  x" by auto
  thus ?thesis using x by (auto simp add: assms NegChecks)
qed (auto simp add: assms NegChecks)
qed (use assms funcs_term_subst[of _  $\vartheta$ ] in auto)

lemma trmssst_funcs_term_cases:
  assumes "t  $\in$  trmssst (S ·sst  $\vartheta$ )" "f  $\in$  funcs_term t"
  shows "( $\exists u \in$  trmssst S. f  $\in$  funcs_term u)  $\vee$  ( $\exists x \in$  fvsst S. f  $\in$  funcs_term ( $\vartheta$  x))"
  using assms(1)
proof (induction S)
  case (Cons s S) thus ?case
  proof (cases "t  $\in$  trmssst (S ·sst  $\vartheta$ )")
    case False
    hence "t  $\in$  trmssstp (s ·sstp  $\vartheta$ )" using Cons.premis(1) subst_sst_cons[of s S  $\vartheta$ ] trmssst_cons by
  auto
  thus ?thesis using trmssstp_funcs_term_cases[OF _ assms(2)] by fastforce
  qed auto
qed simp

lemma fvsst_is_subterm_trmssst_subst:
  assumes "x  $\in$  fvsst T"
  and "bvarssst T  $\cap$  subst_domain  $\vartheta$  = {}"
  shows " $\vartheta$  x  $\in$  subtermsset (trmssst (T ·sst  $\vartheta$ ))"

```

```

using trmssst_subst[OF assms(2)] subterms_subst_subset'[of  $\vartheta$  "trmssst T"]
  fvsst_is_subterm_trmssst[OF assms(1)]
by (metis (no_types, lifting) image_iff subset_iff subst_apply_term.simps(1))

lemma fvsst_subst_fv_subset:
  assumes "x ∈ fvsst S" "x ∉ bvarssst S" "fv ( $\vartheta$  x) ∩ bvarssst S = {}"
  shows "fv ( $\vartheta$  x) ⊆ fvsst (S ·sst  $\vartheta$ )"
using assms
proof (induction S)
  case (Cons a S)
  note 1 = fv_subst_subset[of _ _  $\vartheta$ ]
  note 2 = subst_apply_fv_union subst_apply_fv_unfold[of _  $\vartheta$ ] fv_subset image_eqI
  note 3 = fvsstp_subst_cases
  note 4 = fvsstp.simps
  from Cons show ?case
  proof (cases "x ∈ fvsst S")
    case False
    hence 5: "x ∈ fvsstp a" "fv ( $\vartheta$  x) ∩ set (bvarssstp a) = {}" "x ∉ set (bvarssstp a)"
    using Cons.prem1 by auto
    hence "fv ( $\vartheta$  x) ⊆ fvsstp (a ·sstp  $\vartheta$ )"
    proof (cases a)
      case (NegChecks X F G)
      let ?δ = "rm_vars (set X)  $\vartheta$ "
      have *: "x ∈ fvpairs F ∪ fvpairs G" using NegChecks 5(1) by auto
      have **: "fv ( $\vartheta$  x) ∩ set X = {}" using NegChecks 5(2) by simp
      have ***: " $\vartheta$  x = ?δ x" using NegChecks 5(3) by auto
      have "fv ( $\vartheta$  x) ⊆ fvpairs (F ·pairs ?δ) ∪ fvpairs (G ·pairs ?δ)"
      using fvpairs_subst_fv_subset[of x _ ?δ] * *** by auto
      thus ?thesis using NegChecks ** by auto
    qed (metis (full_types) 1 5(1) 3(1) 4(1), metis (full_types) 1 5(1) 3(2) 4(2),
      metis (full_types) 2 5(1) 3(3) 4(3), metis (full_types) 2 5(1) 3(4) 4(4),
      metis (full_types) 2 5(1) 3(5) 4(5), metis (full_types) 2 5(1) 3(6) 4(6))
    thus ?thesis by (auto simp add: subst_sst_cons[of a S  $\vartheta$ ])
  qed (auto simp add: subst_sst_cons[of a S  $\vartheta$ ])
qed simp

lemma (in intruder_model) wftrms_trmssst_subst:
  assumes "wftrms (trmssst A ·set δ)"
  shows "wftrms (trmssst (A ·sst δ))"
using assms
proof (induction A)
  case (Cons a A)
  hence IH: "wftrms (trmssst (A ·sst δ))" and *: "wftrms (trmssstp a ·set δ)" by auto
  have "wftrms (trmssstp (a ·sstp δ))" by (rule wftrms_trmssstp_subst[OF *])
  thus ?case using IH trmssst_subst_cons[of a A δ] by blast
qed simp

lemma fvsst_subst_obtain_var:
  assumes "x ∈ fvsst (S ·sst δ)"
  shows "∃ y ∈ fvsst S. x ∈ fv (δ y)"
using assms
proof (induction S)
  case (Cons s S)
  hence "x ∈ fvsst (S ·sst δ) ⇒ ∃ y ∈ fvsst S. x ∈ fv (δ y)"
  using bvarssst_cons_subset[of S s]
  by blast
  thus ?case
  proof (cases "x ∈ fvsst (S ·sst δ)")
    case False
    hence *: "x ∈ fvsstp (s ·sstp δ)"
    using Cons.prem1 subst_sst_cons[of s S δ]
    by fastforce
  qed

```

```

have "∃y ∈ fvsstp s. x ∈ fv (δ y)"
proof (cases s)
  case (NegChecks X F G)
  hence "x ∈ fvpairs (F ·pairs rm_vars (set X) δ) ∨ x ∈ fvpairs (G ·pairs rm_vars (set X) δ)"
    and **: "x ∉ set X"
    using * by simp_all
  then obtain y where y: "y ∈ fvpairs F ∨ y ∈ fvpairs G" "x ∈ fv ((rm_vars (set X) δ) y)"
    using fvpairs_subst_obtain_var[of _ _ "rm_vars (set X) δ"]
    by blast
  have "y ∉ set X"
  proof
    assume y_in: "y ∈ set X"
    hence "(rm_vars (set X) δ) y = Var y" by auto
    hence "x = y" using y(2) by simp
    thus False using ** y_in by metis
  qed
  thus ?thesis using NegChecks y by auto
qed (use * fv_subst_obtain_var in force)+
thus ?thesis by auto
qed auto
qed simp

```

```

lemma fvsst_subst_subset_range_vars_if_subset_domain:
  assumes "fvsst S ⊆ subst_domain σ"
  shows "fvsst (S ·sst σ) ⊆ range_vars σ"
using assms fvsst_subst_obtain_var[of _ S σ] subst_dom_vars_in_subst[of _ σ] subst_fv_imgI[of σ]
by (metis (no_types) in_mono subsetI)

```

```

lemma fvsst_in_fv_trmssst: "x ∈ fvsst S ⇒ x ∈ fvset (trmssst S)"
proof (induction S)
  case (Cons s S) thus ?case
  proof (cases "x ∈ fvsst S")
    case False
    hence *: "x ∈ fvsstp s" using Cons.prem by simp
    hence "x ∈ fvset (trmssstp s)"
    proof (cases s)
      case (NegChecks X F G)
      hence "x ∈ fvpairs F ∨ x ∈ fvpairs G" using * by simp_all
      thus ?thesis using * fvpairs_in_fv_trmspairs[of x] NegChecks by auto
    qed auto
    thus ?thesis by simp
  qed simp
qed simp

```

```

lemma stateful_strand_step_subst_comp:
  assumes "range_vars δ ∩ set (bvarssstp x) = {}"
  shows "x ·sstp δ ∘s ∅ = (x ·sstp δ) ·sstp ∅"
proof (cases x)
  case (NegChecks X F G)
  hence *: "range_vars δ ∩ set X = {}" using assms by simp
  have "H ·pairs rm_vars (set X) (δ ∘s ∅) = (H ·pairs rm_vars (set X) δ) ·pairs rm_vars (set X) ∅" for H
    using pairs_subst_comp rm_vars_comp[OF *] by (induct H) (auto simp add: subst_apply_pairs_def)
  thus ?thesis using NegChecks by simp
qed simp_all

```

```

lemma stateful_strand_subst_comp:
  assumes "range_vars δ ∩ bvarssst S = {}"
  shows "S ·sst δ ∘s ∅ = (S ·sst δ) ·sst ∅"
using assms
proof (induction S)
  case (Cons s S)
  hence IH: "S ·sst δ ∘s ∅ = (S ·sst δ) ·sst ∅" using Cons by auto

```

```

have "s ·sstp δ ∘s ∅ = (s ·sstp δ) ·sstp ∅"
  using Cons.premis stateful_strand_step_subst_comp[of δ s ∅]
  unfolding range_vars_alt_def by auto
  thus ?case using IH by (simp add: subst_apply_stateful_strand_def)
qed simp

lemma subst_apply_bvars_disj_NegChecks:
  assumes "set X ∩ subst_domain ∅ = {}"
  shows "NegChecks X F G ·sstp ∅ = NegChecks X (F ·pairs ∅) (G ·pairs ∅)"
proof -
  have "rm_vars (set X) ∅ = ∅" using assms rm_vars_apply'[of ∅ "set X"] by auto
  thus ?thesis by simp
qed

lemma subst_apply_NegChecks_no_bvars[simp]:
  "∀ [] ⟨∇≠: F ∇≠: F'⟩ ·sstp ∅ = ∇ [] ⟨∇≠: (F ·pairs ∅) ∇≠: (F' ·pairs ∅)⟩"
  "∀ [] ⟨∇≠: [] ∇≠: F'⟩ ·sstp ∅ = ∇ [] ⟨∇≠: [] ∇≠: (F' ·pairs ∅)⟩"
  "∀ [] ⟨∇≠: F ∇≠: []⟩ ·sstp ∅ = ∇ [] ⟨∇≠: (F ·pairs ∅) ∇≠: []⟩"
  "∀ [] ⟨∇≠: [] ∇≠: [(t,s)]⟩ ·sstp ∅ = ∇ [] ⟨∇≠: [] ∇≠: [(t · ∅, s · ∅)]⟩" (is ?A)
  "∀ [] ⟨∇≠: [(t,s)] ∇≠: []⟩ ·sstp ∅ = ∇ [] ⟨∇≠: [(t · ∅, s · ∅)] ∇≠: []⟩" (is ?B)
by simp_all

lemma setops_sst_mono:
  "set M ⊆ set N ⟹ setops_sst M ⊆ setops_sst N"
by (auto simp add: setops_sst_def)

lemma setops_sst_nil[simp]: "setops_sst [] = {}"
by (simp add: setops_sst_def)

lemma setops_sst_cons[simp]: "setops_sst (a#A) = setops_sstp a ∪ setops_sst A"
by (simp add: setops_sst_def)

lemma setops_sst_cons_subset[simp]: "setops_sst A ⊆ setops_sst (a#A)"
using setops_sst_cons[of a A] by blast

lemma setops_sst_append: "setops_sst (A@B) = setops_sst A ∪ setops_sst B"
proof (induction A)
  case (Cons a A) thus ?case by (cases a) (auto simp add: setops_sst_def)
qed (simp add: setops_sst_def)

lemma setops_sstp_member_iff:
  "(t,s) ∈ setops_sstp x ⟷
    (x = Insert t s ∨ x = Delete t s ∨ (∃ ac. x = InSet ac t s) ∨
     (∃ X F F'. x = NegChecks X F F' ∧ (t,s) ∈ set F'))"
by (cases x) auto

lemma setops_sst_member_iff:
  "(t,s) ∈ setops_sst A ⟷
    (Insert t s ∈ set A ∨ Delete t s ∈ set A ∨ (∃ ac. InSet ac t s ∈ set A) ∨
     (∃ X F F'. NegChecks X F F' ∈ set A ∧ (t,s) ∈ set F'))"
  (is "?P ⟷ ?Q")
proof (induction A)
  case (Cons a A) thus ?case
  proof (cases "(t, s) ∈ setops_sstp a")
    case True thus ?thesis using setops_sstp_member_iff[of t s a] by auto
  qed auto
qed simp

lemma setops_sstp_subst:
  assumes "set (bvars_sstp a) ∩ subst_domain ∅ = {}"
  shows "setops_sstp (a ·sstp ∅) = setops_sstp a ·pset ∅"
proof (cases a)
  case (NegChecks X F G)

```

```

hence "rm_vars (set X)  $\vartheta$  =  $\vartheta$ " using assms rm_vars_apply' [of  $\vartheta$  "set X"] by auto
hence "setopssstp (a ·sstp  $\vartheta$ ) = set (G ·pairs  $\vartheta$ )"
      "setopssstp a ·pset  $\vartheta$  = set G ·pset  $\vartheta$ "
      using NegChecks image_Un by simp_all
thus ?thesis by (simp add: subst_apply_pairs_def)
qed simp_all

lemma setopssstp_subst':
  assumes "¬is_NegChecks a"
  shows "setopssstp (a ·sstp  $\vartheta$ ) = setopssstp a ·pset  $\vartheta$ "
using assms by (cases a) auto

lemma setopssstp_subst'':
  fixes t::('a,'b) term × ('a,'b) term" and  $\delta$ ::('a,'b) subst"
  assumes t: "t ∈ setopssstp (b ·sstp  $\delta$ )"
  shows "∃ s ∈ setopssstp b. t = s ·p rm_vars (set (bvarssstp b))  $\delta$ "
proof (cases "is_NegChecks b")
  case True
  then obtain X F G where b: "b = NegChecks X F G" by (cases b) moura+
  hence "setopssstp b = set G" "setopssstp (b ·sstp  $\delta$ ) = set (G ·pairs rm_vars (set (bvarssstp b))  $\delta$ )"
    by simp_all
  thus ?thesis using t subst_apply_pairs_pset_subst [of G] by blast
next
  case False
  hence "setopssstp (b ·sstp  $\delta$ ) = setopssstp b ·pset rm_vars (set (bvarssstp b))  $\delta$ "
    using setopssstp_subst' bvarssstp_NegChecks by fastforce
  thus ?thesis using t by blast
qed

lemma setopssst_subst:
  assumes "bvarssst S ∩ subst_domain  $\vartheta$  = {}"
  shows "setopssst (S ·sst  $\vartheta$ ) = setopssst S ·pset  $\vartheta$ "
using assms
proof (induction S)
  case (Cons a S)
  have "bvarssst S ∩ subst_domain  $\vartheta$  = {}" and *: "set (bvarssstp a) ∩ subst_domain  $\vartheta$  = {}"
    using Cons.premis by auto
  hence IH: "setopssst (S ·sst  $\vartheta$ ) = setopssst S ·pset  $\vartheta$ "
    using Cons.IH by auto
  show ?case
    using setopssstp_subst[OF *] IH unfolding setopssst_def
    by (auto simp add: subst_apply_stateful_strand_def)
qed (simp add: setopssst_def)

lemma setopssst_subst':
  fixes p::('a,'b) term × ('a,'b) term" and  $\delta$ ::('a,'b) subst"
  assumes "p ∈ setopssst (S ·sst  $\delta$ )"
  shows "∃ s ∈ setopssst S. ∃ X. set X ⊆ bvarssst S ∧ p = s ·p rm_vars (set X)  $\delta$ "
using assms
proof (induction S)
  case (Cons a S)
  note 0 = setopssst_cons [of a S] bvarssst_Cons [of a S]
  note 1 = setopssst_cons [of "a ·sstp  $\delta$ " "S ·sst  $\delta$ "] substsst_cons [of a S  $\delta$ ]
  have "p ∈ setopssst (S ·sst  $\delta$ ) ∨ p ∈ setopssstp (a ·sstp  $\delta$ )" using Cons.premis 1 by auto
  thus ?case
  proof
    assume *: "p ∈ setopssstp (a ·sstp  $\delta$ )"
    show ?thesis using setopssstp_subst'' [OF *] 0 by blast
  next
    assume *: "p ∈ setopssst (S ·sst  $\delta$ )"
    show ?thesis using Cons.IH [OF *] 0 by blast
  qed
qed simp

```

## 4.1.3 Stateful Constraint Semantics

```

context intruder_model
begin

definition negchecks_model where
  "negchecks_model ( $\mathcal{I}::('a,'b)$  subst) ( $D::('a,'b)$  dbstate)  $X F G \equiv$ 
    ( $\forall \delta.$  subst_domain  $\delta = \text{set } X \wedge \text{ground } (\text{subst\_range } \delta) \longrightarrow$ 
      ( $\text{list\_ex } (\lambda f. \text{fst } f \cdot (\delta \circ_s \mathcal{I}) \neq \text{snd } f \cdot (\delta \circ_s \mathcal{I})) F \vee$ 
         $\text{list\_ex } (\lambda f. f \cdot_p (\delta \circ_s \mathcal{I}) \notin D) G$ ))"

fun strand_sem_stateful::
  "('fun,'var) terms  $\Rightarrow$  ('fun,'var) dbstate  $\Rightarrow$  ('fun,'var) stateful_strand  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$ 
  bool"
  ("[[_; _; _]]_s")
where
  "[[ $M; D; []$ ]]_s = ( $\lambda \mathcal{I}. \text{True}$ )"
  | "[[ $M; D; \text{Send } t\#S$ ]]_s = ( $\lambda \mathcal{I}. M \vdash t \cdot \mathcal{I} \wedge [[M; D; S]]_s \mathcal{I}$ )"
  | "[[ $M; D; \text{Receive } t\#S$ ]]_s = ( $\lambda \mathcal{I}. [[\text{insert } (t \cdot \mathcal{I}) M; D; S]]_s \mathcal{I}$ )"
  | "[[ $M; D; \text{Equality } _ t t'\#S$ ]]_s = ( $\lambda \mathcal{I}. t \cdot \mathcal{I} = t' \cdot \mathcal{I} \wedge [[M; D; S]]_s \mathcal{I}$ )"
  | "[[ $M; D; \text{Insert } t s\#S$ ]]_s = ( $\lambda \mathcal{I}. [[M; \text{insert } ((t,s) \cdot_p \mathcal{I}) D; S]]_s \mathcal{I}$ )"
  | "[[ $M; D; \text{Delete } t s\#S$ ]]_s = ( $\lambda \mathcal{I}. [[M; D - \{(t,s) \cdot_p \mathcal{I}\}; S]]_s \mathcal{I}$ )"
  | "[[ $M; D; \text{InSet } _ t s\#S$ ]]_s = ( $\lambda \mathcal{I}. (t,s) \cdot_p \mathcal{I} \in D \wedge [[M; D; S]]_s \mathcal{I}$ )"
  | "[[ $M; D; \text{NegChecks } X F F'\#S$ ]]_s = ( $\lambda \mathcal{I}. \text{negchecks\_model } \mathcal{I} D X F F' \wedge [[M; D; S]]_s \mathcal{I}$ )"

lemmas strand_sem_stateful_induct =
  strand_sem_stateful.induct[case_names Nil ConsSnd ConsRcv ConsEq
    ConsIns ConsDel ConsIn ConsNegChecks]

abbreviation constr_sem_stateful (infix " $\models_s$ " 91) where " $\mathcal{I} \models_s A \equiv [[\{\}; \{\}; A]]_s \mathcal{I}$ "

lemma stateful_strand_sem_NegChecks_no_bvars:
  "[[ $M; D; [\langle t \text{ not in } s \rangle]$ ]]_s  $\mathcal{I} \Longrightarrow (t \cdot \mathcal{I}, s \cdot \mathcal{I}) \notin D$ "
  "[[ $M; D; [\langle t \neq s \rangle]$ ]]_s  $\mathcal{I} \Longrightarrow t \cdot \mathcal{I} \neq s \cdot \mathcal{I}$ "
by (simp_all add: negchecks_model_def empty_dom_iff_empty_subst)

lemma strand_sem_ik_mono_stateful:
  "[[ $M; D; A$ ]]_s  $\mathcal{I} \Longrightarrow [[M \cup M'; D; A]]_s \mathcal{I}$ "
using ideduct_mono by (induct A arbitrary:  $M M' D$  rule: strand_sem_stateful.induct) force+

lemma strand_sem_append_stateful:
  "[[ $M; D; A \oplus B$ ]]_s  $\mathcal{I} \longleftrightarrow [[M; D; A]]_s \mathcal{I} \wedge [[M \cup (\text{ik}_{sst} A \cdot_{set} \mathcal{I}); \text{dbupd}_{sst} A \mathcal{I} D; B]]_s \mathcal{I}$ "
  (is "?P  $\longleftrightarrow$  ?Q  $\wedge$  ?R")
proof -
  have 1: "?P  $\Longrightarrow$  ?Q" by (induct A rule: strand_sem_stateful.induct) auto

  have 2: "?P  $\Longrightarrow$  ?R"
  proof (induction A arbitrary:  $M D B$ )
    case (Cons a A) thus ?case
    proof (cases a)
      case (Receive t)
      have "insert (t  $\cdot$   $\mathcal{I}$ ) ( $M \cup (\text{ik}_{sst} A \cdot_{set} \mathcal{I})$ ) =  $M \cup (\text{ik}_{sst} (a\#A) \cdot_{set} \mathcal{I})$ "
      "dbupdsst A  $\mathcal{I} D$  = dbupdsst (a#A)  $\mathcal{I} D$ "
      using Receive by (auto simp add: iksst_def)
      thus ?thesis using Cons Receive by force
    qed (auto simp add: iksst_def)
  qed (simp add: iksst_def)

  have 3: "?Q  $\Longrightarrow$  ?R  $\Longrightarrow$  ?P"
  proof (induction A arbitrary:  $M D$ )
    case (Cons a A) thus ?case
    proof (cases a)

```

```

case (Receive t)
have "insert (t · I) (M ∪ (iksst A ·set I)) = M ∪ (iksst (a#A) ·set I)"
  "dbupdsst A I D = dbupdsst (a#A) I D"
  using Receive by (auto simp add: iksst_def)
  thus ?thesis using Cons Receive by simp
qed (auto simp add: iksst_def)
qed (simp add: iksst_def)

show ?thesis by (metis 1 2 3)
qed

lemma negchecks_model_db_subset:
  fixes F F'::"('a,'b) term × ('a,'b) term) list"
  assumes "D' ⊆ D"
  and "negchecks_model I D X F F'"
  shows "negchecks_model I D' X F F'"
proof -
  have "list_ex (λf. f ·p δ ∘s I ∉ D') F'"
  when "list_ex (λf. f ·p δ ∘s I ∉ D) F'"
  for δ::"('a,'b) subst"
  using Bex_set[of F' "λf. f ·p δ ∘s I ∉ D'"]
    Bex_set[of F' "λf. f ·p δ ∘s I ∉ D"]
    that assms(1)
  by blast
  thus ?thesis using assms(2) by (auto simp add: negchecks_model_def)
qed

lemma negchecks_model_db_supset:
  fixes F F'::"('a,'b) term × ('a,'b) term) list"
  assumes "D' ⊆ D"
  and "∀f ∈ set F'. ∀δ. subst_domain δ = set X ∧ ground (subst_range δ) → f ·p (δ ∘s I) ∉ D - D'"
  and "negchecks_model I D' X F F'"
  shows "negchecks_model I D X F F'"
proof -
  have "list_ex (λf. f ·p δ ∘s I ∉ D) F'"
  when "list_ex (λf. f ·p δ ∘s I ∉ D') F'" "subst_domain δ = set X ∧ ground (subst_range δ)"
  for δ::"('a,'b) subst"
  using Bex_set[of F' "λf. f ·p δ ∘s I ∉ D'"]
    Bex_set[of F' "λf. f ·p δ ∘s I ∉ D"]
    that assms(1,2)
  by blast
  thus ?thesis using assms(3) by (auto simp add: negchecks_model_def)
qed

lemma negchecks_model_subst:
  fixes F F'::"('a,'b) term × ('a,'b) term) list"
  assumes "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
  shows "negchecks_model (δ ∘s ∅) D X F F' ↔ negchecks_model ∅ D X (F ·pairs δ) (F' ·pairs δ)"
proof -
  have 0: "σ ∘s (δ ∘s ∅) = δ ∘s (σ ∘s ∅)"
  when σ: "subst_domain σ = set X" "ground (subst_range σ)" for σ
  by (metis (no_types, lifting) σ subst_compose_assoc assms(1) inf_sup_aci(1)
    subst_comp_eq_if_disjoint_vars sup_inf_absorb range_vars_alt_def)

  { fix σ::"('a,'b) subst" and t t'
    assume σ: "subst_domain σ = set X" "ground (subst_range σ)"
    and *: "list_ex (λf. fst f · (σ ∘s (δ ∘s ∅)) ≠ snd f · (σ ∘s (δ ∘s ∅))) F"
    obtain f where f: "f ∈ set F" "fst f · σ ∘s (δ ∘s ∅) ≠ snd f · σ ∘s (δ ∘s ∅)"
    using * by (induct F) auto
    hence "(fst f · δ) · σ ∘s ∅ ≠ (snd f · δ) · σ ∘s ∅" using 0[OF σ] by simp
    moreover have "(fst f · δ, snd f · δ) ∈ set (F ·pairs δ)"
    using f(1) by (auto simp add: subst_apply_pairs_def)
  }

```



```

ultimately have "list_ex (λf. fst f · (σ ∘s ϑ) ≠ snd f · (σ ∘s ϑ)) (F ·pairs δ)"
  using f(1) Bex_set by fastforce
} moreover {
  fix σ::('a,'b) subst" and t t'
  assume σ: "subst_domain σ = set X" "ground (subst_range σ)"
  and *: "list_ex (λf. f ·p σ ∘s (δ ∘s ϑ) ∉ D) F'"
  obtain f where f: "f ∈ set F'" "f ·p σ ∘s (δ ∘s ϑ) ∉ D"
  using * by (induct F') auto
  hence "f ·p δ ·p σ ∘s ϑ ∉ D" using 0[OF σ] by (metis subst_pair_compose)
  moreover have "f ·p δ ∈ set (F' ·pairs δ)"
  using f(1) by (auto simp add: subst_apply_pairs_def)
  ultimately have "list_ex (λf. f ·p σ ∘s ϑ ∉ D) (F' ·pairs δ)"
  using f(1) Bex_set by fastforce
} moreover {
  fix σ::('a,'b) subst" and t t'
  assume σ: "subst_domain σ = set X" "ground (subst_range σ)"
  and *: "list_ex (λf. fst f · (σ ∘s ϑ) ≠ snd f · (σ ∘s ϑ)) (F ·pairs δ)"
  obtain f where f: "f ∈ set (F ·pairs δ)" "fst f · σ ∘s ϑ ≠ snd f · σ ∘s ϑ"
  using * by (induct F) (auto simp add: subst_apply_pairs_def)
  then obtain g where g: "g ∈ set F" "f = g ·p δ" by (auto simp add: subst_apply_pairs_def)
  have "fst g · σ ∘s (δ ∘s ϑ) ≠ snd g · σ ∘s (δ ∘s ϑ)"
  using f(2) g 0[OF σ] by (simp add: prod.case_eq_if)
  hence "list_ex (λf. fst f · (σ ∘s (δ ∘s ϑ)) ≠ snd f · (σ ∘s (δ ∘s ϑ))) F"
  using g Bex_set by fastforce
} moreover {
  fix σ::('a,'b) subst" and t t'
  assume σ: "subst_domain σ = set X" "ground (subst_range σ)"
  and *: "list_ex (λf. f ·p (σ ∘s ϑ) ∉ D) (F' ·pairs δ)"
  obtain f where f: "f ∈ set (F' ·pairs δ)" "f ·p σ ∘s ϑ ∉ D"
  using * by (induct F') (auto simp add: subst_apply_pairs_def)
  then obtain g where g: "g ∈ set F'" "f = g ·p δ" by (auto simp add: subst_apply_pairs_def)
  have "g ·p σ ∘s (δ ∘s ϑ) ∉ D"
  using f(2) g 0[OF σ] by (simp add: prod.case_eq_if)
  hence "list_ex (λf. f ·p (σ ∘s (δ ∘s ϑ)) ∉ D) F'"
  using g Bex_set by fastforce
} ultimately show ?thesis using assms unfolding negchecks_model_def by blast
qed

```

lemma strand\_sem\_subst\_stateful:

```

fixes δ::('fun,'var) subst"
assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarssst S = {}"
shows "⌊M; D; S⌋s (δ ∘s ϑ) ⟷ ⌊M; D; S ·sst δ⌋s ϑ"

```

proof

```

note [simp] = subst_sst_cons[of _ _ δ] subst_subst_compose[of _ δ ϑ]

```

```

have "(subst_domain δ ∪ range_vars δ) ∩ (subst_domain γ ∪ range_vars γ) = {}"

```

```

  when δ: "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
  and γ: "subst_domain γ = set X" "ground (subst_range γ)"
  for X and γ::('fun,'var) subst"

```

```

  using δ γ unfolding range_vars_alt_def by auto

```

```

hence 0: "γ ∘s δ = δ ∘s γ"

```

```

  when δ: "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
  and γ: "subst_domain γ = set X" "ground (subst_range γ)"
  for γ X

```

```

  by (metis δ γ subst_comp_eq_if_disjoint_vars)

```

```

show "⌊M; D; S⌋s (δ ∘s ϑ) ⟹ ⌊M; D; S ·sst δ⌋s ϑ" using assms

```

```

proof (induction S arbitrary: M D rule: strand_sem_stateful_induct)

```

```

  case (ConsNegChecks M D X F F' S)

```

```

  hence *: "⌊M; D; S ·sst δ⌋s ϑ" and **: "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"

```

```

  unfolding bvarssst_def negchecks_model_def by (force, auto)

```

```

  have "negchecks_model (δ ∘s ϑ) D X F F'" using ConsNegChecks by auto

```

```

  hence "negchecks_model ϑ D X (F ·pairs δ) (F' ·pairs δ)"

```

```

    using 0[OF **] negchecks_model_subst[OF **] by blast
    moreover have "rm_vars (set X)  $\delta = \delta$ " using ConsNegChecks.prem(2) by force
    ultimately show ?case using * by auto
qed simp_all

show "[M; D; S ·sst  $\delta$ ]s  $\vartheta \implies [M; D; S]s (\delta \circ_s \vartheta)" using assms
proof (induction S arbitrary: M D rule: strand_sem_stateful_induct)
  case (ConsNegChecks M D X F F' S)
  have  $\delta$ : "rm_vars (set X)  $\delta = \delta$ " using ConsNegChecks.prem(2) by force
  hence *: "[M; D; S]s (\delta \circ_s \vartheta)" and **: "(subst_domain  $\delta \cup \text{range\_vars } \delta) \cap \text{set } X = \{\}$ "
    using ConsNegChecks unfolding bvarssst_def negchecks_model_def by auto
  have "negchecks_model  $\vartheta$  D X (F ·pairs  $\delta$ ) (F' ·pairs  $\delta$ )"
    using ConsNegChecks.prem(1)  $\delta$  by (auto simp add: subst_compose_assoc negchecks_model_def)
  hence "negchecks_model (\delta \circ_s \vartheta) D X F F'"
    using 0[OF **] negchecks_model_subst[OF **] by blast
  thus ?case using * by auto
qed simp_all
qed

end$ 
```

#### 4.1.4 Well-Formedness Lemmata

```

lemma wfvarsoccssst_subset_wfrestrictedvarssst[simp]:
  "wfvarsoccssst S  $\subseteq$  wfrestrictedvarssst S"
by (induction S)
  (auto simp add: wfrestrictedvarssst_def wfvarsoccssst_def
    split: stateful_strand_step.split poscheckvariant.split)

lemma wfvarsoccssst_append: "wfvarsoccssst (S@S') = wfvarsoccssst S  $\cup$  wfvarsoccssst S'"
by (simp add: wfvarsoccssst_def)

lemma wfrestrictedvarssst_union[simp]:
  "wfrestrictedvarssst (S@T) = wfrestrictedvarssst S  $\cup$  wfrestrictedvarssst T"
by (simp add: wfrestrictedvarssst_def)

lemma wfrestrictedvarssst_singleton:
  "wfrestrictedvarssst [s] = wfrestrictedvarssstp s"
by (simp add: wfrestrictedvarssst_def)

lemma wfsst_prefix[dest]: "wf'sst V (S@S')  $\implies$  wf'sst V S"
by (induct S rule: wf'sst.induct) auto

lemma wfsst_vars_mono: "wf'sst V S  $\implies$  wf'sst (V  $\cup$  W) S"
proof (induction S arbitrary: V)
  case (Cons x S) thus ?case
  proof (cases x)
    case (Send t)
    hence "wf'sst (V  $\cup$  fv t  $\cup$  W) S" using Cons.prem(1) Cons.IH by simp
    thus ?thesis using Send by (simp add: sup_commute sup_left_commute)
  next
    case (Equality a t t')
    show ?thesis
    proof (cases a)
      case Assign
      hence "wf'sst (V  $\cup$  fv t  $\cup$  W) S" "fv t'  $\subseteq$  V  $\cup$  W" using Equality Cons.prem(1) Cons.IH by auto
      thus ?thesis using Equality Assign by (simp add: sup_commute sup_left_commute)
    next
      case Check
      thus ?thesis using Equality Cons by auto
    qed
  next
    case (InSet a t t')
    show ?thesis

```

```

proof (cases a)
  case Assign
  hence "wf'_{sst} (V ∪ fv t ∪ fv t' ∪ W) S" using InSet Cons.prem1 Cons.IH by auto
  thus ?thesis using InSet Assign by (simp add: sup_commute sup_left_commute)
next
  case Check thus ?thesis using InSet Cons by auto
qed
qed auto
qed simp

```

```

lemma wf_{sst}I[intro]: "wfrestrictedvars_{sst} S ⊆ V ⇒ wf'_{sst} V S"
proof (induction S)
  case (Cons x S) thus ?case
  proof (cases x)
    case (Send t)
    hence "wf'_{sst} V S" "V ∪ fv t = V"
    using Cons
    unfolding wfrestrictedvars_{sst}_def
    by auto
    thus ?thesis using Send by simp
  next
    case (Equality a t t')
    show ?thesis
    proof (cases a)
      case Assign
      hence "wf'_{sst} V S" "fv t' ⊆ V"
      using Equality Cons
      unfolding wfrestrictedvars_{sst}_def
      by auto
      thus ?thesis using wf_{sst}_vars_mono Equality Assign by simp
    next
      case Check
      thus ?thesis
      using Equality Cons
      unfolding wfrestrictedvars_{sst}_def
      by auto
    qed
  next
    case (InSet a t t')
    show ?thesis
    proof (cases a)
      case Assign
      hence "wf'_{sst} V S" "fv t ∪ fv t' ⊆ V"
      using InSet Cons
      unfolding wfrestrictedvars_{sst}_def
      by auto
      thus ?thesis using wf_{sst}_vars_mono InSet Assign by (simp add: Un_assoc)
    next
      case Check
      thus ?thesis
      using InSet Cons
      unfolding wfrestrictedvars_{sst}_def
      by auto
    qed
  qed (simp_all add: wfrestrictedvars_{sst}_def)
qed (simp add: wfrestrictedvars_{sst}_def)

```

```

lemma wf_{sst}I'[intro]:
  assumes "⋃ ((λx. case x of
    | Receive t ⇒ fv t
    | Equality Assign _ t' ⇒ fv t'
    | Insert t t' ⇒ fv t ∪ fv t'
    | _ ⇒ {}) ' set S) ⊆ V"

```

```

shows "wf'_{sst} V S"
using assms
proof (induction S)
  case (Cons x S) thus ?case
  proof (cases x)
    case (Equality a t t')
    thus ?thesis using Cons by (cases a) (auto simp add: wf_{sst}_vars_mono)
  next
    case (InSet a t t')
    thus ?thesis using Cons by (cases a) (auto simp add: wf_{sst}_vars_mono Un_assoc)
  qed (simp_all add: wf_{sst}_vars_mono)
qed simp

lemma wf_{sst}_append_exec: "wf'_{sst} V (S@S')  $\implies$  wf'_{sst} (V  $\cup$  wfvarsoccs_{sst} S) S'"
proof (induction S arbitrary: V)
  case (Cons x S V) thus ?case
  proof (cases x)
    case (Send t)
    hence "wf'_{sst} (V  $\cup$  fv t  $\cup$  wfvarsoccs_{sst} S) S'" using Cons.premis Cons.IH by simp
    thus ?thesis using Send unfolding wfvarsoccs_{sst}_def by (auto simp add: sup_assoc)
  next
    case (Equality a t t') show ?thesis
    proof (cases a)
      case Assign
      hence "wf'_{sst} (V  $\cup$  fv t  $\cup$  wfvarsoccs_{sst} S) S'" using Equality Cons.premis Cons.IH by auto
      thus ?thesis using Equality Assign unfolding wfvarsoccs_{sst}_def by (auto simp add: sup_assoc)
    next
      case Check
      hence "wf'_{sst} (V  $\cup$  wfvarsoccs_{sst} S) S'" using Equality Cons.premis Cons.IH by auto
      thus ?thesis using Equality Check unfolding wfvarsoccs_{sst}_def by (auto simp add: sup_assoc)
    qed
  next
    case (InSet a t t') show ?thesis
    proof (cases a)
      case Assign
      hence "wf'_{sst} (V  $\cup$  fv t  $\cup$  fv t'  $\cup$  wfvarsoccs_{sst} S) S'" using InSet Cons.premis Cons.IH by auto
      thus ?thesis using InSet Assign unfolding wfvarsoccs_{sst}_def by (auto simp add: sup_assoc)
    next
      case Check
      hence "wf'_{sst} (V  $\cup$  wfvarsoccs_{sst} S) S'" using InSet Cons.premis Cons.IH by auto
      thus ?thesis using InSet Check unfolding wfvarsoccs_{sst}_def by (auto simp add: sup_assoc)
    qed
  qed (auto simp add: wfvarsoccs_{sst}_def)
qed (simp add: wfvarsoccs_{sst}_def)

lemma wf_{sst}_append:
  "wf'_{sst} X S  $\implies$  wf'_{sst} Y T  $\implies$  wf'_{sst} (X  $\cup$  Y) (S@T)"
proof (induction X S rule: wf'_{sst}.induct)
  case 1 thus ?case by (metis wf_{sst}_vars_mono Un_commute append_Nil)
next
  case 3 thus ?case by (metis append_Cons Un_commute Un_assoc wf'_{sst}.simps(3))
next
  case (4 V t t' S)
  hence *: "fv t'  $\subseteq$  V" and "wf'_{sst} (V  $\cup$  fv t  $\cup$  Y) (S @ T)" by simp_all
  hence "wf'_{sst} (V  $\cup$  Y  $\cup$  fv t) (S @ T)" by (metis Un_commute Un_assoc)
  thus ?case using * by auto
next
  case (8 V t t' S)
  hence "wf'_{sst} (V  $\cup$  fv t  $\cup$  fv t'  $\cup$  Y) (S @ T)" by simp_all
  hence "wf'_{sst} (V  $\cup$  Y  $\cup$  fv t  $\cup$  fv t') (S @ T)" by (metis Un_commute Un_assoc)
  thus ?case by auto
qed auto

```

```

lemma wfsst_append_suffix:
  "wf'sst V S  $\implies$  wfrestrictedvarssst S'  $\subseteq$  wfrestrictedvarssst S  $\cup$  V  $\implies$  wf'sst V (S@S')"
proof (induction V S rule: wf'sst.induct)
  case (2 V t S)
  hence *: "fv t  $\subseteq$  V" "wf'sst V S" by simp_all
  hence "wfrestrictedvarssst S'  $\subseteq$  wfrestrictedvarssst S  $\cup$  V"
    using "2.prem"(2) unfolding wfrestrictedvarssst_def by auto
  thus ?case using "2.IH" * by simp
next
  case (3 V t S)
  hence *: "wf'sst (V  $\cup$  fv t) S" by simp_all
  hence "wfrestrictedvarssst S'  $\subseteq$  wfrestrictedvarssst S  $\cup$  (V  $\cup$  fv t)"
    using "3.prem"(2) unfolding wfrestrictedvarssst_def by auto
  thus ?case using "3.IH" * by simp
next
  case (4 V t t' S)
  hence *: "fv t'  $\subseteq$  V" "wf'sst (V  $\cup$  fv t) S" by simp_all
  moreover have "varssstp ( $\langle t := t' \rangle$ ) = fv t  $\cup$  fv t'"
    by simp
  moreover have "wfrestrictedvarssst ( $\langle t := t' \rangle$ #S) = fv t  $\cup$  fv t'  $\cup$  wfrestrictedvarssst S"
    unfolding wfrestrictedvarssst_def by auto
  ultimately have "wfrestrictedvarssst S'  $\subseteq$  wfrestrictedvarssst S  $\cup$  (V  $\cup$  fv t)"
    using "4.prem"(2) by blast
  thus ?case using "4.IH" * by simp
next
  case (6 V t t' S)
  hence *: "fv t  $\cup$  fv t'  $\subseteq$  V" "wf'sst V S" by simp_all
  moreover have "varssstp (insert $\langle t, t' \rangle$ ) = fv t  $\cup$  fv t'"
    by simp
  moreover have "wfrestrictedvarssst (insert $\langle t, t' \rangle$ #S) = fv t  $\cup$  fv t'  $\cup$  wfrestrictedvarssst S"
    unfolding wfrestrictedvarssst_def by auto
  ultimately have "wfrestrictedvarssst S'  $\subseteq$  wfrestrictedvarssst S  $\cup$  V"
    using "6.prem"(2) by blast
  thus ?case using "6.IH" * by simp
next
  case (8 V t t' S)
  hence *: "wf'sst (V  $\cup$  fv t  $\cup$  fv t') S" by simp_all
  moreover have "varssstp (select $\langle t, t' \rangle$ ) = fv t  $\cup$  fv t'"
    by simp
  moreover have "wfrestrictedvarssst (select $\langle t, t' \rangle$ #S) = fv t  $\cup$  fv t'  $\cup$  wfrestrictedvarssst S"
    unfolding wfrestrictedvarssst_def by auto
  ultimately have "wfrestrictedvarssst S'  $\subseteq$  wfrestrictedvarssst S  $\cup$  (V  $\cup$  fv t  $\cup$  fv t')"
    using "8.prem"(2) by blast
  thus ?case using "8.IH" * by simp
qed (simp_all add: wfsstI wfrestrictedvarssst_def)

lemma wfsst_append_suffix':
  assumes "wf'sst V S"
  and " $\bigcup ((\lambda x. \text{case } x \text{ of}$ 
    Receive t  $\Rightarrow$  fv t
    | Equality Assign _ t'  $\Rightarrow$  fv t'
    | Insert t t'  $\Rightarrow$  fv t  $\cup$  fv t'
    | _  $\Rightarrow$  {}) ' set S')  $\subseteq$  wfvarsoccssst S  $\cup$  V"
  shows "wf'sst V (S@S)"
using assms
by (induction V S rule: wf'sst.induct)
  (auto simp add: wfsstI' wfsst-vars_mono wfvarsoccssst_def)

lemma wfsst_subst_apply:
  "wf'sst V S  $\implies$  wf'sst (fvset ( $\delta$  ' V)) (S  $\cdot$ sst  $\delta$ )"
proof (induction S arbitrary: V rule: wf'sst.induct)
  case (2 V t S)
  hence "wf'sst V S" "fv t  $\subseteq$  V" by simp_all

```

```

hence "wf'_{sst} (fv_{set} (\delta ' V)) (S \cdot_{sst} \delta)" "fv (t \cdot \delta) \subseteq fv_{set} (\delta ' V)"
  using "2.IH" subst_apply_fv_subset by simp_all
thus ?case by (simp add: subst_apply_stateful_strand_def)
next
case (3 V t S)
hence "wf'_{sst} (V \cup fv t) S" by simp
hence "wf'_{sst} (fv_{set} (\delta ' (V \cup fv t))) (S \cdot_{sst} \delta)" using "3.IH" by metis
hence "wf'_{sst} (fv_{set} (\delta ' V) \cup fv (t \cdot \delta)) (S \cdot_{sst} \delta)" by (metis subst_apply_fv_union)
thus ?case by (simp add: subst_apply_stateful_strand_def)
next
case (4 V t t' S)
hence "wf'_{sst} (V \cup fv t) S" "fv t' \subseteq V" by auto
hence "wf'_{sst} (fv_{set} (\delta ' (V \cup fv t))) (S \cdot_{sst} \delta)" and *: "fv (t' \cdot \delta) \subseteq fv_{set} (\delta ' V)"
  using "4.IH" subst_apply_fv_subset by force+
hence "wf'_{sst} (fv_{set} (\delta ' V) \cup fv (t \cdot \delta)) (S \cdot_{sst} \delta)" by (metis subst_apply_fv_union)
thus ?case using * by (simp add: subst_apply_stateful_strand_def)
next
case (6 V t t' S)
hence "wf'_{sst} V S" "fv t \cup fv t' \subseteq V" by auto
hence "wf'_{sst} (fv_{set} (\delta ' V)) (S \cdot_{sst} \delta)" "fv (t \cdot \delta) \subseteq fv_{set} (\delta ' V)" "fv (t' \cdot \delta) \subseteq fv_{set} (\delta ' V)"
  using "6.IH" subst_apply_fv_subset by force+
thus ?case by (simp add: sup_assoc subst_apply_stateful_strand_def)
next
case (8 V t t' S)
hence "wf'_{sst} (V \cup fv t \cup fv t') S" by auto
hence "wf'_{sst} (fv_{set} (\delta ' (V \cup fv t \cup fv t'))) (S \cdot_{sst} \delta)"
  using "8.IH" subst_apply_fv_subset by force
hence "wf'_{sst} (fv_{set} (\delta ' V) \cup fv (t \cdot \delta) \cup fv (t' \cdot \delta)) (S \cdot_{sst} \delta)" by (metis subst_apply_fv_union)
thus ?case by (simp add: subst_apply_stateful_strand_def)
qed (auto simp add: subst_apply_stateful_strand_def)

end

```

## 4.2 Extending the Typing Result to Stateful Constraints (Stateful Typing)

```

theory Stateful_Typing
imports Typing_Result Stateful_Strands
begin

  Locale setup

  locale stateful_typed_model = typed_model arity public Ana \Gamma
    for arity::"'fun \Rightarrow nat"
      and public::"'fun \Rightarrow bool"
      and Ana::"('fun,'var) term \Rightarrow (('fun,'var) term list \times ('fun,'var) term list)"
      and \Gamma::"('fun,'var) term \Rightarrow ('fun,'atom::finite) term_type"
  +
  fixes Pair::"'fun"
  assumes Pair_arity: "arity Pair = 2"
  and Ana_subst': "\f T \delta K M. Ana (Fun f T) = (K,M) \Longrightarrow Ana (Fun f T \cdot \delta) = (K \cdot_{list} \delta, M \cdot_{list} \delta)"
begin

  lemma Ana_invar_subst' [simp]: "Ana_invar_subst S"
  using Ana_subst' unfolding Ana_invar_subst_def by force

  definition pair where
    "pair d \equiv case d of (t,t') \Rightarrow Fun Pair [t,t']"

  fun tr_pairs::
    " (('fun,'var) term \times ('fun,'var) term) list \Rightarrow
      ('fun,'var) dbstatelist \Rightarrow
      (('fun,'var) term \times ('fun,'var) term) list list"

```

where

```
"tr_pairs [] D = [[]]"
| "tr_pairs ((s,t)#F) D =
  concat (map (λd. map ((#) (pair (s,t), pair d)) (tr_pairs F D)) D)"
```

A translation/reduction  $tr$  from stateful constraints to (lists of) "non-stateful" constraints. The output represents a finite disjunction of constraints whose models constitute exactly the models of the input constraint. The typing result for "non-stateful" constraints is later lifted to the stateful setting through this reduction procedure.

```
fun tr::("fun,'var) stateful_strand ⇒ ('fun,'var) dbstatelist ⇒ ('fun,'var) strand list"
```

where

```
"tr [] D = [[]]"
| "tr (send⟨t⟩#A) D = map ((#) (send⟨t⟩st)) (tr A D)"
| "tr (receive⟨t⟩#A) D = map ((#) (receive⟨t⟩st)) (tr A D)"
| "tr (⟨ac: t ≐ t'⟩#A) D = map ((#) (⟨ac: t ≐ t'⟩st)) (tr A D)"
| "tr (insert⟨t,s⟩#A) D = tr A (List.insert (t,s) D)"
| "tr (delete⟨t,s⟩#A) D =
  concat (map (λDi. map (λB. (map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di)@
    (map (λd. ∀ [] ⟨≠: [(pair (t,s), pair d)]⟩st) [d←D. d ∉ set Di])@B)
    (tr A [d←D. d ∉ set Di])))
    (subseqs D)))"
| "tr (⟨ac: t ∈ s⟩#A) D =
  concat (map (λB. map (λd. ⟨ac: (pair (t,s)) ≐ (pair d)⟩st#B) D) (tr A D))"
| "tr (∀X⟨≠: F ∨ ≠: F'⟩#A) D =
  map ((@) (map (λG. ∀X⟨≠: (F@G)⟩st) (tr_pairs F' D))) (tr A D)"
```

Type-flaw resistance of stateful constraint steps

```
fun tfr_sstp where
```

```
"tfr_sstp (Equality _ t t') = ((∃δ. Unifier δ t t') → Γ t = Γ t')"
| "tfr_sstp (NegChecks X F F') = (
  (F' = [] ∧ (∀x ∈ fv_pairs F-set X. ∃a. Γ (Var x) = TAtom a)) ∨
  (∀f T. Fun f T ∈ subterms_set (trms_pairs F ∪ pair ' set F') →
    T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X)))"
| "tfr_sstp _ = True"
```

Type-flaw resistance of stateful constraints

```
definition tfr_sst where "tfr_sst S ≡ tfr_set (trms_sst S ∪ pair ' setops_sst S) ∧ list_all tfr_sstp S"
```

### 4.2.1 Small Lemmata

lemma pair\_in\_pair\_image\_iff:

```
"pair (s,t) ∈ pair ' P ↔ (s,t) ∈ P"
```

unfolding pair\_def by fast

lemma subst\_apply\_pairs\_pair\_image\_subst:

```
"pair ' set (F ·pairs ∅) = pair ' set F ·set ∅"
```

unfolding subst\_apply\_pairs\_def pair\_def by (induct F) auto

lemma Ana\_subst\_subterms\_cases:

```
fixes ∅::("fun,'var) subst"
```

```
assumes t: "t ∈ subterms_set (M ·set ∅)"
```

```
and s: "s ∈ set (snd (Ana t))"
```

```
shows "(∃u ∈ subterms_set M. t = u · ∅ ∧ s ∈ set (snd (Ana u)) ·set ∅) ∨ (∃x ∈ fv_set M. t ⊆ ∅ x)"
```

proof (cases "t ∈ subterms\_set M ·<sub>set</sub> ∅")

case True

then obtain u where u: "u ∈ subterms\_set M" "t = u · ∅" by mouna

show ?thesis

proof (cases u)

case (Var x)

hence "x ∈ fv\_set M" using fv\_subset\_subterms[OF u(1)] by simp

thus ?thesis using u(2) Var by fastforce

next

```

case (Fun f T)
hence "set (snd (Ana t)) = set (snd (Ana u)) ·set ∅"
  using Ana_subst'[of f T _ ∅] u(2) by (cases "Ana u") auto
thus ?thesis using s u by blast
qed
qed (use s t subtermsset_subst in blast)

lemma tfrsstp_alt_def:
  "list_all tfrsstp S =
    ((∀ ac t t'. Equality ac t t' ∈ set S ∧ (∃ δ. Unifier δ t t') → Γ t = Γ t') ∧
    (∀ X F F'. NegChecks X F F' ∈ set S → (
      (F' = [] ∧ (∀ x ∈ fvpairs F-set X. ∃ a. Γ (Var x) = TAtom a)) ∨
      (∀ f T. Fun f T ∈ subtermsset (trmspairs F ∪ pair ' set F') →
        T = [] ∨ (∃ s ∈ set T. s ∉ Var ' set X))))))"
  (is "?P S = ?Q S")
proof
  show "?P S ⇒ ?Q S"
  proof (induction S)
    case (Cons x S) thus ?case by (cases x) auto
  qed simp

  show "?Q S ⇒ ?P S"
  proof (induction S)
    case (Cons x S) thus ?case by (cases x) auto
  qed simp
qed

lemma fun_pair_eq[dest]: "pair d = pair d' ⇒ d = d'"
proof -
  obtain t s t' s' where "d = (t,s)" "d' = (t',s')" by moura
  thus "pair d = pair d' ⇒ d = d'" unfolding pair_def by simp
qed

lemma fun_pair_subst: "pair d · δ = pair (d ·p δ)"
using surj_pair[of d] unfolding pair_def by force

lemma fun_pair_subst_set: "pair ' M ·set δ = pair ' (M ·pset δ)"
proof
  show "pair ' M ·set δ ⊆ pair ' (M ·pset δ)"
    using fun_pair_subst[of _ δ] by fastforce

  show "pair ' (M ·pset δ) ⊆ pair ' M ·set δ"
  proof
    fix t assume t: "t ∈ pair ' (M ·pset δ)"
    then obtain p where p: "p ∈ M" "t = pair (p ·p δ)" by blast
    thus "t ∈ pair ' M ·set δ" using fun_pair_subst[of p δ] by force
  qed
qed

lemma fun_pair_eq_subst: "pair d · δ = pair d' · ∅ ⇔ d ·p δ = d' ·p ∅"
by (metis fun_pair_subst fun_pair_eq[of "d ·p δ" "d' ·p ∅"])

lemma setopssst_pair_image_cons[simp]:
  "pair ' setopssst (x#S) = pair ' setopssstp x ∪ pair ' setopssst S"
  "pair ' setopssst (send⟨t⟩#S) = pair ' setopssst S"
  "pair ' setopssst (receive⟨t⟩#S) = pair ' setopssst S"
  "pair ' setopssst ((ac: t ≐ t')#S) = pair ' setopssst S"
  "pair ' setopssst (insert⟨t,s⟩#S) = {pair (t,s)} ∪ pair ' setopssst S"
  "pair ' setopssst (delete⟨t,s⟩#S) = {pair (t,s)} ∪ pair ' setopssst S"
  "pair ' setopssst ((ac: t ∈ s)#S) = {pair (t,s)} ∪ pair ' setopssst S"
  "pair ' setopssst (∀X(∀≠: F ∨≠: G)#S) = pair ' set G ∪ pair ' setopssst S"
unfolding setopssst_def by auto

```



```

lemma setopssst_pair_image_subst_cons[simp]:
  "pair ' setopssst (x#S ·sst ∅) = pair ' setopssstp (x ·sstp ∅) ∪ pair ' setopssst (S ·sst ∅)"
  "pair ' setopssst (send⟨t⟩#S ·sst ∅) = pair ' setopssst (S ·sst ∅)"
  "pair ' setopssst (receive⟨t⟩#S ·sst ∅) = pair ' setopssst (S ·sst ∅)"
  "pair ' setopssst ((ac: t ≐ t')#S ·sst ∅) = pair ' setopssst (S ·sst ∅)"
  "pair ' setopssst (insert⟨t,s⟩#S ·sst ∅) = {pair (t,s) · ∅} ∪ pair ' setopssst (S ·sst ∅)"
  "pair ' setopssst (delete⟨t,s⟩#S ·sst ∅) = {pair (t,s) · ∅} ∪ pair ' setopssst (S ·sst ∅)"
  "pair ' setopssst ((ac: t ∈ s)#S ·sst ∅) = {pair (t,s) · ∅} ∪ pair ' setopssst (S ·sst ∅)"
  "pair ' setopssst (∀X(∀≠: F ∨≠: G)#S ·sst ∅) =
    pair ' set (G ·pairs rm_vars (set X) ∅) ∪ pair ' setopssst (S ·sst ∅)"
using substsst_cons[of _ S ∅] unfolding setopssst_def pair_def by auto

lemma setopssst_are_pairs: "t ∈ pair ' setopssst A ⟹ ∃ s s'. t = pair (s,s')"
proof (induction A)
  case (Cons a A) thus ?case
    by (cases a) (auto simp add: setopssst_def)
qed (simp add: setopssst_def)

lemma fun_pair_wftrm: "wftrm t ⟹ wftrm t' ⟹ wftrm (pair (t,t'))"
using Pair_arity unfolding wftrm_def pair_def by auto

lemma wftrms_pairs: "wftrms (trmspairs F) ⟹ wftrms (pair ' set F)"
using fun_pair_wftrm by blast

lemma tfrsst_Nil[simp]: "tfrsst []"
by (simp add: tfrsst_def setopssst_def)

lemma tfrsst_append: "tfrsst (A@B) ⟹ tfrsst A"
proof -
  assume assms: "tfrsst (A@B)"
  let ?M = "trmssst A ∪ pair ' setopssst A"
  let ?N = "trmssst (A@B) ∪ pair ' setopssst (A@B)"
  let ?P = "λt t'. ∀x ∈ fv t ∪ fv t'. ∃a. Γ (Var x) = Var a"
  let ?Q = "λX t t'. X = [] ∨ (∀x ∈ (fv t ∪ fv t')-set X. ∃a. Γ (Var x) = Var a)"
  have *: "SMP ?M - Var'V ⊆ SMP ?N - Var'V" "?M ⊆ ?N"
    using SMP_mono[of ?M ?N] setopssst_append[of A B]
    by auto
  { fix s t assume **: "tfrset ?N" "s ∈ SMP ?M - Var'V" "t ∈ SMP ?M - Var'V" "(∃δ. Unifier δ s t)"
    hence "s ∈ SMP ?N - Var'V" "t ∈ SMP ?N - Var'V" using * by auto
    hence "Γ s = Γ t" using **(1,4) unfolding tfrset_def by blast
  } moreover have "∀t ∈ ?N. wftrm t ⟹ ∀t ∈ ?M. wftrm t" using * by blast
  ultimately have "tfrset ?N ⟹ tfrset ?M" unfolding tfrset_def by blast
  hence "tfrset ?M" using assms unfolding tfrsst_def by metis
  thus "tfrsst A" using assms unfolding tfrsst_def by simp
qed

lemma tfrsst_append': "tfrsst (A@B) ⟹ tfrsst B"
proof -
  assume assms: "tfrsst (A@B)"
  let ?M = "trmssst B ∪ pair ' setopssst B"
  let ?N = "trmssst (A@B) ∪ pair ' setopssst (A@B)"
  let ?P = "λt t'. ∀x ∈ fv t ∪ fv t'. ∃a. Γ (Var x) = Var a"
  let ?Q = "λX t t'. X = [] ∨ (∀x ∈ (fv t ∪ fv t')-set X. ∃a. Γ (Var x) = Var a)"
  have *: "SMP ?M - Var'V ⊆ SMP ?N - Var'V" "?M ⊆ ?N"
    using SMP_mono[of ?M ?N] setopssst_append[of A B]
    by auto
  { fix s t assume **: "tfrset ?N" "s ∈ SMP ?M - Var'V" "t ∈ SMP ?M - Var'V" "(∃δ. Unifier δ s t)"
    hence "s ∈ SMP ?N - Var'V" "t ∈ SMP ?N - Var'V" using * by auto
    hence "Γ s = Γ t" using **(1,4) unfolding tfrset_def by blast
  } moreover have "∀t ∈ ?N. wftrm t ⟹ ∀t ∈ ?M. wftrm t" using * by blast
  ultimately have "tfrset ?N ⟹ tfrset ?M" unfolding tfrset_def by blast
  hence "tfrset ?M" using assms unfolding tfrsst_def by metis
  thus "tfrsst B" using assms unfolding tfrsst_def by simp

```

qed

lemma  $tfr_{sst\_cons}$ : " $tfr_{sst} (a\#A) \implies tfr_{sst} A$ "

using  $tfr_{sst\_append}$ '[of "[a]" A] by simp

lemma  $tfr_{sstp\_subst}$ :

assumes  $s$ : " $tfr_{sstp} s$ "

and  $\vartheta$ : " $wt_{subst} \vartheta$ " " $wf_{trms} (subst\_range \vartheta)$ " " $set (bvars_{sstp} s) \cap range\_vars \vartheta = \{\}$ "

shows " $tfr_{sstp} (s \cdot_{sstp} \vartheta)$ "

proof (cases  $s$ )

case (Equality  $a t t'$ )

thus ?thesis

proof (cases " $\exists \delta. Unifier \delta (t \cdot \vartheta) (t' \cdot \vartheta)$ ")

case True

hence " $\exists \delta. Unifier \delta t t'$ " by (metis  $subst\_subst\_compose$ [of  $\_ \vartheta$ ])

moreover have " $\Gamma t = \Gamma (t \cdot \vartheta)$ " " $\Gamma t' = \Gamma (t' \cdot \vartheta)$ " by (metis  $wt\_subst\_trm''$ [OF  $assms(2)$ ])+

ultimately have " $\Gamma (t \cdot \vartheta) = \Gamma (t' \cdot \vartheta)$ " using  $s$  Equality by simp

thus ?thesis using Equality True by simp

qed simp

next

case (NegChecks  $X F G$ )

let  $?P = "\lambda F G. G = [] \wedge (\forall x \in fv_{pairs} F. set X. \exists a. \Gamma (Var x) = TAtom a)"$

let  $?Q = "\lambda F G. \forall f T. Fun f T \in subterms_{set} (trms_{pairs} F \cup pair' set G) \longrightarrow$

$T = [] \vee (\exists s \in set T. s \notin Var' set X)"$

let  $? \vartheta = "rm\_vars (set X) \vartheta"$

have " $?P F G \vee ?Q F G$ " using NegChecks  $assms(1)$  by simp

hence " $?P (F \cdot_{pairs} ? \vartheta) (G \cdot_{pairs} ? \vartheta) \vee ?Q (F \cdot_{pairs} ? \vartheta) (G \cdot_{pairs} ? \vartheta)"$

proof

assume \*: " $?P F G$ "

have " $G \cdot_{pairs} ? \vartheta = []$ " using \* by simp

moreover have " $\exists a. \Gamma (Var x) = TAtom a$ " when  $x$ : " $x \in fv_{pairs} (F \cdot_{pairs} ? \vartheta) - set X$ " for  $x$

proof -

obtain  $t t'$  where  $t$ : " $(t, t') \in set (F \cdot_{pairs} ? \vartheta)$ " " $x \in fv t \cup fv t' - set X$ "

using  $x(1)$  by auto

then obtain  $u u'$  where  $u$ : " $(u, u') \in set F$ " " $u \cdot ? \vartheta = t$ " " $u' \cdot ? \vartheta = t'$ "

unfolding  $subst\_apply\_pairs\_def$  by auto

obtain  $y$  where  $y$ : " $y \in fv u \cup fv u' - set X$ " " $x \in fv (? \vartheta y)$ "

using  $t(2)$   $u(2,3)$   $rm\_vars\_fv\_obtain$  by fast

hence  $a$ : " $\exists a. \Gamma (Var y) = TAtom a$ " using  $u$  \* by auto

have  $a'$ : " $\Gamma (Var y) = \Gamma (? \vartheta y)$ "

using  $wt\_subst\_trm''$ [OF  $wt\_subst\_rm\_vars$ [OF  $\vartheta(1)$ , of " $set X$ "], of " $Var y$ "]

by simp

have " $(\exists z. ? \vartheta y = Var z) \vee (\exists c. ? \vartheta y = Fun c [])$ "

proof (cases " $? \vartheta y \in subst\_range \vartheta$ ")

case True thus ?thesis

using  $a'$   $\vartheta(2)$   $const\_type\_inv\_wf$

by (cases " $? \vartheta y$ ") fastforce+

qed fastforce

hence " $? \vartheta y = Var x$ " using  $y(2)$  by fastforce

hence " $\Gamma (Var x) = \Gamma (Var y)$ " using  $a'$  by simp

thus ?thesis using  $a$  by presburger

qed

ultimately show ?thesis by simp

next

assume \*: " $?Q F G$ "

have \*\*: " $set X \cap range\_vars ? \vartheta = \{\}$ "

using  $\vartheta(3)$  NegChecks  $rm\_vars\_img\_fv\_subset$ [of " $set X$ "  $\vartheta$ ] by auto

have " $?Q (F \cdot_{pairs} ? \vartheta) (G \cdot_{pairs} ? \vartheta)"$

using  $ineq\_subterm\_inj\_cond\_subst$ [OF \*\* \*]

$trms_{pairs\_subst}$ [of  $F$  " $rm\_vars (set X) \vartheta$ "]

```

      subst_apply_pairs_pair_image_subst[of G "rm_vars (set X)  $\vartheta$ "]
    by (metis (no_types, lifting) image_Un)
  thus ?thesis by simp
qed
thus ?thesis using NegChecks by simp
qed simp_all

lemma tfrsstp_all_wt_subst_apply:
  assumes S: "list_all tfrsstp S"
  and  $\vartheta$ : "wtsubst  $\vartheta$ " "wftrms (subst_range  $\vartheta$ )" "bvarssst S  $\cap$  range_vars  $\vartheta$  = {}"
  shows "list_all tfrsstp (S ·sst  $\vartheta$ )"
proof -
  have "set (bvarssstp S)  $\cap$  range_vars  $\vartheta$  = {}" when "s  $\in$  set S" for s
  using that  $\vartheta$ (3) unfolding bvarssst_def range_vars_alt_def by fastforce
  thus ?thesis
  using tfrsstp_subst[OF _  $\vartheta$ (1,2)] S
  unfolding list_all_iff
  by (auto simp add: subst_apply_stateful_strand_def)
qed

lemma trpairs_empty_case:
  assumes "trpairs F D = []"
  shows "D = []" "F  $\neq$  []"
proof -
  show "F  $\neq$  []" using assms by (auto intro: ccontr)

  have "trpairs F (a#A)  $\neq$  []" for a A
  by (induct F "a#A" rule: trpairs.induct) fastforce+
  thus "D = []" using assms by (cases D) simp_all
qed

lemma trpairs_elem_length_eq:
  assumes "G  $\in$  set (trpairs F D)"
  shows "length G = length F"
using assms by (induct F D arbitrary: G rule: trpairs.induct) auto

lemma trpairs_index:
  assumes "G  $\in$  set (trpairs F D)" "i < length F"
  shows " $\exists d \in$  set D. G ! i = (pair (F ! i), pair d)"
using assms
proof (induction F D arbitrary: i G rule: trpairs.induct)
  case (2 s t F D)
  obtain d G' where G:
    "d  $\in$  set D" "G'  $\in$  set (trpairs F D)"
    "G = (pair (s,t), pair d)#G'"
  using "2.prem"(1) by moura
  show ?case
  using "2.IH"[OF G(1,2)] "2.prem"(2) G(1,3)
  by (cases i) auto
qed simp

lemma trpairs_cons:
  assumes "G  $\in$  set (trpairs F D)" "d  $\in$  set D"
  shows "(pair (s,t), pair d)#G  $\in$  set (trpairs ((s,t)#F) D)"
using assms by auto

lemma trpairs_has_pair_lists:
  assumes "G  $\in$  set (trpairs F D)" "g  $\in$  set G"
  shows " $\exists f \in$  set F.  $\exists d \in$  set D. g = (pair f, pair d)"
using assms
proof (induction F D arbitrary: G rule: trpairs.induct)
  case (2 s t F D)
  obtain d G' where G:

```

```

    "d ∈ set D" "G' ∈ set (tr_pairs F D)"
    "G = (pair (s,t), pair d)#G'"
    using "2.premis"(1) by maura
  show ?case
    using "2.IH"[OF G(1,2)] "2.premis"(2) G(1,3)
    by (cases "g ∈ set G'") auto
qed simp

lemma tr_pairs_is_pair_lists:
  assumes "f ∈ set F" "d ∈ set D"
  shows "∃ G ∈ set (tr_pairs F D). (pair f, pair d) ∈ set G"
  (is "?P F D f d")
proof -
  have "∀ f ∈ set F. ∀ d ∈ set D. ?P F D f d"
  proof (induction F D rule: tr_pairs.induct)
    case (2 s t F D)
    hence IH: "∀ f ∈ set F. ∀ d ∈ set D. ?P F D f d" by metis
    moreover have "∀ d ∈ set D. ?P ((s,t)#F) D (s,t) d"
    proof
      fix d assume d: "d ∈ set D"
      then obtain G where G: "G ∈ set (tr_pairs F D)"
      using tr_pairs_empty_case(1) by force
      hence "(pair (s, t), pair d)#G ∈ set (tr_pairs ((s,t)#F) D)"
      using d by auto
      thus "?P ((s,t)#F) D (s,t) d" using d G by auto
    qed
    ultimately show ?case by fastforce
  qed simp
  thus ?thesis by (metis assms)
qed

lemma tr_pairs_db_append_subset:
  "set (tr_pairs F D) ⊆ set (tr_pairs F (D@E))" (is ?A)
  "set (tr_pairs F E) ⊆ set (tr_pairs F (D@E))" (is ?B)
proof -
  show ?A
  proof (induction F D rule: tr_pairs.induct)
    case (2 s t F D)
    show ?case
    proof
      fix G assume "G ∈ set (tr_pairs ((s,t)#F) D)"
      then obtain d G' where G':
        "d ∈ set D" "G' ∈ set (tr_pairs F D)" "G = (pair (s,t), pair d)#G'"
      by maura
      have "d ∈ set (D@E)" "G' ∈ set (tr_pairs F (D@E))" using "2.IH"[OF G'(1)] G'(1,2) by auto
      thus "G ∈ set (tr_pairs ((s,t)#F) (D@E))" using G'(3) by auto
    qed
  qed simp

  show ?B
  proof (induction F E rule: tr_pairs.induct)
    case (2 s t F E)
    show ?case
    proof
      fix G assume "G ∈ set (tr_pairs ((s,t)#F) E)"
      then obtain d G' where G':
        "d ∈ set E" "G' ∈ set (tr_pairs F E)" "G = (pair (s,t), pair d)#G'"
      by maura
      have "d ∈ set (D@E)" "G' ∈ set (tr_pairs F (D@E))" using "2.IH"[OF G'(1)] G'(1,2) by auto
      thus "G ∈ set (tr_pairs ((s,t)#F) (D@E))" using G'(3) by auto
    qed
  qed simp
qed

```

```

lemma tr_pairs_trms_subset:
  "G ∈ set (tr_pairs F D) ⇒ trms_pairs G ⊆ pair ' set F ∪ pair ' set D"
proof (induction F D arbitrary: G rule: tr_pairs.induct)
  case (2 s t F D G)
  obtain d G' where G:
    "d ∈ set D" "G' ∈ set (tr_pairs F D)" "G = (pair (s,t), pair d)#G'"
  using "2.premis"(1) by moura

  show ?case using "2.IH"[OF G(1,2)] G(1,3) by auto
qed simp

lemma tr_pairs_trms_subset':
  "⋃ (trms_pairs ' set (tr_pairs F D)) ⊆ pair ' set F ∪ pair ' set D"
using tr_pairs_trms_subset by blast

lemma tr_trms_subset:
  "A' ∈ set (tr A D) ⇒ trms_st A' ⊆ trms_sst A ∪ pair ' setops_sst A ∪ pair ' set D"
proof (induction A D arbitrary: A' rule: tr.induct)
  case 1 thus ?case by simp
next
  case (2 t A D)
  then obtain A'' where A'': "A' = send⟨t⟩_st#A''" "A'' ∈ set (tr A D)" by moura
  hence "trms_st A'' ⊆ trms_sst A ∪ pair ' setops_sst A ∪ pair ' set D" by (metis "2.IH")
  thus ?case using A'' by (auto simp add: setops_sst_def)
next
  case (3 t A D)
  then obtain A'' where A'': "A' = receive⟨t⟩_st#A''" "A'' ∈ set (tr A D)" by moura
  hence "trms_st A'' ⊆ trms_sst A ∪ pair ' setops_sst A ∪ pair ' set D" by (metis "3.IH")
  thus ?case using A'' by (auto simp add: setops_sst_def)
next
  case (4 ac t t' A D)
  then obtain A'' where A'': "A' = ⟨ac: t ≐ t'⟩_st#A''" "A'' ∈ set (tr A D)" by moura
  hence "trms_st A'' ⊆ trms_sst A ∪ pair ' setops_sst A ∪ pair ' set D" by (metis "4.IH")
  thus ?case using A'' by (auto simp add: setops_sst_def)
next
  case (5 t s A D)
  hence "A' ∈ set (tr A (List.insert (t,s) D))" by simp
  hence "trms_st A' ⊆ trms_sst A ∪ pair ' setops_sst A ∪ pair ' set (List.insert (t, s) D)"
    by (metis "5.IH")
  thus ?case using A' by (auto simp add: setops_sst_def)
next
  case (6 t s A D)
  from 6 obtain Di A'' B C where A'':
    "Di ∈ set (subseqs D)" "A'' ∈ set (tr A [d←D. d ∉ set Di])" "A' = (B@C)@A''"
    "B = map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩_st) Di"
    "C = map (λd. Inequality [] [(pair (t,s), pair d)]) [d←D. d ∉ set Di]"
  by moura
  hence "trms_st A'' ⊆ trms_sst A ∪ pair ' setops_sst A ∪ pair ' set [d←D. d ∉ set Di]"
    by (metis "6.IH")
  hence "trms_st A'' ⊆ trms_sst (Delete t s#A) ∪ pair ' setops_sst (Delete t s#A) ∪ pair ' set D"
    by (auto simp add: setops_sst_def)
  moreover have "trms_st (B@C) ⊆ insert (pair (t,s)) (pair ' set D)"
    using A''(4,5) subseqs_set_subset[OF A''(1)] by auto
  moreover have "pair (t,s) ∈ pair ' setops_sst (Delete t s#A)" by (simp add: setops_sst_def)
  ultimately show ?case using A''(3) trms_st_append[of "B@C" A'] by auto
next
  case (7 ac t s A D)
  from 7 obtain d A'' where A'':
    "d ∈ set D" "A'' ∈ set (tr A D)"
    "A' = ⟨ac: (pair (t,s)) ≐ (pair d)⟩_st#A''"
  by moura
  hence "trms_st A'' ⊆ trms_sst A ∪ pair ' setops_sst A ∪ pair ' set D" by (metis "7.IH")

```

```

moreover have "trmsst A' = {pair (t,s), pair d} ∪ trmsst A'"
using A''(1,3) by auto
ultimately show ?case using A''(1) by (auto simp add: setopssst_def)
next
case (8 X F F' A D)
from 8 obtain A'' where A'':
  "A'' ∈ set (tr A D)" "A' = (map (λG. ∀X⟨V≠: (F@G)⟩st) (trpairs F' D))@A'"
  by moura

define B where "B ≡ ⋃ (trmspairs ' set (trpairs F' D))"

have "trmsst A'' ⊆ trmssst A ∪ pair ' setopssst A ∪ pair ' set D" by (metis A''(1) "8.IH")
hence "trmsst A' ⊆ B ∪ trmspairs F ∪ trmssst A ∪ pair ' setopssst A ∪ pair ' set D"
  using A'' B_def by auto
moreover have "B ⊆ pair ' set F' ∪ pair ' set D"
  using trpairs-trms_subset'[of F' D] B_def by simp
moreover have "pair ' setopssst (∀X⟨V≠: F ∨ ∉: F')#A) = pair ' set F' ∪ pair ' setopssst A"
  by (auto simp add: setopssst_def)
ultimately show ?case by auto
qed

lemma trpairs-vars_subset:
  "G ∈ set (trpairs F D) ⟹ fvpairs G ⊆ fvpairs F ∪ fvpairs D"
proof (induction F D arbitrary: G rule: trpairs.induct)
case (2 s t F D G)
obtain d G' where G:
  "d ∈ set D" "G' ∈ set (trpairs F D)" "G = (pair (s,t), pair d)#G'"
  using "2.prem" (1) by moura

show ?case using "2.IH"[OF G(1,2)] G(1,3) unfolding pair_def by auto
qed simp

lemma trpairs-vars_subset': "⋃ (fvpairs ' set (trpairs F D)) ⊆ fvpairs F ∪ fvpairs D"
using trpairs-vars_subset[of _ F D] by blast

lemma tr_vars_subset:
  assumes "A' ∈ set (tr A D)"
  shows "fvst A' ⊆ fvsst A ∪ (⋃ (t,t') ∈ set D. fv t ∪ fv t')" (is ?P)
  and "bvarsst A' ⊆ bvarssst A" (is ?Q)
proof -
show ?P using assms
proof (induction A arbitrary: A' D rule: strand_sem_stateful_induct)
case (ConsIn A' D ac t s A)
then obtain A'' d where *:
  "d ∈ set D" "A' = ⟨ac: (pair (t,s)) ≐ (pair d)⟩st#A'"
  "A'' ∈ set (tr A D)"
  by moura
hence "fvst A'' ⊆ fvsst A ∪ (⋃ (t,t') ∈ set D. fv t ∪ fv t')" by (metis ConsIn.IH)
thus ?case using * unfolding pair_def by auto
next
case (ConsDel A' D t s A)
define Dfv where "Dfv ≡ λD::('fun,'var) dbstatelist. (⋃ (t,t') ∈ set D. fv t ∪ fv t')"
define fltD where "fltD ≡ λDi. filter (λd. d ∉ set Di) D"
define constr where
  "constr ≡ λDi. (map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di)@
    (map (λd. ∀ []⟨V≠: [(pair (t,s), pair d)]⟩st) (fltD Di))"
from ConsDel obtain A'' Di where *:
  "Di ∈ set (subseqs D)" "A' = (constr Di)@A'" "A'' ∈ set (tr A (fltD Di))"
  unfolding constr_def fltD_def by moura
hence "fvst A'' ⊆ fvsst A ∪ Dfv (fltD Di)"
  unfolding Dfv_def constr_def fltD_def by (metis ConsDel.IH)
moreover have "Dfv (fltD Di) ⊆ Dfv D" unfolding Dfv_def constr_def fltD_def by auto
moreover have "Dfv Di ⊆ Dfv D"

```

```

using subseqs_set_subset(1)[OF *(1)] unfolding Dfv_def constr_def fltD_def by fast
moreover have "fvst (constr Di) ⊆ fv t ∪ fv s ∪ (Dfv Di ∪ Dfv (fltD Di))"
unfolding Dfv_def constr_def fltD_def pair_def by auto
moreover have "fvst (Delete t s#A) = fv t ∪ fv s ∪ fvst A" by auto
moreover have "fvst A' = fvst (constr Di) ∪ fvst A'" using * by force
ultimately have "fvst A' ⊆ fvst (Delete t s#A) ∪ Dfv D" by auto
thus ?case unfolding Dfv_def fltD_def constr_def by simp
next
case (ConsNegChecks A' D X F F' A)
then obtain A'' where A'':
  "A'' ∈ set (tr A D)" "A' = (map (λG. ∀X⟨V≠: (F@G)⟩st) (trpairs F' D))@A'"
  by moura

define B where "B ≡ ⋃ (fvpairs ' set (trpairs F' D))"

have 1: "fvst (map (λG. ∀X⟨V≠: (F@G)⟩st) (trpairs F' D)) ⊆ (B ∪ fvpairs F) - set X"
  unfolding B_def by auto

have 2: "B ⊆ fvpairs F' ∪ fvpairs D"
  using trpairs_vars_subset'[of F' D]
  unfolding B_def by simp

have "fvst A' ⊆ ((fvpairs F' ∪ fvpairs D ∪ fvpairs F) - set X) ∪ fvst A'"
  using 1 2 A''(2) by fastforce
thus ?case using ConsNegChecks.IH[OF A''(1)] by auto
qed fastforce+

show ?Q using assms by (induct A arbitrary: A' D rule: strand_sem_stateful_induct) fastforce+
qed

lemma tr_vars_disj:
  assumes "A' ∈ set (tr A D)" "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  and "fvsst A ∩ bvarssst A = {}"
  shows "fvst A' ∩ bvarsst A' = {}"
  using assms tr_vars_subset by fast

lemma wf_fun_pair_ineqs_map:
  assumes "wfst X A"
  shows "wfst X (map (λd. ∀Y⟨V≠: [(pair (t, s), pair d)]⟩st) D@A)"
  using assms by (induct D) auto

lemma wf_fun_pair_negchecks_map:
  assumes "wfst X A"
  shows "wfst X (map (λG. ∀Y⟨V≠: (F@G)⟩st) M@A)"
  using assms by (induct M) auto

lemma wf_fun_pair_eqs_ineqs_map:
  fixes A:: "('fun, 'var) strand"
  assumes "wfst X A" "Di ∈ set (subseqs D)" "∀(t,t') ∈ set D. fv t ∪ fv t' ⊆ X"
  shows "wfst X ((map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di)@
    (map (λd. ∀[]⟨V≠: [(pair (t,s), pair d)]⟩st) [d←D. d ∉ set Di])@A)"
proof -
  let ?c1 = "map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di"
  let ?c2 = "map (λd. ∀[]⟨V≠: [(pair (t,s), pair d)]⟩st) [d←D. d ∉ set Di]"
  have 1: "wfst X (?c2@A)" using wf_fun_pair_ineqs_map[OF assms(1)] by simp
  have 2: "∀(t,t') ∈ set Di. fv t ∪ fv t' ⊆ X"
    using assms(2,3) by (meson contra_subsetD subseqs_set_subset(1))
  have "wfst X (?c1@B)" when "wfst X B" for B:: "('fun, 'var) strand"
    using 2 that by (induct Di) auto
  thus ?thesis using 1 by simp
qed

lemma trmssst_wt_subst_ex:

```

```

assumes  $\vartheta$ : "wtsubst  $\vartheta$ " "wftrms (subst_range  $\vartheta$ )"
and t: "t  $\in$  trmssst (S ·sst  $\vartheta$ )"
shows " $\exists s \delta. s \in$  trmssst S  $\wedge$  wtsubst  $\delta \wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  t = s ·  $\delta$ "
using t
proof (induction S)
case (Cons s S) thus ?case
proof (cases "t  $\in$  trmssst (S ·sst  $\vartheta$ )")
case False
hence "t  $\in$  trmssstp (s ·sstp  $\vartheta$ )"
using Cons.premss trmssst-subst_cons[of s S  $\vartheta$ ]
by auto
then obtain u where u: "u  $\in$  trmssstp s" "t = u · rm_vars (set (bvarssstp s))  $\vartheta$ "
using trmssstp-subst'' by blast
thus ?thesis
using trmssst-subst_cons[of s S  $\vartheta$ ]
wt_subst_rm_vars[OF  $\vartheta$ (1), of "set (bvarssstp s)"]
wf_trms_subst_rm_vars'[OF  $\vartheta$ (2), of "set (bvarssstp s)"]
by fastforce
qed auto
qed simp

lemma setopssst-wt_subst_ex:
assumes  $\vartheta$ : "wtsubst  $\vartheta$ " "wftrms (subst_range  $\vartheta$ )"
and t: "t  $\in$  pair ' setopssst (S ·sst  $\vartheta$ )"
shows " $\exists s \delta. s \in$  pair ' setopssst S  $\wedge$  wtsubst  $\delta \wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  t = s ·  $\delta$ "
using t
proof (induction S)
case (Cons x S) thus ?case
proof (cases x)
case (Insert t' s)
hence "t = pair (t', s) ·  $\vartheta \vee$  t  $\in$  pair ' setopssst (S ·sst  $\vartheta$ )"
using Cons.premss subst_sst_cons[of _ S  $\vartheta$ ]
unfolding pair_def by (force simp add: setopssst_def)
thus ?thesis
using Insert Cons.IH  $\vartheta$  by (cases "t = pair (t', s) ·  $\vartheta$ ") (fastforce, auto)
next
case (Delete t' s)
hence "t = pair (t', s) ·  $\vartheta \vee$  t  $\in$  pair ' setopssst (S ·sst  $\vartheta$ )"
using Cons.premss subst_sst_cons[of _ S  $\vartheta$ ]
unfolding pair_def by (force simp add: setopssst_def)
thus ?thesis
using Delete Cons.IH  $\vartheta$  by (cases "t = pair (t', s) ·  $\vartheta$ ") (fastforce, auto)
next
case (InSet ac t' s)
hence "t = pair (t', s) ·  $\vartheta \vee$  t  $\in$  pair ' setopssst (S ·sst  $\vartheta$ )"
using Cons.premss subst_sst_cons[of _ S  $\vartheta$ ]
unfolding pair_def by (force simp add: setopssst_def)
thus ?thesis
using InSet Cons.IH  $\vartheta$  by (cases "t = pair (t', s) ·  $\vartheta$ ") (fastforce, auto)
next
case (NegChecks X F F')
hence "t  $\in$  pair ' set (F' ·pairs rm_vars (set X)  $\vartheta$ )  $\vee$  t  $\in$  pair ' setopssst (S ·sst  $\vartheta$ )"
using Cons.premss subst_sst_cons[of _ S  $\vartheta$ ]
unfolding pair_def by (force simp add: setopssst_def)
thus ?thesis
proof
assume "t  $\in$  pair ' set (F' ·pairs rm_vars (set X)  $\vartheta$ )"
then obtain s where s: "t = s · rm_vars (set X)  $\vartheta$ " "s  $\in$  pair ' set F'"
using subst_apply_pairs_pair_image_subst[of F' "rm_vars (set X)  $\vartheta$ "] by auto
thus ?thesis
using NegChecks setopssst-pair_image_cons(8)[of X F F' S]
wt_subst_rm_vars[OF  $\vartheta$ (1), of "set X"]
wf_trms_subst_rm_vars'[OF  $\vartheta$ (2), of "set X"]

```



```

    by fast
    qed (use Cons.IH in auto)
    qed (auto simp add: setopssst_def subst_sst_cons[of _ S ∅])
    qed (simp add: setopssst_def)

lemma setopssst_wftrms:
  "wftrms (trmssst A)  $\implies$  wftrms (pair ' setopssst A)"
  "wftrms (trmssst A)  $\implies$  wftrms (trmssst A  $\cup$  pair ' setopssst A)"
proof -
  show "wftrms (trmssst A)  $\implies$  wftrms (pair ' setopssst A)"
  proof (induction A)
    case (Cons a A)
    hence 0: "wftrms (trmssst a)" "wftrms (pair ' setopssst A)" by auto
    thus ?case
    proof (cases a)
      case (NegChecks X F F')
      hence "wftrms (trmspairs F'" using 0 by simp
      thus ?thesis using NegChecks wftrms_pairs[of F'] 0 by (auto simp add: setopssst_def)
    qed (auto simp add: setopssst_def dest: fun_pair_wftrms)
    qed (auto simp add: setopssst_def)
    thus "wftrms (trmssst A)  $\implies$  wftrms (trmssst A  $\cup$  pair ' setopssst A)" by fast
  qed

lemma SMP_MP_split:
  assumes "t  $\in$  SMP M"
  and M: " $\forall m \in M. \text{is\_Fun } m$ "
  shows "( $\exists \delta. \text{wt}_{\text{subst}} \delta \wedge \text{wf}_{\text{trms}} (\text{subst\_range } \delta) \wedge t \in M \cdot_{\text{set}} \delta$ )  $\vee$ 
    t  $\in$  SMP ((subtermsset M  $\cup \bigcup ((\text{set} \circ \text{fst} \circ \text{Ana}) ' M)) - M$ )"
  (is "?P t  $\vee$  ?Q t")
using assms(1)
proof (induction t rule: SMP.induct)
  case (MP t)
  have "wtsubst Var" "wftrms (subst_range Var)" "M  $\cdot_{\text{set}}$  Var = M" by simp_all
  thus ?case using MP by metis
next
  case (Subterm t t')
  show ?case using Subterm.IH
  proof
    assume "?P t"
    then obtain s  $\delta$  where s: "s  $\in$  M" "t = s  $\cdot$   $\delta$ " and  $\delta$ : "wtsubst  $\delta$ " "wftrms (subst_range  $\delta$ )" by
    moura
    then obtain f T where fT: "s = Fun f T" using M by fast

    have "( $\exists s'. s' \sqsubseteq s \wedge t' = s' \cdot \delta$ )  $\vee$  ( $\exists x \in \text{fv } s. t' \sqsubseteq \delta x$ )"
    using subterm_subst_unfold[OF Subterm.hyps(2)[unfolded s(2)]] by blast
    thus ?thesis
    proof
      assume " $\exists s'. s' \sqsubseteq s \wedge t' = s' \cdot \delta$ "
      then obtain s' where s': "s'  $\sqsubseteq$  s" "t' = s'  $\cdot$   $\delta$ " by moura
      show ?thesis
      proof (cases "s'  $\in$  M")
        case True thus ?thesis using s'  $\delta$  by blast
      next
        case False
        hence "s'  $\in$  (subtermsset M  $\cup \bigcup ((\text{set} \circ \text{fst} \circ \text{Ana}) ' M)) - M$ " using s'(1) s(1) by force
        thus ?thesis using SMP.Substitution[OF SMP.MP[of s']  $\delta$ ] s' by presburger
      qed
    next
      assume " $\exists x \in \text{fv } s. t' \sqsubseteq \delta x$ "
      then obtain x where x: "x  $\in$  fv s" "t'  $\sqsubseteq$   $\delta x$ " by moura
      have "Var x  $\notin$  M" using M by blast
      hence "Var x  $\in$  (subtermsset M  $\cup \bigcup ((\text{set} \circ \text{fst} \circ \text{Ana}) ' M)) - M$ "
      using s(1) var_is_subterm[OF x(1)] by blast
    qed
  qed

```

```

    hence "δ x ∈ SMP ((subtermsset M ∪ ∪ ((set ∘ fst ∘ Ana) ' M)) - M)"
      using SMP.Substitution[OF SMP.MP[of "Var x"] δ] by auto
    thus ?thesis using SMP.Subterm x(2) by presburger
  qed
qed (metis SMP.Subterm[OF _ Subterm.hyps(2)])
next
case (Substitution t δ)
show ?case using Substitution.IH
proof
  assume "?P t"
  then obtain ϑ where "wtsubst ϑ" "wftrms (subst_range ϑ)" "t ∈ M ·set ϑ" by moura
  hence "wtsubst (ϑ ∘s δ)" "wftrms (subst_range (ϑ ∘s δ))" "t · δ ∈ M ·set (ϑ ∘s δ)"
    using wt_subst_compose[of ϑ, OF _ Substitution.hyps(2)]
      wf_trm_subst_compose[of ϑ _ δ, OF _ wf_trm_subst_ranged[OF Substitution.hyps(3)]]
      wf_trm_subst_range_iff
    by (arg0, blast, auto)
  thus ?thesis by blast
next
  assume "?Q t" thus ?thesis using SMP.Substitution[OF _ Substitution.hyps(2,3)] by meson
qed
next
case (Ana t K T k)
show ?case using Ana.IH
proof
  assume "?P t"
  then obtain ϑ where ϑ: "wtsubst ϑ" "wftrms (subst_range ϑ)" "t ∈ M ·set ϑ" by moura
  then obtain s where s: "s ∈ M" "t = s · ϑ" by auto
  then obtain f S where fT: "s = Fun f S" using M by (cases s) auto
  obtain K' T' where s_Ana: "Ana s = (K', T')" by (metis surj_pair)
  hence "set K = set K' ·set ϑ" "set T = set T' ·set ϑ"
    using Ana_subst'[of f S K' T'] fT Ana.hyps(2) s(2) by auto
  then obtain k' where k': "k' ∈ set K'" "k = k' · ϑ" using Ana.hyps(3) by fast
  show ?thesis
  proof (cases "k' ∈ M")
    case True thus ?thesis using k' ϑ(1,2) by blast
  next
    case False
    hence "k' ∈ (subtermsset M ∪ ∪ ((set ∘ fst ∘ Ana) ' M)) - M" using k'(1) s_Ana s(1) by force
    thus ?thesis using SMP.Substitution[OF SMP.MP[of k'] ϑ(1,2)] k'(2) by presburger
  qed
next
  assume "?Q t" thus ?thesis using SMP.Ana[OF _ Ana.hyps(2,3)] by meson
qed
qed

```

lemma setops\_subterm\_trms:

```

  assumes t: "t ∈ pair ' setopssst S"
  and s: "s ⊆ t"
  shows "s ∈ subtermsset (trmssst S)"
proof -
  obtain u u' where u: "pair (u,u') ∈ pair ' setopssst S" "t = pair (u,u')"
    using t setopssst_are_pairs[of _ S] by blast
  hence "s ⊆ u ∨ s ⊆ u'" using s unfolding pair_def by auto
  thus ?thesis using u setopssst_member_iff[of u u' S] unfolding trmssst_def by force
qed

```

lemma setops\_subterms\_cases:

```

  assumes t: "t ∈ subtermsset (pair ' setopssst S)"
  shows "t ∈ subtermsset (trmssst S) ∨ t ∈ pair ' setopssst S"
proof -
  obtain s s' where s: "pair (s,s') ∈ pair ' setopssst S" "t ⊆ pair (s,s')"
    using t setopssst_are_pairs[of _ S] by blast
  hence "t ∈ pair ' setopssst S ∨ t ⊆ s ∨ t ⊆ s'" unfolding pair_def by auto

```

thus ?thesis using s setops<sub>sst</sub>\_member\_iff[of s s' S] unfolding trms<sub>sst</sub>\_def by force  
qed

lemma setops\_SMP\_cases:

assumes "t ∈ SMP (pair ' setops<sub>sst</sub> S)"

and "∀p. Ana (pair p) = ([], [])"

shows "(∃δ. wt<sub>subst</sub> δ ∧ wf<sub>trms</sub> (subst\_range δ) ∧ t ∈ pair ' setops<sub>sst</sub> S ·<sub>set</sub> δ) ∨ t ∈ SMP (trms<sub>sst</sub> S)"

proof -

have 0: "⋃((set ∘ fst ∘ Ana) ' pair ' setops<sub>sst</sub> S) = {}"

proof (induction S)

case (Cons x S) thus ?case

using assms(2) by (cases x) (auto simp add: setops<sub>sst</sub>\_def)

qed (simp add: setops<sub>sst</sub>\_def)

have 1: "∀m ∈ pair ' setops<sub>sst</sub> S. is\_Fun m"

proof (induction S)

case (Cons x S) thus ?case

unfolding pair\_def by (cases x) (auto simp add: assms(2) setops<sub>sst</sub>\_def)

qed (simp add: setops<sub>sst</sub>\_def)

have 2:

"subterms<sub>set</sub> (pair ' setops<sub>sst</sub> S) ∪

⋃((set ∘ fst ∘ Ana) ' (pair ' setops<sub>sst</sub> S)) - pair ' setops<sub>sst</sub> S

⊆ subterms<sub>set</sub> (trms<sub>sst</sub> S)"

using 0 setops\_subterms\_cases by fast

show ?thesis

using SMP\_MP\_split[OF assms(1) 1] SMP\_mono[OF 2] SMP\_subterms\_eq[of "trms<sub>sst</sub> S"]

by blast

qed

lemma tfr\_setops\_if\_tfr\_trms:

assumes "Pair ∉ ⋃(funs\_term ' SMP (trms<sub>sst</sub> S))"

and "∀p. Ana (pair p) = ([], [])"

and "∀s ∈ pair ' setops<sub>sst</sub> S. ∀t ∈ pair ' setops<sub>sst</sub> S. (∃δ. Unifier δ s t) ⟶ Γ s = Γ t"

and "∀s ∈ pair ' setops<sub>sst</sub> S. ∀t ∈ pair ' setops<sub>sst</sub> S.

(∃σ ∅ ρ. wt<sub>subst</sub> σ ∧ wt<sub>subst</sub> ∅ ∧ wf<sub>trms</sub> (subst\_range σ) ∧ wf<sub>trms</sub> (subst\_range ∅) ∧  
Unifier ρ (s · σ) (t · ∅))

⟶ (∃δ. Unifier δ s t)"

and tfr: "tfr<sub>set</sub> (trms<sub>sst</sub> S)"

shows "tfr<sub>set</sub> (trms<sub>sst</sub> S ∪ pair ' setops<sub>sst</sub> S)"

proof -

have 0: "t ∈ SMP (trms<sub>sst</sub> S) - range Var ∨ t ∈ SMP (pair ' setops<sub>sst</sub> S) - range Var"

when "t ∈ SMP (trms<sub>sst</sub> S ∪ pair ' setops<sub>sst</sub> S) - range Var" for t

using that SMP\_union by blast

have 1: "s ∈ SMP (trms<sub>sst</sub> S) - range Var"

when st: "s ∈ SMP (pair ' setops<sub>sst</sub> S) - range Var"

"t ∈ SMP (trms<sub>sst</sub> S) - range Var"

"∃δ. Unifier δ s t"

for s t

proof -

have "(∃δ. s ∈ pair ' setops<sub>sst</sub> S ·<sub>set</sub> δ) ∨ s ∈ SMP (trms<sub>sst</sub> S) - range Var"

using st setops\_SMP\_cases[of s S] assms(2) by blast

moreover {

fix δ assume δ: "s ∈ pair ' setops<sub>sst</sub> S ·<sub>set</sub> δ"

then obtain s' where s': "s' ∈ pair ' setops<sub>sst</sub> S" "s = s' · δ" by blast

then obtain u u' where u: "s' = Fun Pair [u, u']"

using setops<sub>sst</sub>\_are\_pairs[of s'] unfolding pair\_def by fast

hence \*: "s = Fun Pair [u · δ, u' · δ]" using δ s' by simp

obtain f T where fT: "t = Fun f T" using st(2) by (cases t) auto

```

    hence "f ≠ Pair" using st(2) assms(1) by auto
    hence False using st(3) * fT s' u by fast
  } ultimately show ?thesis by meson
qed

have 2: "Γ s = Γ t"
  when "s ∈ SMP (trmssst S) - range Var"
    "t ∈ SMP (trmssst S) - range Var"
    "∃δ. Unifier δ s t"
  for s t
  using that tfr unfolding tfrset_def by blast

have 3: "Γ s = Γ t"
  when st: "s ∈ SMP (pair ' setopssst S) - range Var"
    "t ∈ SMP (pair ' setopssst S) - range Var"
    "∃δ. Unifier δ s t"
  for s t
proof -
  let ?P = "λs δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ s ∈ pair ' setopssst S ·set δ"
  have "(∃δ. ?P s δ) ∨ s ∈ SMP (trmssst S) - range Var"
    "(∃δ. ?P t δ) ∨ t ∈ SMP (trmssst S) - range Var"
    using setops_SMP_cases[of _ S] assms(2) st(1,2) by auto
  hence "(∃δ δ'. ?P s δ ∧ ?P t δ') ∨ Γ s = Γ t" by (metis 1 2 st)
  moreover {
    fix δ δ' assume *: "?P s δ" "?P t δ'"
    then obtain s' t' where **:
      "s' ∈ pair ' setopssst S" "t' ∈ pair ' setopssst S" "s = s' · δ" "t = t' · δ'"
    by blast
    hence "∃θ. Unifier θ s' t'" using st(3) assms(4) * by blast
    hence "Γ s' = Γ t'" using assms(3) ** by blast
    hence "Γ s = Γ t" using * **(3,4) wt_subst_trm''[of δ s'] wt_subst_trm''[of δ' t'] by argo
  } ultimately show ?thesis by blast
qed

show ?thesis using 0 1 2 3 unfolding tfrset_def by metis
qed

```

#### 4.2.2 The Typing Result for Stateful Constraints

```

context
begin
private lemma tr_wf':
  assumes "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  and "∀(t,t') ∈ set D. fv t ∪ fv t' ⊆ X"
  and "wf'sst X A" "fvsst A ∩ bvarssst A = {}"
  and "A' ∈ set (tr A D)"
  shows "wfst X A'"
proof -
  define P where
    "P = (λ(D::('fun,'var) dbstatelist) (A::('fun,'var) stateful_strand).
      (∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}) ∧ fvsst A ∩ bvarssst A = {})"

  have "P D A" using assms(1,4) by (simp add: P_def)
  with assms(5,3,2) show ?thesis
proof (induction A arbitrary: A' D X rule: wf'sst.induct)
  case 1 thus ?case by simp
next
  case (2 X t A A')
  then obtain A'' where A'': "A' = receive⟨t⟩st#A'" "A'' ∈ set (tr A D)" "fv t ⊆ X"
  by maura
  have *: "wf'sst X A" "∀(s,s') ∈ set D. fv s ∪ fv s' ⊆ X" "P D A"
  using 2(1,2,3,4) apply (force, force)
  using 2(5) unfolding P_def by force

```

```

show ?case using "2.IH"[OF A''(2) *] A''(1,3) by simp
next
case (3 X t A A')
then obtain A'' where A'': "A' = send⟨t⟩st#A''" "A'' ∈ set (tr A D)"
  by moura
have *: "wf'sst (X ∪ fv t) A" "∀ (s,s') ∈ set D. fv s ∪ fv s' ⊆ X ∪ fv t" "P D A"
  using 3(1,2,3,4) apply (force, force)
  using 3(5) unfolding P_def by force
show ?case using "3.IH"[OF A''(2) *] A''(1) by simp
next
case (4 X t t' A A')
then obtain A'' where A'': "A' = ⟨assign: t ≐ t'⟩st#A''" "A'' ∈ set (tr A D)" "fv t' ⊆ X"
  by moura
have *: "wf'sst (X ∪ fv t) A" "∀ (s,s') ∈ set D. fv s ∪ fv s' ⊆ X ∪ fv t" "P D A"
  using 4(1,2,3,4) apply (force, force)
  using 4(5) unfolding P_def by force
show ?case using "4.IH"[OF A''(2) *] A''(1,3) by simp
next
case (5 X t t' A A')
then obtain A'' where A'': "A' = ⟨check: t ≐ t'⟩st#A''" "A'' ∈ set (tr A D)"
  by moura
have *: "wf'sst X A" "P D A"
  using 5(3) apply force
  using 5(5) unfolding P_def by force
show ?case using "5.IH"[OF A''(2) *(1) 5(4) *(2)] A''(1) by simp
next
case (6 X t s A A')
hence A': "A' ∈ set (tr A (List.insert (t,s) D))" "fv t ⊆ X" "fv s ⊆ X" by auto
have *: "wf'sst X A" "∀ (s,s') ∈ set (List.insert (t,s) D). fv s ∪ fv s' ⊆ X" using 6 by auto
have **: "P (List.insert (t,s) D) A" using 6(5) unfolding P_def by force
show ?case using "6.IH"[OF A'(1) * **] A'(2,3) by simp
next
case (7 X t s A A')
let ?constr = "λDi. (map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di)@
  (map (λd. ∀ []⟨≠: [(pair (t,s), pair d)]⟩st) [d←D. d ∉ set Di])"
from 7 obtain Di A'' where A'':
  "A' = ?constr Di@A'" "A'' ∈ set (tr A [d←D. d ∉ set Di])"
  "Di ∈ set (subseqs D)"
  by moura
have *: "wf'sst X A" "∀ (t',s') ∈ set [d←D. d ∉ set Di]. fv t' ∪ fv s' ⊆ X"
  using 7 by auto
have **: "P [d←D. d ∉ set Di] A" using 7 unfolding P_def by force
have ***: "∀ (t, t') ∈ set D. fv t ∪ fv t' ⊆ X" using 7 by auto
show ?case
  using "7.IH"[OF A''(2) * **] A''(1) wf_fun_pair_eqs_ineqs_map[OF _ A''(3) ***]
  by simp
next
case (8 X t s A A')
then obtain d A'' where A'':
  "A' = ⟨assign: (pair (t,s)) ≐ (pair d)⟩st#A'"
  "A'' ∈ set (tr A D)" "d ∈ set D"
  by moura
have *: "wf'sst (X ∪ fv t ∪ fv s) A" "∀ (t',s') ∈ set D. fv t' ∪ fv s' ⊆ X ∪ fv t ∪ fv s" "P D A"
  using 8(1,2,3,4) apply (force, force)
  using 8(5) unfolding P_def by force
have **: "fv (pair d) ⊆ X" using A''(3) "8.prem" (3) unfolding pair_def by fastforce
have ***: "fv (pair (t,s)) = fv s ∪ fv t" unfolding pair_def by auto
show ?case using "8.IH"[OF A''(2) *] A''(1) ** *** unfolding pair_def by (simp add: Un_assoc)
next
case (9 X t s A A')
then obtain d A'' where A'':
  "A' = ⟨check: (pair (t,s)) ≐ (pair d)⟩st#A'"
  "A'' ∈ set (tr A D)" "d ∈ set D"

```

```

by maura
have *: "wf'_{sst} X A" "P D A"
  using 9(3) apply force
  using 9(5) unfolding P_def by force
have **: "fv (pair d) ⊆ X" using A''(3) "9.prem" (3) unfolding pair_def by fastforce
have ***: "fv (pair (t,s)) = fv s ∪ fv t" unfolding pair_def by auto
show ?case using "9.IH"[OF A''(2) *(1) 9(4) *(2)] A''(1) ** *** by (simp add: Un_assoc)
next
case (10 X Y F' A A')
from 10 obtain A'' where A'':
  "A' = (map (λG. ∀Y(∀≠: (F@G)_{st}) (tr_{pairs} F' D))@A'') "A'' ∈ set (tr A D)"
by maura

have *: "wf'_{sst} X A" "∀(t',s') ∈ set D. fv t' ∪ fv s' ⊆ X" using 10 by auto

have "bvars_{sst} A ⊆ bvars_{sst} (∀Y(∀≠: F ∨≠: F')#A)" "fv_{sst} A ⊆ fv_{sst} (∀Y(∀≠: F ∨≠: F')#A)" by
auto
hence **: "P D A" using 10 unfolding P_def by blast

show ?case using "10.IH"[OF A''(2) * **] A''(1) wf_fun_pair_negchecks_map by simp
qed
qed

private lemma tr_wf_{trms}:
  assumes "A' ∈ set (tr A [])" "wf_{trms} (trms_{sst} A)"
  shows "wf_{trms} (trms_{st} A)"
using tr_trms_subset[OF assms(1)] setops_{sst}_wf_{trms}(2)[OF assms(2)]
by auto

lemma tr_wf:
  assumes "A' ∈ set (tr A [])"
  and "wf_{sst} A"
  and "wf_{trms} (trms_{sst} A)"
  shows "wf_{st} { } A'"
  and "wf_{trms} (trms_{st} A)"
  and "fv_{st} A' ∩ bvars_{st} A' = { }"
using tr_wf'[OF _ _ _ _ assms(1)]
  tr_wf_{trms}[OF assms(1,3)]
  tr_vars_disj[OF assms(1)]
  assms(2)
by fastforce+

private lemma tr_tfr_{sttp}:
  assumes "A' ∈ set (tr A D)" "list_all tfr_{sttp} A"
  and "fv_{sst} A ∩ bvars_{sst} A = { }" (is "?P0 A D")
  and "∀(t,s) ∈ set D. (fv t ∪ fv s) ∩ bvars_{sst} A = { }" (is "?P1 A D")
  and "∀t ∈ pair ' setops_{sst} A ∪ pair ' set D. ∀t' ∈ pair ' setops_{sst} A ∪ pair ' set D.
    (∃δ. Unifier δ t t') ⟶ Γ t = Γ t'" (is "?P3 A D")
  shows "list_all tfr_{sttp} A'"
proof -
  have sublm: "list_all tfr_{sttp} A" "?P0 A D" "?P1 A D" "?P3 A D"
  when p: "list_all tfr_{sttp} (a#A)" "?P0 (a#A) D" "?P1 (a#A) D" "?P3 (a#A) D"
  for a A D
  using p(1) apply (simp add: tfr_{sst}_def)
  using p(2) fv_{sst}_cons_subset bvars_{sst}_cons_subset apply fast
  using p(3) bvars_{sst}_cons_subset apply fast
  using p(4) setops_{sst}_cons_subset by fast

  show ?thesis using assms
proof (induction A D arbitrary: A' rule: tr.induct)
  case 1 thus ?case by simp
next
  case (2 t A D)

```

```

note prems = "2.prems"
note IH = "2.IH"
from prems(1) obtain A'' where A'': "A' = send⟨t⟩st#A'" "A'' ∈ set (tr A D)"
  by maura
have "list_all tfrstp A'" using IH[OF A''(2)] prems(5) sublimm[OF prems(2,3,4,5)] by meson
thus ?case using A''(1) by simp
next
case (3 t A D)
note prems = "3.prems"
note IH = "3.IH"
from prems(1) obtain A'' where A'': "A' = receive⟨t⟩st#A'" "A'' ∈ set (tr A D)"
  by maura
have "list_all tfrstp A'" using IH[OF A''(2)] prems(5) sublimm[OF prems(2,3,4,5)] by meson
thus ?case using A''(1) by simp
next
case (4 ac t t' A D)
note prems = "4.prems"
note IH = "4.IH"
from prems(1) obtain A'' where A'':
  "A' = ⟨ac: t ≐ t'⟩st#A'" "A'' ∈ set (tr A D)"
  by maura
have "list_all tfrstp A'" using IH[OF A''(2)] prems(5) sublimm[OF prems(2,3,4,5)] by meson
moreover have "(∃δ. Unifier δ t t') ⟹ Γ t = Γ t'" using prems(2) by (simp add: tfrsst_def)
ultimately show ?case using A''(1) by auto
next
case (5 t s A D)
note prems = "5.prems"
note IH = "5.IH"
from prems(1) have A': "A' ∈ set (tr A (List.insert (t,s) D))" by simp

have 1: "list_all tfrstp A" using sublimm[OF prems(2,3,4,5)] by simp

have "pair ' setopssst (Insert t s#A) ∪ pair ' set D =
  pair ' setopssst A ∪ pair ' set (List.insert (t,s) D)"
  by (simp add: setopssst_def)
hence 3: "?P3 A (List.insert (t,s) D)" using prems(5) by metis
moreover have "?P1 A (List.insert (t, s) D)" using prems(3,4) bvarssst_cons_subset[of A] by auto
ultimately have "list_all tfrstp A'" using IH[OF A' sublimm(1,2)[OF prems(2,3,4,5)] _ 3] by metis
thus ?case using A'(1) by auto
next
case (6 t s A D)
note prems = "6.prems"
note IH = "6.IH"

define constr where constr:
  "constr ≡ (λDi. (map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di)@
    (map (λd. ∀ [] (∀≠: [(pair (t,s), pair d)]st) [d←D. d ∉ set Di])))"

from prems(1) obtain Di A'' where A'':
  "A' = constr Di@A'" "A'' ∈ set (tr A [d←D. d ∉ set Di])"
  "Di ∈ set (subseqs D)"
  unfolding constr by auto

define Q1 where "Q1 ≡ (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀x ∈ (fvpairs F) - set X. ∃a. Γ (Var x) = TAtom a)"

define Q2 where "Q2 ≡ (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀f T. Fun f T ∈ subtermsset (trmspairs F) ⟶ T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X))"

have "set [d←D. d ∉ set Di] ⊆ set D"
  "pair ' setopssst A ∪ pair ' set [d←D. d ∉ set Di]
  ⊆ pair ' setopssst (Delete t s#A) ∪ pair ' set D"
  by (auto simp add: setopssst_def)

```

```

hence *: "?P3 A [d←D. d ∉ set Di]" using prems(5) by blast
have **: "?P1 A [d←D. d ∉ set Di]" using prems(4,5) by auto
have 1: "list_all tfrstp A'"
  using IH[OF A''(3,2) sublm(1,2)[OF prems(2,3,4,5)] ** *]
  by metis

have 2: "⟨ac: u ≐ u'⟩st ∈ set A'' ∨
  (∃ d ∈ set Di. u = pair (t,s) ∧ u' = pair d)"
  when "⟨ac: u ≐ u'⟩st ∈ set A'" for ac u u'
  using that A''(1) unfolding constr by force
have 3: "Inequality X U ∈ set A' ⇒ Inequality X U ∈ set A'' ∨
  (∃ d ∈ set [d←D. d ∉ set Di].
    U = [(pair (t,s), pair d)] ∧ Q2 [(pair (t,s), pair d)] X)"
  for X U
  using A''(1) unfolding Q2_def constr by force
have 4:
  "∀ d ∈ set D. (∃ δ. Unifier δ (pair (t,s)) (pair d)) ⇒ Γ (pair (t,s)) = Γ (pair d)"
  using prems(5) by (simp add: setopssst_def)

{ fix ac u u'
  assume a: "⟨ac: u ≐ u'⟩st ∈ set A'" "∃ δ. Unifier δ u u'"
  hence "⟨ac: u ≐ u'⟩st ∈ set A'' ∨ (∃ d ∈ set Di. u = pair (t,s) ∧ u' = pair d)"
    using 2 by metis
  hence "Γ u = Γ u'"
    using 1(1) 4 subseqs_set_subset[OF A''(3)] a(2) tfrstp_list_all_alt_def[of A'']
    by blast
} moreover {
  fix u U
  assume "∀ U (∀ ≠: u) ∈ set A'"
  hence "∀ U (∀ ≠: u) ∈ set A'' ∨
    (∃ d ∈ set [d←D. d ∉ set Di]. u = [(pair (t,s), pair d)] ∧ Q2 u U)"
    using 3 by metis
  hence "Q1 u U ∨ Q2 u U"
    using 1 4 subseqs_set_subset[OF A''(3)] tfrstp_list_all_alt_def[of A'']
    unfolding Q1_def Q2_def
    by blast
} ultimately show ?case using tfrstp_list_all_alt_def[of A'] unfolding Q1_def Q2_def by blast
next
case (7 ac t s A D)
note prems = "7.prems"
note IH = "7.IH"

from prems(1) obtain d A'' where A'':
  "A' = ⟨ac: (pair (t,s)) ≐ (pair d)⟩st#A'"
  "A'' ∈ set (tr A D)" "d ∈ set D"
  by moura

have "list_all tfrstp A'"
  using IH[OF A''(2) sublm(1,2,3)[OF prems(2,3,4,5)] sublm(4)[OF prems(2,3,4,5)]]
  by metis
moreover have "(∃ δ. Unifier δ (pair (t,s)) (pair d)) ⇒ Γ (pair (t,s)) = Γ (pair d)"
  using prems(2,5) A''(3) unfolding tfrsst_def by (simp add: setopssst_def)
ultimately show ?case using A''(1) by fastforce
next
case (8 X F F' A D)
note prems = "8.prems"
note IH = "8.IH"

define constr where "constr = (map (λG. ∀ X (∀ ≠: (F OG) ∈ set (trpairs F' D)))"

define Q1 where "Q1 ≡ (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀ x ∈ (fvpairs F) - set X. ∃ a. Γ (Var x) = TAtom a)"

```



```

define Q2 where "Q2  $\equiv (\lambda(M::('fun, 'var) terms) X.
  \forall f T. \text{Fun } f T \in \text{subterms}_{set} M \longrightarrow T = [] \vee (\exists s \in \text{set } T. s \notin \text{Var } ' \text{set } X))"$ 

have Q2_subset: "Q2 M' X" when "M'  $\subseteq M$ " "Q2 M X" for X M M'
  using that unfolding Q2_def by auto

have Q2_supset: "Q2 (M  $\cup M'$ ) X" when "Q2 M X" "Q2 M' X" for X M M'
  using that unfolding Q2_def by auto

from prems(1) obtain A'' where A'': "A' = constr@A''" "A''  $\in \text{set } (\text{tr } A D)"$ 
  using constr_def by moura

have 0: "F' = []  $\implies \text{constr} = [\forall X(\forall \neq: F)_{st}]$ " unfolding constr_def by simp

have 1: "list_all tfr_stp A''"
  using IH[OF A''(2) sublm(1,2,3)[OF prems(2,3,4,5)] sublm(4)[OF prems(2,3,4,5)]]
  by metis

have 2: "(F' = []  $\wedge Q1 F X$ )  $\vee Q2 (\text{trms}_{pairs} F \cup \text{pair } ' \text{set } F') X$ "
  using prems(2) unfolding Q1_def Q2_def by simp

have 3: "list_all tfr_stp constr" when "F' = []" "Q1 F X"
  using that 0 2 tfr_stp_list_all_alt_def[of constr] unfolding Q1_def by auto

{ fix c assume "c  $\in \text{set } \text{constr}$ "
  hence " $\exists G \in \text{set } (\text{tr}_{pairs} F' D). c = \forall X(\forall \neq: (F@G))_{st}$ " unfolding constr_def by force
} moreover {
  fix G
  assume G: "G  $\in \text{set } (\text{tr}_{pairs} F' D)"$ 
  and c: " $\forall X(\forall \neq: (F@G))_{st} \in \text{set } \text{constr}$ "
  and e: "Q2 (trms_pairs F  $\cup \text{pair } ' \text{set } F')$  X"

  have d_Q2: "Q2 (pair ' set D) X" unfolding Q2_def
  proof (intro allI impI)
    fix f T assume "Fun f T  $\in \text{subterms}_{set} (\text{pair } ' \text{set } D)"$ 
    then obtain d where d: "d  $\in \text{set } D$ " "Fun f T  $\in \text{subterms } (\text{pair } d)"$  by auto
    hence "fv (pair d)  $\cap \text{set } X = \{\}$ " using prems(4) unfolding pair_def by force
    thus "T = []  $\vee (\exists s \in \text{set } T. s \notin \text{Var } ' \text{set } X)"$ "
      by (metis fv_disj_Fun_subterm_param_cases d(2))
  qed

  have "trms_pairs (F@G)  $\subseteq \text{trms}_{pairs} F \cup \text{pair } ' \text{set } F' \cup \text{pair } ' \text{set } D"$ 
  using tr_pairs_trms_subset[OF G] by auto
  hence "Q2 (trms_pairs (F@G)) X" using Q2_subset[OF _ Q2_supset[OF e d_Q2]] by metis
  hence "tfr_stp ( $\forall X(\forall \neq: (F@G))_{st}$ )" by (metis Q2_def tfr_stp.simps(2))
} ultimately have 4: "list_all tfr_stp constr" when "Q2 (trms_pairs F  $\cup \text{pair } ' \text{set } F')$  X"
  using that Ball_set by blast

have 5: "list_all tfr_stp constr" using 2 3 4 by metis

show ?case using 1 5 A''(1) by simp
qed
qed

lemma tr_tfr:
  assumes "A'  $\in \text{set } (\text{tr } A [])"$  and "tfr_sst A" and "fv_sst A  $\cap \text{bvars}_{sst} A = \{\}$ "
  shows "tfr_st A'"
proof -
  have *: "trms_st A'  $\subseteq \text{trms}_{sst} A \cup \text{pair } ' \text{setops}_{sst} A"$  using tr_trms_subset[OF assms(1)] by simp
  hence "SMP (trms_st A')  $\subseteq \text{SMP } (\text{trms}_{sst} A \cup \text{pair } ' \text{setops}_{sst} A)"$  using SMP_mono by simp
  moreover have "tfr_set (trms_sst A  $\cup \text{pair } ' \text{setops}_{sst} A)"$  using assms(2) unfolding tfr_sst_def by fast
  ultimately have 1: "tfr_set (trms_st A')" by (metis tfr_subset(2)[OF _ *])

```

```

have **: "list_all tfrsstp A" using assms(2) unfolding tfrsst_def by fast
have "pair ' setopssst A  $\subseteq$  SMP (trmssst A  $\cup$  pair ' setopssst A) - Var'V"
  using setopssst_are_pairs unfolding pair_def by auto
hence ***: " $\forall t \in \text{pair}'\text{setops}_{sst} A. \forall t' \in \text{pair}'\text{setops}_{sst} A. (\exists \delta. \text{Unifier } \delta \ t \ t') \longrightarrow \Gamma \ t = \Gamma \ t'$ "
  using assms(2) unfolding tfrsst_def tfrset_def by blast
have 2: "list_all tfrstp A'"
  using tr_tfrsstp [OF assms(1) ** assms(3)] *** unfolding pair_def by fastforce

show ?thesis by (metis 1 2 tfrst_def)
qed

private lemma fun_pair_ineqs:
  assumes "d  $\cdot_p \delta \cdot_p \vartheta \neq d' \cdot_p \mathcal{I}$ "
  shows "pair d  $\cdot \delta \cdot \vartheta \neq \text{pair } d' \cdot \mathcal{I}$ "
proof -
  have "d  $\cdot_p (\delta \circ_s \vartheta) \neq d' \cdot_p \mathcal{I}$ " using assms subst_pair_compose by metis
  hence "pair d  $\cdot (\delta \circ_s \vartheta) \neq \text{pair } d' \cdot \mathcal{I}$ " using fun_pair_eq_subst by metis
  thus ?thesis by simp
qed

private lemma tr_Delete_constr_iff_aux1:
  assumes " $\forall d \in \text{set } Di. (t,s) \cdot_p \mathcal{I} = d \cdot_p \mathcal{I}$ "
  and " $\forall d \in \text{set } D - \text{set } Di. (t,s) \cdot_p \mathcal{I} \neq d \cdot_p \mathcal{I}$ "
  shows "[M; map ( $\lambda d. \langle \text{check: } (\text{pair } (t,s)) \doteq (\text{pair } d) \rangle_{st} Di \rangle @$ 
    (map ( $\lambda d. \forall [] \langle \nabla \neq: [(\text{pair } (t,s), \text{pair } d)] \rangle_{st} [d \leftarrow D. d \notin \text{set } Di] \rangle_d \mathcal{I}$ ))]"
proof -
  from assms(2) have
    "[M; map ( $\lambda d. \forall [] \langle \nabla \neq: [(\text{pair } (t,s), \text{pair } d)] \rangle_{st} [d \leftarrow D. d \notin \text{set } Di] \rangle_d \mathcal{I}$ ]"
  proof (induction D)
    case (Cons d D)
    hence IH: "[M; map ( $\lambda d. \forall [] \langle \nabla \neq: [(\text{pair } (t,s), \text{pair } d)] \rangle_{st} [d \leftarrow D. d \notin \text{set } Di] \rangle_d \mathcal{I}$ ]" by auto
    thus ?case
      proof (cases "d  $\in \text{set } Di$ ")
        case False
        hence "(t,s)  $\cdot_p \mathcal{I} \neq d \cdot_p \mathcal{I}$ " using Cons by simp
        hence "pair (t,s)  $\cdot \mathcal{I} \neq \text{pair } d \cdot \mathcal{I}$ " using fun_pair_eq_subst by metis
        moreover have " $\bigwedge t (\delta :: ('fun, 'var) \text{subst}). \text{subst\_domain } \delta = \{\} \implies t \cdot \delta = t$ " by auto
        ultimately have " $\forall \delta. \text{subst\_domain } \delta = \{\} \longrightarrow \text{pair } (t,s) \cdot \delta \cdot \mathcal{I} \neq \text{pair } d \cdot \delta \cdot \mathcal{I}$ " by metis
        thus ?thesis using IH by (simp add: ineq_model_def)
      qed simp
    qed simp
  moreover {
    fix B assume "[M; B]d  $\mathcal{I}$ "
    with assms(1) have "[M; (map ( $\lambda d. \langle \text{check: } (\text{pair } (t,s)) \doteq (\text{pair } d) \rangle_{st} Di \rangle @ B \rangle_d \mathcal{I}$ )]"
      unfolding pair_def by (induction Di) auto
  } ultimately show ?thesis by metis
qed

private lemma tr_Delete_constr_iff_aux2:
  assumes "ground M"
  and "[M; (map ( $\lambda d. \langle \text{check: } (\text{pair } (t,s)) \doteq (\text{pair } d) \rangle_{st} Di \rangle @$ 
    (map ( $\lambda d. \forall [] \langle \nabla \neq: [(\text{pair } (t,s), \text{pair } d)] \rangle_{st} [d \leftarrow D. d \notin \text{set } Di] \rangle_d \mathcal{I}$ )))]"
  shows " $(\forall d \in \text{set } Di. (t,s) \cdot_p \mathcal{I} = d \cdot_p \mathcal{I}) \wedge (\forall d \in \text{set } D - \text{set } Di. (t,s) \cdot_p \mathcal{I} \neq d \cdot_p \mathcal{I})$ "
proof -
  let ?c1 = "map ( $\lambda d. \langle \text{check: } (\text{pair } (t,s)) \doteq (\text{pair } d) \rangle_{st} Di$ )"
  let ?c2 = "map ( $\lambda d. \forall [] \langle \nabla \neq: [(\text{pair } (t,s), \text{pair } d)] \rangle_{st} [d \leftarrow D. d \notin \text{set } Di] \rangle_d \mathcal{I}$ )"

  have "M  $\cdot_{set} \mathcal{I} = M$ " using assms(1) subst_all_ground_ident by metis
  moreover have "ikst ?c1 =  $\{\}$ " by auto
  ultimately have *:
    "[M; map ( $\lambda d. \langle \text{check: } (\text{pair } (t,s)) \doteq (\text{pair } d) \rangle_{st} Di \rangle_d \mathcal{I}$ ]"
    "[M; map ( $\lambda d. \forall [] \langle \nabla \neq: [(\text{pair } (t,s), \text{pair } d)] \rangle_{st} [d \leftarrow D. d \notin \text{set } Di] \rangle_d \mathcal{I}$ )]"

```

```

using strand_sem_split(3,4)[of M ?c1 ?c2 I] assms(2) by auto

from *(1) have 1: "∀d ∈ set Di. (t,s) ·p I = d ·p I" unfolding pair_def by (induct Di) auto
from *(2) have 2: "∀d ∈ set D - set Di. (t,s) ·p I ≠ d ·p I"
proof (induction D arbitrary: Di)
  case (Cons d D) thus ?case
  proof (cases "d ∈ set Di")
    case False
    hence IH: "∀d ∈ set D - set Di. (t,s) ·p I ≠ d ·p I" using Cons by force
    have "∧t (δ::('fun,'var) subst). subst_domain δ = {} ∧ ground (subst_range δ) ⟷ δ = Var"
      by auto
    moreover have "ineq_model I [] [(pair (t,s)), (pair d)]"
      using False Cons.prems by simp
    ultimately have "pair (t,s) · I ≠ pair d · I" by (simp add: ineq_model_def)
    thus ?thesis using IH unfolding pair_def by force
  qed simp
qed simp

show ?thesis by (metis 1 2)
qed

private lemma tr_Delete_constr_iff:
  fixes I::('fun,'var) subst
  assumes "ground M"
  shows "set Di ·pset I ⊆ {(t,s) ·p I} ∧ (t,s) ·p I ∉ (set D - set Di) ·pset I ⟷
    [M; (map (λd. (check: (pair (t,s)) ≐ (pair d))st) Di)@
      (map (λd. ∀ [] (V≠: [(pair (t,s), pair d)]st) [d←D. d ∉ set Di])]_d I"
proof -
  let ?constr = "(map (λd. (check: (pair (t,s)) ≐ (pair d))st) Di)@
    (map (λd. ∀ [] (V≠: [(pair (t,s), pair d)]st) [d←D. d ∉ set Di])"
  { assume "set Di ·pset I ⊆ {(t,s) ·p I}" "(t,s) ·p I ∉ (set D - set Di) ·pset I"
    hence "∀d ∈ set Di. (t,s) ·p I = d ·p I" "∀d ∈ set D - set Di. (t,s) ·p I ≠ d ·p I"
      by auto
    hence "[M; ?constr]_d I" using tr_Delete_constr_iff_aux1 by simp
  } moreover {
    assume "[M; ?constr]_d I"
    hence "∀d ∈ set Di. (t,s) ·p I = d ·p I" "∀d ∈ set D - set Di. (t,s) ·p I ≠ d ·p I"
      using assms tr_Delete_constr_iff_aux2 by auto
    hence "set Di ·pset I ⊆ {(t,s) ·p I} ∧ (t,s) ·p I ∉ (set D - set Di) ·pset I" by force
  } ultimately show ?thesis by metis
qed

private lemma tr_NotInSet_constr_iff:
  fixes I::('fun,'var) subst
  assumes "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ set X = {}"
  shows "(∀δ. subst_domain δ = set X ∧ ground (subst_range δ) ⟶ (t,s) ·p δ ·p I ∉ set D ·pset I)
    ⟷ [M; map (λd. ∀ X (V≠: [(pair (t,s), pair d)]st) D)]_d I"
proof -
  { assume "∀δ. subst_domain δ = set X ∧ ground (subst_range δ) ⟶ (t,s) ·p δ ·p I ∉ set D ·pset I"
    with assms have "[M; map (λd. ∀ X (V≠: [(pair (t,s), pair d)]st) D)]_d I"
  }
  proof (induction D)
    case (Cons d D)
    obtain t' s' where d: "d = (t',s')" by mouna
    have "[M; map (λd. ∀ X (V≠: [(pair (t,s), pair d)]st) D)]_d I"
      "map (λd. ∀ X (V≠: [(pair (t,s), pair d)]st) (d#D)) =
        ∀ X (V≠: [(pair (t,s), pair d)]st)#map (λd. ∀ X (V≠: [(pair (t,s), pair d)]st) D"
      using Cons by auto
    moreover have
      "∀δ. subst_domain δ = set X ∧ ground (subst_range δ) ⟶ pair (t, s) · δ · I ≠ pair d · I"
      using fun_pair_ineqs[of I _ "(t,s)" I d] Cons.prems(2) by auto
    moreover have "(fv t' ∪ fv s') ∩ set X = {}" using Cons.prems(1) d by auto
    hence "∀δ. subst_domain δ = set X ⟶ pair d · δ = pair d" using d unfolding pair_def by auto
    ultimately show ?case by (simp add: ineq_model_def)
  qed

```

```

qed simp
} moreover {
  fix  $\delta :: ('fun, 'var) \text{subst}$ 
  assume "[M; map ( $\lambda d. \forall X (\forall \neq: [(pair (t,s), pair d)]_{st}) D]_d \mathcal{I}$ "
    and  $\delta$ : "subst_domain  $\delta = \text{set } X$ " "ground (subst_range  $\delta$ )"
  with assms have "(t,s)  $\cdot_p \delta \cdot_p \mathcal{I} \notin \text{set } D \cdot_{pset} \mathcal{I}$ "
  proof (induction D)
    case (Cons d D)
    obtain t' s' where d: "d = (t', s')" by mouna
    have "(t,s)  $\cdot_p \delta \cdot_p \mathcal{I} \notin \text{set } D \cdot_{pset} \mathcal{I}$ "
      "pair (t,s)  $\cdot \delta \cdot \mathcal{I} \neq \text{pair } d \cdot \delta \cdot \mathcal{I}$ "
    using Cons d by (auto simp add: ineq_model_def simp del: subst_range.simps)
    moreover have "pair d  $\cdot \delta = \text{pair } d$ "
    using Cons.prem1 fun_pair_subst[of d  $\delta$ ] d  $\delta$ (1) unfolding pair_def by auto
    ultimately show ?case unfolding pair_def by force
  qed simp
} ultimately show ?thesis by metis
qed

lemma tr_NegChecks_constr_iff:
  " $(\forall G \in \text{set } L. \text{ineq\_model } \mathcal{I} X (F@G)) \longleftrightarrow [M; \text{map } (\lambda G. \forall X (\forall \neq: (F@G)_{st}) L]_d \mathcal{I}$ " (is ?A)
  "negchecks_model  $\mathcal{I} D X F F' \longleftrightarrow [M; D; [\forall X (\forall \neq: F \vee \notin: F')]_s \mathcal{I}$ " (is ?B)
proof -
  show ?A by (induct L) auto
  show ?B by simp
qed

lemma tr_pairs_sem_equiv:
  fixes  $\mathcal{I} :: ('fun, 'var) \text{subst}$ 
  assumes " $\forall (t, t') \in \text{set } D. (fv t \cup fv t') \cap \text{set } X = \{\}$ "
  shows "negchecks_model  $\mathcal{I} (\text{set } D \cdot_{pset} \mathcal{I}) X F F' \longleftrightarrow$ 
     $(\forall G \in \text{set } (tr\_pairs F' D). \text{ineq\_model } \mathcal{I} X (F@G))$ "
proof -
  define P where
    "P  $\equiv \lambda \delta :: ('fun, 'var) \text{subst}. \text{subst\_domain } \delta = \text{set } X \wedge \text{ground } (\text{subst\_range } \delta)$ "

  define Ineq where
    "Ineq  $\equiv \lambda (\delta :: ('fun, 'var) \text{subst}) F. \text{list\_ex } (\lambda f. \text{fst } f \cdot \delta \circ_s \mathcal{I} \neq \text{snd } f \cdot \delta \circ_s \mathcal{I}) F$ "

  define Ineq' where
    "Ineq'  $\equiv \lambda (\delta :: ('fun, 'var) \text{subst}) F. \text{list\_ex } (\lambda f. \text{fst } f \cdot \delta \circ_s \mathcal{I} \neq \text{snd } f \cdot \mathcal{I}) F$ "

  define Notin where
    "Notin  $\equiv \lambda (\delta :: ('fun, 'var) \text{subst}) D F'. \text{list\_ex } (\lambda f. f \cdot_p \delta \circ_s \mathcal{I} \notin \text{set } D \cdot_{pset} \mathcal{I}) F'$ "

  have sublm:
    " $((s, t) \cdot_p \delta \circ_s \mathcal{I} \notin \text{set } D \cdot_{pset} \mathcal{I}) \longleftrightarrow (\text{list\_all } (\lambda d. \text{Ineq'} \delta [(pair (s, t), pair d)]) D)$ "
    for s t  $\delta$  D
    unfolding pair_def by (induct D) (auto simp add: Ineq'_def)

  have "Notin  $\delta$  D F'  $\longleftrightarrow (\forall G \in \text{set } (tr\_pairs F' D). \text{Ineq'} \delta G)$ "
    (is "?A  $\longleftrightarrow$  ?B")
    when "P  $\delta$ " for  $\delta$ 
  proof
    show "?A  $\implies$  ?B"
    proof (induction F' D rule: tr_pairs.induct)
      case (2 s t F' D)
      show ?case
      proof (cases "Notin  $\delta$  D F'")
        case False
        hence "(s, t)  $\cdot_p \delta \circ_s \mathcal{I} \notin \text{set } D \cdot_{pset} \mathcal{I}$ "
          using "2.prem1"
          by (auto simp add: Notin_def)
      qed
    qed
  end

```

```

hence "pair (s,t) · δ ∘s I ≠ pair d · I" when "d ∈ set D" for d
  using that sublimm Ball_set[of D "λd. Ineq' δ [(pair (s,t), pair d)]"]
  by (simp add: Ineq'_def)
moreover have "∃ d ∈ set D. ∃ G'. G = (pair (s,t), pair d)#G'"
  when "G ∈ set (tr_pairs ((s,t)#F') D)" for G
  using that tr_pairs_index[OF that, of 0] by force
ultimately show ?thesis by (simp add: Ineq'_def)
qed (auto dest: "2.IH" simp add: Ineq'_def)
qed (simp add: Notin_def)

have "¬?A ⇒ ¬?B"
proof (induction F' D rule: tr_pairs.induct)
  case (2 s t F' D)
  then obtain G where G: "G ∈ set (tr_pairs F' D)" "¬Ineq' δ G"
  by (auto simp add: Notin_def)

  obtain d where d: "d ∈ set D" "pair (s,t) · δ ∘s I = pair d · I"
  using "2.prem"
  unfolding pair_def by (auto simp add: Notin_def)
  thus ?case
  using G(2) tr_pairs_cons[OF G(1) d(1)]
  by (auto simp add: Ineq'_def)
qed (simp add: Ineq'_def)
thus "?B ⇒ ?A" by metis
qed

hence *: "(∀ δ. P δ ⇒ Ineq δ F ∨ Notin δ D F') ⇔
  (∀ G ∈ set (tr_pairs F' D). ∀ δ. P δ ⇒ Ineq δ F ∨ Ineq' δ G)"
by auto

have "snd g · δ = snd g"
  when "G ∈ set (tr_pairs F' D)" "g ∈ set G" "P δ"
  for δ g G
  using assms that(3) tr_pairs_has_pair_lists[OF that(1,2)]
  unfolding pair_def by (fastforce simp add: P_def)
hence **: "Ineq' δ G = Ineq δ G"
  when "G ∈ set (tr_pairs F' D)" "P δ"
  for δ G
  using Bex_set[of G "λf. fst f · δ ∘s I ≠ snd f · I"]
  Bex_set[of G "λf. fst f · δ ∘s I ≠ snd f · δ ∘s I"]
  that
  by (simp add: Ineq_def Ineq'_def)

show ?thesis
  using *
  by (simp add: Ineq_def Ineq'_def Notin_def P_def negchecks_model_def ineq_model_def)
qed

lemma tr_sem_equiv':
  assumes "∀ (t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  and "fvsst A ∩ bvarssst A = {}"
  and "ground M"
  and "I: interpretationsubst I"
  shows "[M; set D ·pset I; A]s I ⇔ (∃ A' ∈ set (tr A D). [M; A']d I)" (is "?P ⇔ ?Q")
proof
  have I_grounds: "∧ t. fv (t · I) = {}" by (rule interpretation_grounds[OF I])
  have "∃ A' ∈ set (tr A D). [M; A']d I" when ?P using that assms(1,2,3)
  proof (induction A arbitrary: D rule: strand_sem_stateful_induct)
    case (ConsRcv M D t A)
    have "[insert (t · I) M; set D ·pset I; A]s I"
      "∀ (t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
      "fvsst A ∩ bvarssst A = {}" "ground (insert (t · I) M)"
      using I ConsRcv.prem unfolding fvsst_def bvarssst_def by force+
    then obtain A' where A': "A' ∈ set (tr A D)" "[insert (t · I) M; A']d I" by (metis ConsRcv.IH)
  end
end

```

```

thus ?case by auto
next
case (ConsSnd M D t A)
have "[M; set D ·pset I; A]s I"
  "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  "fvsst A ∩ bvarssst A = {}" "ground M"
  and *: "M ⊢ t · I"
  using I ConsSnd.prem unfolding fvsst_def bvarssst_def by force+
then obtain A' where A': "A' ∈ set (tr A D)" "[M; A']d I" by (metis ConsSnd.IH)
thus ?case using * by auto
next
case (ConsEq M D ac t t' A)
have "[M; set D ·pset I; A]s I"
  "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  "fvsst A ∩ bvarssst A = {}" "ground M"
  and *: "t · I = t' · I"
  using I ConsEq.prem unfolding fvsst_def bvarssst_def by force+
then obtain A' where A': "A' ∈ set (tr A D)" "[M; A']d I" by (metis ConsEq.IH)
thus ?case using * by auto
next
case (ConsIns M D t s A)
have "[M; set (List.insert (t,s) D) ·pset I; A]s I"
  "∀(t,t') ∈ set (List.insert (t,s) D). (fv t ∪ fv t') ∩ bvarssst A = {}"
  "fvsst A ∩ bvarssst A = {}" "ground M"
  using ConsIns.prem unfolding fvsst_def bvarssst_def by force+
then obtain A' where A': "A' ∈ set (tr A (List.insert (t,s) D))" "[M; A']d I"
  by (metis ConsIns.IH)
thus ?case by auto
next
case (ConsDel M D t s A)
have *: "[M; (set D ·pset I) - {(t,s) ·p I}; A]s I"
  "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  "fvsst A ∩ bvarssst A = {}" "ground M"
  using ConsDel.prem unfolding fvsst_def bvarssst_def by force+
then obtain Di where Di:
  "Di ⊆ set D" "Di ·pset I ⊆ {(t,s) ·p I}" "(t,s) ·p I ∉ (set D - Di) ·pset I"
  using subset_subset_pairs_diff_exists'[of "set D"] by maura
hence **: "(set D ·pset I) - {(t,s) ·p I} = (set D - Di) ·pset I" by blast

obtain Di' where Di': "set Di' = Di" "Di' ∈ set (subseqs D)"
  using subset_sublist_exists[OF Di(1)] by maura
hence ***: "(set D ·pset I) - {(t,s) ·p I} = (set [d ← D. d ∉ set Di'] ·pset I)"
  using Di ** by auto

define constr where "constr ≡
  map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di'@
  map (λd. ∀ [] ⟨≠: [(pair (t,s), pair d)]⟩st) [d ← D. d ∉ set Di']"

have ****: "∀(t,t') ∈ set [d ← D. d ∉ set Di']. (fv t ∪ fv t') ∩ bvarssst A = {}"
  using *(2) Di(1) Di'(1) subseqs_set_subset[OF Di'(2)] by simp
have "set D - Di = set [d ← D. d ∉ set Di']" using Di Di' by auto
hence *****: "[M; set [d ← D. d ∉ set Di'] ·pset I; A]s I"
  using *(1) ** by metis
obtain A' where A': "A' ∈ set (tr A [d ← D. d ∉ set Di'])" "[M; A']d I"
  using ConsDel.IH[OF *****] by maura
hence constr_sat: "[M; constr]d I"
  using Di Di' *(1) *** tr_Delete_constr_iff[OF *(4), of I Di' t s D]
  unfolding constr_def by auto

have "constr@A' ∈ set (tr (Delete t s A) D)" using A'(1) Di' unfolding constr_def by auto
moreover have "ikst constr = {}" unfolding constr_def by auto
hence "[M ·set I; constr]d I" "[M ∪ (ikst constr ·set I); A']d I"
  using constr_sat A'(2) subst_all_ground_ident[OF *(4)] by simp_all

```

```

ultimately show ?case
  using strand_sem_append(2)[of _ _ I]
    subst_all_ground_ident[OF *(4), of I]
  by metis
next
case (ConsIn M D ac t s A)
have "[M; set D ·pset I; A]s I"
  "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  "fvsst A ∩ bvarssst A = {}" "ground M"
  and *: "(t,s) ·p I ∈ set D ·pset I"
  using I ConsIn.premis unfolding fvsst_def bvarssst_def by force+
then obtain A' where A': "A' ∈ set (tr A D)" "[M; A']d I" by (metis ConsIn.IH)
moreover obtain d where "d ∈ set D" "pair (t,s) · I = pair d · I"
  using * unfolding pair_def by auto
ultimately show ?case using * by auto
next
case (ConsNegChecks M D X F F' A)
let ?ineqs = "(map (λG. ∀X(∀≠: (F@G)st) (trpairs F' D)))"
have 1: "[M; set D ·pset I; A]s I" "ground M" using ConsNegChecks by auto
have 2: "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}" "fvsst A ∩ bvarssst A = {}"
  using ConsNegChecks.premis(2,3) I unfolding fvsst_def bvarssst_def by fastforce+

have 3: "negchecks_model I (set D ·pset I) X F F'" using ConsNegChecks.premis(1) by simp
from 1 2 obtain A' where A': "A' ∈ set (tr A D)" "[M; A']d I" by (metis ConsNegChecks.IH)

have 4: "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ set X = {}"
  using ConsNegChecks.premis(2) unfolding bvarssst_def by auto

have "[M; ?ineqs]d I"
  using 3 trpairs_sem_equiv[OF 4] tr_NegChecks_constr_iff
  by metis
moreover have "ikst ?ineqs = {}" by auto
moreover have "M ·set I = M" using 1(2) I by (simp add: subst_all_ground_ident)
ultimately show ?case
  using strand_sem_append(2)[of M ?ineqs I A'] A'
  by force
qed simp
thus "?P ⇒ ?Q" by metis

have "(∃A' ∈ set (tr A D). [M; A']d I) ⇒ ?P" using assms(1,2,3)
proof (induction A arbitrary: D rule: strand_sem_stateful_induct)
  case (ConsRcv M D t A)
  have "∃A' ∈ set (tr A D). [insert (t · I) M; A']d I"
    "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
    "fvsst A ∩ bvarssst A = {}" "ground (insert (t · I) M)"
    using I ConsRcv.premis unfolding fvsst_def bvarssst_def by force+
  hence "[insert (t · I) M; set D ·pset I; A]s I" by (metis ConsRcv.IH)
  thus ?case by auto
next
  case (ConsSnd M D t A)
  have "∃A' ∈ set (tr A D). [M; A']d I"
    "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
    "fvsst A ∩ bvarssst A = {}" "ground M"
    and *: "M ⊢ t · I"
    using I ConsSnd.premis unfolding fvsst_def bvarssst_def by force+
  hence "[M; set D ·pset I; A]s I" by (metis ConsSnd.IH)
  thus ?case using * by auto
next
  case (ConsEq M D ac t t' A)
  have "∃A' ∈ set (tr A D). [M; A']d I"
    "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
    "fvsst A ∩ bvarssst A = {}" "ground M"
    and *: "t · I = t' · I"

```

```

    using  $\mathcal{I}$  ConsEq.prems unfolding  $fv_{sst\_def}$   $bvars_{sst\_def}$  by force+
  hence "[M; set D ·pset  $\mathcal{I}$ ; A]s  $\mathcal{I}$ " by (metis ConsEq.IH)
  thus ?case using * by auto
next
case (ConsIns M D t s A)
hence " $\exists A' \in \text{set } (tr A (List.insert (t,s) D)). [M; A']_d \mathcal{I}$ "
  " $\forall (t,t') \in \text{set } (List.insert (t,s) D). (fv t \cup fv t') \cap bvars_{sst} A = \{\}$ "
  " $fv_{sst} A \cap bvars_{sst} A = \{\}$ " "ground M"
  unfolding  $fv_{sst\_def}$   $bvars_{sst\_def}$  by auto+
hence "[M; set (List.insert (t,s) D) ·pset  $\mathcal{I}$ ; A]s  $\mathcal{I}$ " by (metis ConsIns.IH)
thus ?case by auto
next
case (ConsDel M D t s A)
define constr where "constr  $\equiv$ 
   $\lambda Di. \text{map } (\lambda d. \langle \text{check: } (pair (t,s)) \doteq (pair d) \rangle_{st}) Di @$ 
   $\text{map } (\lambda d. \forall [] \langle \nabla \neq: [(pair (t,s), pair d)]_{st} \rangle [d \leftarrow D. d \notin \text{set } Di])"$ 
let ?flt = " $\lambda Di. \text{filter } (\lambda d. d \notin \text{set } Di) D$ "

have " $\exists Di \in \text{set } (subseqs D). \exists B' \in \text{set } (tr A (?flt Di)). B = \text{constr } Di @ B'$ "
  when " $B \in \text{set } (tr (delete \langle t,s \rangle \# A) D)$ " for B
  using that unfolding constr_def by auto
then obtain A' Di where A':
  " $\text{constr } Di @ A' \in \text{set } (tr (Delete t s \# A) D)$ "
  " $A' \in \text{set } (tr A (?flt Di))$ "
  " $Di \in \text{set } (subseqs D)$ "
  " $[M; \text{constr } Di @ A']_d \mathcal{I}$ "
  using ConsDel.prems(1) by blast

have 1: " $\forall (t,t') \in \text{set } (?flt Di). (fv t \cup fv t') \cap bvars_{sst} A = \{\}$ " using ConsDel.prems(2) by
auto
have 2: " $fv_{sst} A \cap bvars_{sst} A = \{\}$ " using ConsDel.prems(3) by force+
have " $ik_{st} (\text{constr } Di) = \{\}$ " unfolding constr_def by auto
hence 3: " $[M; A']_d \mathcal{I}$ "
  using subst_all_ground_ident[OF ConsDel.prems(4)] A'(4)
  strand_sem_split(4)[of M "constr Di" A'  $\mathcal{I}$ ]
  by simp
have IH: " $[M; \text{set } (?flt Di) ·_{pset} \mathcal{I}; A]s \mathcal{I}$ "
  by (metis ConsDel.IH[OF _ 1 2 ConsDel.prems(4)] 3 A'(2))

have "[M; constr Di]d  $\mathcal{I}$ "
  using subst_all_ground_ident[OF ConsDel.prems(4)] strand_sem_split(3) A'(4)
  by metis
hence *: " $\text{set } Di ·_{pset} \mathcal{I} \subseteq \{(t,s) ·_p \mathcal{I}\}$ " " $(t,s) ·_p \mathcal{I} \notin (\text{set } D - \text{set } Di) ·_{pset} \mathcal{I}$ "
  using tr_Delete_constr_iff[OF ConsDel.prems(4), of  $\mathcal{I}$  Di t s D] unfolding constr_def by auto
have 4: " $\text{set } (?flt Di) ·_{pset} \mathcal{I} = (\text{set } D ·_{pset} \mathcal{I}) - \{(t,s) ·_p \mathcal{I}\}$ "
proof
  show " $\text{set } (?flt Di) ·_{pset} \mathcal{I} \subseteq (\text{set } D ·_{pset} \mathcal{I}) - \{(t,s) ·_p \mathcal{I}\}$ "
  proof
    fix u u' assume u: " $(u,u') \in \text{set } (?flt Di) ·_{pset} \mathcal{I}$ "
    then obtain v v' where v: " $(v,v') \in \text{set } D - \text{set } Di$ " " $(v,v') ·_p \mathcal{I} = (u,u')$ " by auto
    hence " $(u,u') \neq (t,s) ·_p \mathcal{I}$ " using * by force
    thus " $(u,u') \in (\text{set } D ·_{pset} \mathcal{I}) - \{(t,s) ·_p \mathcal{I}\}$ "
    using u v * subseqs_set_subset[OF A'(3)] by auto
  qed
  show " $(\text{set } D ·_{pset} \mathcal{I}) - \{(t,s) ·_p \mathcal{I}\} \subseteq \text{set } (?flt Di) ·_{pset} \mathcal{I}$ "
  using * subseqs_set_subset[OF A'(3)] by force
qed

show ?case using 4 IH by simp
next
case (ConsIn M D ac t s A)
have " $\exists A' \in \text{set } (tr A D). [M; A']_d \mathcal{I}$ "
  " $\forall (t,t') \in \text{set } D. (fv t \cup fv t') \cap bvars_{sst} A = \{\}$ "

```



```

    "fvsst A ∩ bvarssst A = {}" "ground M"
    and *: "(t,s) ·p I ∈ set D ·pset I"
    using ConsIn.prem(1,2,3,4) apply (fastforce, fastforce, fastforce, fastforce)
    using ConsIn.prem(1) tr.simps(7)[of ac t s A D] unfolding pair_def by fastforce
    hence "[M; set D ·pset I; A]s I" by (metis ConsIn.IH)
    moreover obtain d where "d ∈ set D" "pair (t,s) · I = pair d · I"
    using * unfolding pair_def by auto
    ultimately show ?case using * by auto
next
case (ConsNegChecks M D X F F' A)
let ?ineqs = "(map (λG. ∀X(∀≠: (F@G)st) (trpairs F' D)))"

obtain B where B:
  "?ineqs@B ∈ set (tr (NegChecks X F F'#A) D)" "[M; ?ineqs@B]d I" "B ∈ set (tr A D)"
  using ConsNegChecks.prem(1) by maura
moreover have "M ·set I = M"
  using ConsNegChecks.prem(4) I by (simp add: subst_all_ground_ident)
moreover have "ikst ?ineqs = {}" by auto
ultimately have "[M; B]d I" using strand_sem_split(4)[of M ?ineqs B I] by simp
moreover have "∀(t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}" "fvsst A ∩ bvarssst A = {}"
  using ConsNegChecks.prem(2,3) unfolding fvsst_def bvarssst_def by force+
ultimately have "[M; set D ·pset I; A]s I"
  by (metis ConsNegChecks.IH B(3) ConsNegChecks.prem(4))
moreover have "∀(t, t') ∈ set D. (fv t ∪ fv t') ∩ set X = {}"
  using ConsNegChecks.prem(2) unfolding bvarssst_def by force
ultimately show ?case
  using trpairs_sem_equiv tr_NegChecks_constr_iff
    B(2) strand_sem_split(3)[of M ?ineqs B I] (M ·set I = M)
  by simp
qed simp
thus "?Q ⇒ ?P" by metis
qed

lemma tr_sem_equiv:
  assumes "fvsst A ∩ bvarssst A = {}" and "interpretationsubst I"
  shows "I ⊨s A ↔ (∃A' ∈ set (tr A []). (I ⊨ ⟨A'⟩))"
  using tr_sem_equiv'[OF _ assms(1) _ assms(2), of "[]" "{}"]
  unfolding constr_sem_d_def
  by auto

theorem stateful_typing_result:
  assumes "wfsst A"
  and "tfrsst A"
  and "wftrms (trmssst A)"
  and "interpretationsubst I"
  and "I ⊨s A"
  obtains Iτ
  where "interpretationsubst Iτ"
  and "Iτ ⊨s A"
  and "wtsubst Iτ"
  and "wftrms (subst_range Iτ)"
proof -
  obtain A' where A':
    "A' ∈ set (tr A [])" "I ⊨ ⟨A'⟩"
    using tr_sem_equiv[of A] assms(1,4,5)
    by auto

  have *: "wfst {} A'"
    "fvst A' ∩ bvarsst A' = {}"
    "tfrst A'" "wftrms (trmsst A')"
  using tr_wf[OF A'(1) assms(1,3)]
    tr_tfr[OF A'(1) assms(2)] assms(1)
  by metis+

```

```

obtain  $\mathcal{I}_\tau$  where  $\mathcal{I}_\tau$ :
  "interpretationsubst  $\mathcal{I}_\tau$ " "[{};  $\mathcal{A}'$ ]d  $\mathcal{I}_\tau$ "
  "wtsubst  $\mathcal{I}_\tau$ " "wftrms (subst_range  $\mathcal{I}_\tau$ )"
using wt_attack_if_tfr_attack_d
  * Ana_invar_subst' assms(4)
   $\mathcal{A}'$ (2)
unfolding constr_sem_d_def
by moura

thus ?thesis
  using that tr_sem_equiv[of  $\mathcal{A}$ ] assms(1,3)  $\mathcal{A}'$ (1)
  unfolding constr_sem_d_def
  by auto
qed

end

end

```

### 4.2.3 Proving type-flaw resistance automatically

definition pair' where

```
"pair' pair_fun d  $\equiv$  case d of (t,t')  $\Rightarrow$  Fun pair_fun [t,t']"
```

fun comp\_tfr<sub>sstp</sub> where

```

"comp_tfrsstp  $\Gamma$  pair_fun ( $\langle \_ : t \doteq t' \rangle$ ) = (mgu t t'  $\neq$  None  $\longrightarrow \Gamma t = \Gamma t'$ )"
| "comp_tfrsstp  $\Gamma$  pair_fun ( $\forall X(\forall \neq : F \vee \notin : F')$ ) = (
  (F' = []  $\wedge (\forall x \in fv_{pairs} F - set X. is\_Var (\Gamma (Var x)))) \vee$ 
  ( $\forall u \in subterms_{set} (trms_{pairs} F \cup pair' pair\_fun ' set F').$ 
    is_Fun u  $\longrightarrow$  (args u = []  $\vee (\exists s \in set (args u). s \notin Var ' set X)))$ )"
| "comp_tfrsstp _ _ _ = True"

```

definition comp\_tfr<sub>sst</sub> where

```

"comp_tfrsst arity Ana  $\Gamma$  pair_fun M S  $\equiv$ 
  list_all (comp_tfrsstp  $\Gamma$  pair_fun) S  $\wedge$ 
  list_all (wftrm' arity) (trms_listsst S)  $\wedge$ 
  has_all_wt_instances_of  $\Gamma$  (trmssst S  $\cup$  pair' pair_fun ' setopssst S) (set M)  $\wedge$ 
  comp_tfrset arity Ana  $\Gamma$  M"

```

locale stateful\_typed\_model' = stateful\_typed\_model arity public Ana  $\Gamma$  Pair

```

for arity::"'fun  $\Rightarrow$  nat"
and public::"'fun  $\Rightarrow$  bool"
and Ana::"('fun, (('fun, 'atom::finite) term_type  $\times$  nat)) term
 $\Rightarrow$  (('fun, (('fun, 'atom) term_type  $\times$  nat)) term list
 $\times$  ('fun, (('fun, 'atom) term_type  $\times$  nat)) term list)"
and  $\Gamma$ ::"('fun, (('fun, 'atom) term_type  $\times$  nat)) term  $\Rightarrow$  ('fun, 'atom) term_type"
and Pair::"'fun"

```

+

```

assumes  $\Gamma\_Var\_fst'$ : " $\bigwedge \tau n m. \Gamma (Var (\tau, n)) = \Gamma (Var (\tau, m))$ "
and Ana_const': " $\bigwedge c T. arity c = 0 \implies Ana (Fun c T) = ([], [])$ "

```

begin

sublocale typed\_model'

by (unfold\_locales, rule  $\Gamma\_Var\_fst'$ , metis Ana\_const', metis Ana\_subst')

lemma pair\_code:

```
"pair d = pair' Pair d"
```

by (simp add: pair\_def pair'\_def)

lemma tfr<sub>sstp</sub>-is\_comp\_tfr<sub>sstp</sub>: "tfr<sub>sstp</sub> a = comp\_tfr<sub>sstp</sub>  $\Gamma$  Pair a"

proof (cases a)

```
case (Equality ac t t')
```

```

thus ?thesis
  using mgu_always_unifies[of t _ t'] mgu_gives_MGU[of t t']
  by auto
next
case (NegChecks X F F')
thus ?thesis
  using tfrsstp.sims(2)[of X F F']
  comp_tfrsstp.sims(2)[of  $\Gamma$  Pair X F F']
  Fun_range_case(2)[of "subtermsset (trmspairs F  $\cup$  pair ' set F')"]
  unfolding is_Var_def pair_code[symmetric]
  by auto
qed auto

lemma tfrsst_if_comp_tfrsst:
  assumes "comp_tfrsst arity Ana  $\Gamma$  Pair M S"
  shows "tfrsst S"
unfolding tfrsst_def
proof
  have comp_tfrset_M: "comp_tfrset arity Ana  $\Gamma$  M"
  using assms unfolding comp_tfrsst_def by blast

  have wftrms_M: "wftrms (set M)"
  and wftrms_S: "wftrms (trmssst S  $\cup$  pair ' setopssst S)"
  and S_trms_instance_M: "has_all_wt_instances_of  $\Gamma$  (trmssst S  $\cup$  pair ' setopssst S) (set M)"
  using assms setopssst_wftrms(2)[of S] trms_listsst_is_trmssst[of S]
  unfolding comp_tfrsst_def comp_tfrset_def list_all_iff pair_code[symmetric] wftrm_code[symmetric]
  finite_SMP_representation_def
  by (meson, meson, blast, meson)

  show "tfrset (trmssst S  $\cup$  pair ' setopssst S)"
  using tfr_subset(3)[OF tfrset_if_comp_tfrset[OF comp_tfrset_M] SMP_SMP_subset]
  SMP_I'[OF wftrms_S wftrms_M S_trms_instance_M]
  by blast

  have "list_all (comp_tfrsstp  $\Gamma$  Pair) S" by (metis assms comp_tfrsst_def)
  thus "list_all tfrsstp S" by (induct S) (simp_all add: tfrsstp_is_comp_tfrsstp)
qed

lemma tfrsst_if_comp_tfrsst':
  assumes "comp_tfrsst arity Ana  $\Gamma$  Pair (SMP0 Ana  $\Gamma$  (trms_listsst S@map pair (setops_listsst S))) S"
  shows "tfrsst S"
by (rule tfrsst_if_comp_tfrsst[OF assms])

end

end

```



# 5 The Parallel Composition Result for Non-Stateful Protocols

In this chapter, we formalize and prove a compositionality result for security protocols. This work is an extension of the work described in [4] and [1, chapter 5].

## 5.1 Labeled Strands (Labeled\_Strands)

```
theory Labeled_Strands
imports Strands_and_Constraints
begin
```

### 5.1.1 Definitions: Labeled Strands and Constraints

```
datatype 'l strand_label =
  LabelN (the_LabelN: "'l") ("ln _")
| LabelS ("★")
```

Labeled strands are strands whose steps are equipped with labels

```
type_synonym ('a,'b,'c) labeled_strand_step = "'c strand_label × ('a,'b) strand_step"
type_synonym ('a,'b,'c) labeled_strand = "('a,'b,'c) labeled_strand_step list"
```

```
abbreviation is_LabelN where "is_LabelN n x  $\equiv$  fst x = ln n"
abbreviation is_LabelS where "is_LabelS x  $\equiv$  fst x = ★"
```

```
definition unlabel where "unlabel S  $\equiv$  map snd S"
definition proj where "proj n S  $\equiv$  filter ( $\lambda s$ . is_LabelN n s  $\vee$  is_LabelS s) S"
abbreviation proj_unl where "proj_unl n S  $\equiv$  unlabel (proj n S)"
```

```
abbreviation wfrestrictedvars_lst where "wfrestrictedvars_lst S  $\equiv$  wfrestrictedvars_st (unlabel S)"
```

```
abbreviation subst_apply_labeled_strand_step (infix "·lstp" 51) where
  "x ·lstp  $\vartheta$   $\equiv$  (case x of (l, s)  $\Rightarrow$  (l, s ·stp  $\vartheta$ ))"
```

```
abbreviation subst_apply_labeled_strand (infix "·lst" 51) where
  "S ·lst  $\vartheta$   $\equiv$  map ( $\lambda x$ . x ·lstp  $\vartheta$ ) S"
```

```
abbreviation trms_lst where "trms_lst S  $\equiv$  trms_st (unlabel S)"
abbreviation trms_proj_lst where "trms_proj_lst n S  $\equiv$  trms_st (proj_unl n S)"
```

```
abbreviation vars_lst where "vars_lst S  $\equiv$  vars_st (unlabel S)"
abbreviation vars_proj_lst where "vars_proj_lst n S  $\equiv$  vars_st (proj_unl n S)"
```

```
abbreviation bvars_lst where "bvars_lst S  $\equiv$  bvars_st (unlabel S)"
abbreviation fv_lst where "fv_lst S  $\equiv$  fv_st (unlabel S)"
```

```
abbreviation wf_lst where "wf_lst V S  $\equiv$  wf_st V (unlabel S)"
```

### 5.1.2 Lemmata: Projections

```
lemma is_LabelS_proj_iff_not_is_LabelN:
  "list_all is_LabelS (proj l A)  $\longleftrightarrow$   $\neg$ list_ex (is_LabelN l) A"
by (induct A) (auto simp add: proj_def)
```

```
lemma proj_subset_if_no_label:
```

```

assumes "¬list_ex (is_LabelN l) A"
shows "set (proj l A) ⊆ set (proj l' A)"
  and "set (proj_unl l A) ⊆ set (proj_unl l' A)"
using assms by (induct A) (auto simp add: unlabeled_def proj_def)

lemma proj_in_setD:
  assumes a: "a ∈ set (proj l A)"
  obtains k b where "a = (k, b)" "k = (ln l) ∨ k = *"
using that a unfolding proj_def by (cases a) auto

lemma proj_set_mono:
  assumes "set A ⊆ set B"
  shows "set (proj n A) ⊆ set (proj n B)"
  and "set (proj_unl n A) ⊆ set (proj_unl n B)"
using assms unfolding proj_def unlabeled_def by auto

lemma unlabeled_nil[simp]: "unlabeled [] = []"
by (simp add: unlabeled_def)

lemma unlabeled_mono: "set A ⊆ set B ⟹ set (unlabeled A) ⊆ set (unlabeled B)"
by (auto simp add: unlabeled_def)

lemma unlabeled_in: "(l,x) ∈ set A ⟹ x ∈ set (unlabeled A)"
unfolding unlabeled_def by force

lemma unlabeled_mem_has_labeled: "x ∈ set (unlabeled A) ⟹ ∃l. (l,x) ∈ set A"
unfolding unlabeled_def by auto

lemma proj_nil[simp]: "proj n [] = []" "proj_unl n [] = []"
unfolding unlabeled_def proj_def by auto

lemma singleton_lst_proj[simp]:
  "proj_unl l [(ln l, a)] = [a]"
  "l ≠ l' ⟹ proj_unl l' [(ln l, a)] = []"
  "proj_unl l [(*, a)] = [a]"
  "unlabeled [(l'', a)] = [a]"
unfolding proj_def unlabeled_def by simp_all

lemma unlabeled_nil_only_if_nil[simp]: "unlabeled A = [] ⟹ A = []"
unfolding unlabeled_def by auto

lemma unlabeled_Cons[simp]:
  "unlabeled ((l,a)#A) = a#unlabeled A"
  "unlabeled (b#A) = snd b#unlabeled A"
unfolding unlabeled_def by simp_all

lemma unlabeled_append[simp]: "unlabeled (A@B) = unlabeled A@unlabeled B"
unfolding unlabeled_def by auto

lemma proj_Cons[simp]:
  "proj n ((ln n,a)#A) = (ln n,a)#proj n A"
  "proj n ((*,a)#A) = (*,a)#proj n A"
  "m ≠ n ⟹ proj n ((ln m,a)#A) = proj n A"
  "l = (ln n) ⟹ proj n ((l,a)#A) = (l,a)#proj n A"
  "l = * ⟹ proj n ((l,a)#A) = (l,a)#proj n A"
  "fst b ≠ * ⟹ fst b ≠ (ln n) ⟹ proj n (b#A) = proj n A"
unfolding proj_def by auto

lemma proj_append[simp]:
  "proj l (A'@B') = proj l A'@proj l B'"
  "proj_unl l (A@B) = proj_unl l A@proj_unl l B"
unfolding proj_def unlabeled_def by auto

```

```

lemma proj_unl_cons[simp]:
  "proj_unl 1 ((ln 1, a)#A) = a#proj_unl 1 A"
  "1 ≠ 1' ⇒ proj_unl 1' ((ln 1, a)#A) = proj_unl 1' A"
  "proj_unl 1 ((*, a)#A) = a#proj_unl 1 A"
unfolding proj_def unlabeled_def by simp_all

lemma trms_unlabel_proj[simp]:
  "trms_stp (snd (ln 1, x)) ⊆ trms_projlst 1 [(ln 1, x)]"
by auto

lemma trms_unlabel_star[simp]:
  "trms_stp (snd (*, x)) ⊆ trms_projlst 1 [(*, x)]"
by auto

lemma trms_lst_union[simp]: "trms_lst A = (⋃ l. trms_projlst 1 A)"
proof (induction A)
  case (Cons a A)
  obtain 1 s where ls: "a = (1,s)" by maura
  have "trms_lst [a] = (⋃ l. trms_projlst 1 [a])"
  proof -
    have *: "trms_lst [a] = trms_stp s" using ls by simp
    show ?thesis
    proof (cases 1)
      case (LabelN n)
      hence "trms_projlst n [a] = trms_stp s" using ls by simp
      moreover have "∀ m. n ≠ m → trms_projlst m [a] = {}" using ls LabelN by auto
      ultimately show ?thesis using * ls by fastforce
    next
      case LabelS
      hence "∀ l. trms_projlst l [a] = trms_stp s" using ls by auto
      thus ?thesis using * ls by fastforce
    qed
  qed
  moreover have "∀ l. trms_projlst 1 (a#A) = trms_projlst 1 [a] ∪ trms_projlst 1 A"
  unfolding unlabeled_def proj_def by auto
  hence "(⋃ l. trms_projlst l (a#A)) = (⋃ l. trms_projlst l [a]) ∪ (⋃ l. trms_projlst l A)" by auto
  ultimately show ?case using Cons.IH ls by auto
qed simp

lemma trms_lst_append[simp]: "trms_lst (A@B) = trms_lst A ∪ trms_lst B"
by (metis trms_st_append unlabeled_append)

lemma trms_projlst_append[simp]: "trms_projlst 1 (A@B) = trms_projlst 1 A ∪ trms_projlst 1 B"
by (metis (no_types, lifting) filter_append proj_def trms_lst_append)

lemma trms_projlst_subset[simp]:
  "trms_projlst 1 A ⊆ trms_projlst 1 (A@B)"
  "trms_projlst 1 B ⊆ trms_projlst 1 (A@B)"
using trms_projlst_append[of 1] by blast+

lemma trms_lst_subset[simp]:
  "trms_lst A ⊆ trms_lst (A@B)"
  "trms_lst B ⊆ trms_lst (A@B)"
proof (induction A)
  case (Cons a A)
  obtain 1 s where *: "a = (1,s)" by maura
  { case 1 thus ?case using Cons * by auto }
  { case 2 thus ?case using Cons * by auto }
qed simp_all

lemma vars_lst_union: "vars_lst A = (⋃ l. vars_projlst 1 A)"
proof (induction A)
  case (Cons a A)

```

```

obtain l s where ls: "a = (l,s)" by moura
have "varslst [a] = (⋃ l. varsprojlst l [a])"
proof -
  have *: "varslst [a] = varsstp s" using ls by auto
  show ?thesis
  proof (cases l)
    case (LabelN n)
    hence "varsprojlst n [a] = varsstp s" using ls by simp
    moreover have "∀m. n ≠ m ⟶ varsprojlst m [a] = {}" using ls LabelN by auto
    ultimately show ?thesis using * ls by fast
  next
    case LabelS
    hence "∀l. varsprojlst l [a] = varsstp s" using ls by auto
    thus ?thesis using * ls by fast
  qed
qed
moreover have "∀l. varsprojlst l (a#A) = varsprojlst l [a] ∪ varsprojlst l A"
  unfolding unlabel_def proj_def by auto
hence "(⋃ l. varsprojlst l (a#A)) = (⋃ l. varsprojlst l [a]) ∪ (⋃ l. varsprojlst l A)"
  using strand_vars_split(1) by auto
ultimately show ?case using Cons.IH ls strand_vars_split(1) by auto
qed simp

lemma unlabel_Cons_inv:
  "unlabel A = b#B ⟹ ∃A'. (∃n. A = (ln n, b)#A') ∨ A = (*, b)#A'"
proof -
  assume *: "unlabel A = b#B"
  then obtain l A' where "A = (l,b)#A'" unfolding unlabel_def by moura
  thus "∃A'. (∃l. A = (ln l, b)#A') ∨ A = (*, b)#A'" by (metis strand_label.exhaust)
qed

lemma unlabel_snoc_inv:
  "unlabel A = B@[b] ⟹ ∃A'. (∃n. A = A'@[ln n, b]) ∨ A = A'@[(*, b)]"
proof -
  assume *: "unlabel A = B@[b]"
  then obtain A' l where "A = A'@[l,b]"
    unfolding unlabel_def by (induct A rule: List.rev_induct) auto
  thus "∃A'. (∃n. A = A'@[ln n, b]) ∨ A = A'@[(*, b)]" by (cases l) auto
qed

lemma proj_idem[simp]: "proj l (proj l A) = proj l A"
unfolding proj_def by auto

lemma proj_ikst_is_proj_rcv_set:
  "ikst (proj_unl n A) = {t. (ln n, Receive t) ∈ set A ∨ (*, Receive t) ∈ set A}"
using ikst_is_rcv_set unfolding unlabel_def proj_def by force

lemma unlabel_ikst_is_rcv_set:
  "ikst (unlabel A) = {t | ∃ l t. (l, Receive t) ∈ set A}"
using ikst_is_rcv_set unfolding unlabel_def by force

lemma proj_ikst_union_is_unlabel_ik:
  "ikst (unlabel A) = (⋃ l. ikst (proj_unl l A))"
proof
  show "(⋃ l. ikst (proj_unl l A)) ⊆ ikst (unlabel A)"
    using unlabel_ikst_is_rcv_set[of A] proj_ikst_is_proj_rcv_set[of _ A] by auto

  show "ikst (unlabel A) ⊆ (⋃ l. ikst (proj_unl l A))"
  proof
    fix t assume "t ∈ ikst (unlabel A)"
    then obtain l where "(l, Receive t) ∈ set A"
      using ikst_is_rcv_set unlabel_mem_has_label[of _ A]
      by moura
  qed

```



```

    thus "t ∈ (⋃ l. ikst (proj_unl l A))" using proj_ikst_is_proj_rcv_set[of _ A] by (cases l) auto
qed
qed

```

```

lemma proj_ik_append[simp]:
  "ikst (proj_unl l (A@B)) = ikst (proj_unl l A) ∪ ikst (proj_unl l B)"
using proj_append(2)[of l A B] ik_append by auto

```

```

lemma proj_ik_append_subst_all:
  "ikst (proj_unl l (A@B)) ·set I = (ikst (proj_unl l A) ·set I) ∪ (ikst (proj_unl l B) ·set I)"
using proj_ik_append[of l] by auto

```

```

lemma ik_proj_subset[simp]: "ikst (proj_unl n A) ⊆ trms_projlst n A"
by auto

```

```

lemma prefix_proj:
  "prefix A B ⇒ prefix (unlabel A) (unlabel B)"
  "prefix A B ⇒ prefix (proj n A) (proj n B)"
  "prefix A B ⇒ prefix (proj_unl n A) (proj_unl n B)"
unfolding prefix_def unlabel_def proj_def by auto

```

### 5.1.3 Lemmata: Well-formedness

```

lemma wfvarsoccsst_proj_union:
  "wfvarsoccsst (unlabel A) = (⋃ l. wfvarsoccsst (proj_unl l A))"
proof (induction A)
  case (Cons a A)
  obtain l s where ls: "a = (l,s)" by mouna
  have "wfvarsoccsst (unlabel [a]) = (⋃ l. wfvarsoccsst (proj_unl l [a]))"
  proof -
    have *: "wfvarsoccsst (unlabel [a]) = wfvarsoccsstp s" using ls by auto
    show ?thesis
    proof (cases l)
      case (LabelN n)
      hence "wfvarsoccsst (proj_unl n [a]) = wfvarsoccsstp s" using ls by simp
      moreover have "∀ m. n ≠ m ⇒ wfvarsoccsst (proj_unl m [a]) = {}" using ls LabelN by auto
      ultimately show ?thesis using * ls by fast
    next
      case LabelS
      hence "∀ l. wfvarsoccsst (proj_unl l [a]) = wfvarsoccsstp s" using ls by auto
      thus ?thesis using * ls by fast
    qed
  qed
  moreover have
    "wfvarsoccsst (proj_unl l (a#A)) =
      wfvarsoccsst (proj_unl l [a]) ∪ wfvarsoccsst (proj_unl l A)"
  for l
  unfolding unlabel_def proj_def by auto
  hence "(⋃ l. wfvarsoccsst (proj_unl l (a#A))) =
    (⋃ l. wfvarsoccsst (proj_unl l [a])) ∪ (⋃ l. wfvarsoccsst (proj_unl l A))"
  using strand_vars_split(1) by auto
  ultimately show ?case using Cons.IH ls strand_vars_split(1) by auto
qed simp

```

```

lemma wf_if_wf_proj:
  assumes "∀ l. wfst V (proj_unl l A)"
  shows "wfst V (unlabel A)"
using assms
proof (induction A arbitrary: V rule: List.rev_induct)
  case (snoc a A)
  hence IH: "wfst V (unlabel A)" using proj_append(2)[of _ A] by auto
  obtain b l where b: "a = (ln l, b) ∨ a = (*, b)" by (cases a, metis strand_label.exhaust)
  hence *: "wfst V (proj_unl l A@[b])"

```

```

by (metis snoc.premis proj_append(2) singleton_lst_proj(1) proj_unl_cons(1,3))
thus ?case using IH b snoc.premis proj_append(2)[of l A "[a]"] unlabeled_append[of A "[a]"]
proof (cases b)
  case (Receive t)
  have "fv t  $\subseteq$  wfvarsoccsst (unlabel A)  $\cup$  V"
  proof
    fix x assume "x  $\in$  fv t"
    hence "x  $\in$  V  $\cup$  wfvarsoccsst (proj_unl l A)" using wf_append_exec[OF *] b Receive by auto
    thus "x  $\in$  wfvarsoccsst (unlabel A)  $\cup$  V" using wfvarsoccsst-proj_union[of A] by auto
  qed
  hence "fv t  $\subseteq$  wfrestrictedvarsst (unlabel A)  $\cup$  V"
    using vars_snd_rcv_strand_subset2(4)[of "unlabel A"] by blast
  hence "wfst V (unlabel A@[Receive t])" by (rule wf_rcv_append''[OF IH])
  thus ?thesis using b Receive unlabeled_append[of A "[a]"] by auto
next
  case (Equality ac s t)
  have "fv t  $\subseteq$  wfvarsoccsst (unlabel A)  $\cup$  V" when "ac = Assign"
  proof
    fix x assume "x  $\in$  fv t"
    hence "x  $\in$  V  $\cup$  wfvarsoccsst (proj_unl l A)" using wf_append_exec[OF *] b Equality that by auto
    thus "x  $\in$  wfvarsoccsst (unlabel A)  $\cup$  V" using wfvarsoccsst-proj_union[of A] by auto
  qed
  hence "fv t  $\subseteq$  wfrestrictedvarslst A  $\cup$  V" when "ac = Assign"
    using vars_snd_rcv_strand_subset2(4)[of "unlabel A"] that by blast
  hence "wfst V (unlabel A@[Equality ac s t])"
    by (cases ac) (metis wf_eq_append''[OF IH], metis wf_eq_check_append''[OF IH])
  thus ?thesis using b Equality unlabeled_append[of A "[a]"] by auto
qed auto
qed simp
end

```

## 5.2 Parallel Compositionality of Security Protocols (Parallel\_Compositionality)

```

theory Parallel_Compositionality
imports Typing_Result Labeled_Strands
begin

```

### 5.2.1 Definitions: Labeled Typed Model Locale

```

locale labeled_typed_model = typed_model arity public Ana  $\Gamma$ 
  for arity::"'fun  $\Rightarrow$  nat"
  and public::"'fun  $\Rightarrow$  bool"
  and Ana::"'(fun, 'var) term  $\Rightarrow$  (('fun, 'var) term list  $\times$  ('fun, 'var) term list)"
  and  $\Gamma$ ::"'(fun, 'var) term  $\Rightarrow$  ('fun, 'atom::finite) term_type"
+
  fixes label_witness1 and label_witness2::"'lbl"
  assumes at_least_2_labels: "label_witness1  $\neq$  label_witness2"
begin

```

The Ground Sub-Message Patterns (GSMP)

```

definition GSMP::"'(fun, 'var) terms  $\Rightarrow$  ('fun, 'var) terms" where
  "GSMP P  $\equiv$  {t  $\in$  SMP P. fv t = {}}"

```

definition typing\_cond where

```

"typing_cond  $\mathcal{A} \equiv$ 
  wfst {}  $\mathcal{A} \wedge$ 
  fvst  $\mathcal{A} \cap$  bvarsst  $\mathcal{A} = \{\}$   $\wedge$ 
  tfrst  $\mathcal{A} \wedge$ 
  wftrms (trmsst  $\mathcal{A}) \wedge$ 
  Ana_invar_subst (ikst  $\mathcal{A} \cup$  assignment_rhsst  $\mathcal{A})"$ 

```

### 5.2.2 Definitions: GSMP Disjointedness and Parallel Composability

**definition** *GSMP\_disjoint* where

"GSMP\_disjoint P1 P2 Secrets  $\equiv$  GSMP P1  $\cap$  GSMP P2  $\subseteq$  Secrets  $\cup$  {m. {}  $\vdash_c$  m}"

**definition** *declassified<sub>lst</sub>* where

"declassified<sub>lst</sub> (A::('fun,'var,'lbl) labeled\_strand)  $\mathcal{I} \equiv$  {t. (\*, Receive t)  $\in$  set A}  $\cdot_{set}$   $\mathcal{I}$ "

**definition** *par\_comp* where

"par\_comp (A::('fun,'var,'lbl) labeled\_strand) (Secrets::('fun,'var) terms)  $\equiv$   
 ( $\forall$  l1 l2. l1  $\neq$  l2  $\longrightarrow$  GSMP\_disjoint (trms\_proj<sub>lst</sub> l1 A) (trms\_proj<sub>lst</sub> l2 A) Secrets)  $\wedge$   
 ( $\forall$  s  $\in$  Secrets.  $\forall$  s'  $\in$  subterms s. {}  $\vdash_c$  s'  $\vee$  s'  $\in$  Secrets)  $\wedge$   
 ground Secrets"

**definition** *strand\_leaks<sub>lst</sub>* where

"strand\_leaks<sub>lst</sub> A Sec  $\mathcal{I} \equiv$  ( $\exists$  t  $\in$  Sec - declassified<sub>lst</sub> A  $\mathcal{I}$ .  $\exists$  l. ( $\mathcal{I} \models$  (proj\_unl l A@[Send t])))"

### 5.2.3 Definitions: Homogeneous and Numbered Intruder Deduction Variants

**definition** *proj\_specific* where

"proj\_specific n t A Secrets  $\equiv$  t  $\in$  GSMP (trms\_proj<sub>lst</sub> n A) - (Secrets  $\cup$  {m. {}  $\vdash_c$  m})"

**definition** *heterogeneous<sub>lst</sub>* where

"heterogeneous<sub>lst</sub> t A Secrets  $\equiv$  (  
 ( $\exists$  l1 l2.  $\exists$  s1  $\in$  subterms t.  $\exists$  s2  $\in$  subterms t.  
 l1  $\neq$  l2  $\wedge$  proj\_specific l1 s1 A Secrets  $\wedge$  proj\_specific l2 s2 A Secrets))"

**abbreviation** *homogeneous<sub>lst</sub>* where

"homogeneous<sub>lst</sub> t A Secrets  $\equiv$   $\neg$ heterogeneous<sub>lst</sub> t A Secrets"

**definition** *intruder\_deduct\_hom::*

"('fun,'var) terms  $\Rightarrow$  ('fun,'var,'lbl) labeled\_strand  $\Rightarrow$  ('fun,'var) terms  $\Rightarrow$  ('fun,'var) term  
 $\Rightarrow$  bool" ("<\_;-;->  $\vdash_{hom}$  -" 50)

**where**

"<M; A; Sec>  $\vdash_{hom}$  t  $\equiv$  <M;  $\lambda$ t. homogeneous<sub>lst</sub> t A Sec  $\wedge$  t  $\in$  GSMP (trms<sub>lst</sub> A)>  $\vdash_r$  t"

**lemma** *intruder\_deduct\_hom\_AxiomH[simp]:*

assumes "t  $\in$  M"

shows "<M; A; Sec>  $\vdash_{hom}$  t"

**using** intruder\_deduct\_restricted.AxiomR[of t M] **assms**

**unfolding** intruder\_deduct\_hom\_def

**by** blast

**lemma** *intruder\_deduct\_hom\_ComposeH[simp]:*

assumes "length X = arity f" "public f" " $\bigwedge$ x. x  $\in$  set X  $\implies$  <M; A; Sec>  $\vdash_{hom}$  x"

and "homogeneous<sub>lst</sub> (Fun f X) A Sec" "Fun f X  $\in$  GSMP (trms<sub>lst</sub> A)"

shows "<M; A; Sec>  $\vdash_{hom}$  Fun f X"

**proof** -

let ?Q = " $\lambda$ t. homogeneous<sub>lst</sub> t A Sec  $\wedge$  t  $\in$  GSMP (trms<sub>lst</sub> A)"

show ?thesis

using intruder\_deduct\_restricted.ComposeR[of X f M ?Q] **assms**

unfolding intruder\_deduct\_hom\_def

**by** blast

**qed**

**lemma** *intruder\_deduct\_hom-DecomposeH:*

assumes "<M; A; Sec>  $\vdash_{hom}$  t" "Ana t = (K, T)" " $\bigwedge$ k. k  $\in$  set K  $\implies$  <M; A; Sec>  $\vdash_{hom}$  k" "t<sub>i</sub>  $\in$  set T"

shows "<M; A; Sec>  $\vdash_{hom}$  t<sub>i</sub>"

**proof** -

let ?Q = " $\lambda$ t. homogeneous<sub>lst</sub> t A Sec  $\wedge$  t  $\in$  GSMP (trms<sub>lst</sub> A)"

show ?thesis

using intruder\_deduct\_restricted.DecomposeR[of M ?Q t] **assms**

```

    unfolding intruder_deduct_hom_def
    by blast
qed

lemma intruder_deduct_hom_induct[consumes 1, case_names AxiomH ComposeH DecomposeH]:
  assumes "⟨M; A; Sec⟩ ⊢hom t" "∧t. t ∈ M ⇒ P M t"
    "∧X f. [length X = arity f; public f;
      ∧x. x ∈ set X ⇒ ⟨M; A; Sec⟩ ⊢hom x;
      ∧x. x ∈ set X ⇒ P M x;
      homogeneouslst (Fun f X) A Sec;
      Fun f X ∈ GSMP (trmslst A)
    ] ⇒ P M (Fun f X)"
    "∧t K T ti. [⟨M; A; Sec⟩ ⊢hom t; P M t; Ana t = (K, T);
      ∧k. k ∈ set K ⇒ ⟨M; A; Sec⟩ ⊢hom k;
      ∧k. k ∈ set K ⇒ P M k; ti ∈ set T] ⇒ P M ti"
  shows "P M t"
proof -
  let ?Q = "λt. homogeneouslst t A Sec ∧ t ∈ GSMP (trmslst A)"
  show ?thesis
    using intruder_deduct_restricted_induct[of M ?Q t "λM Q t. P M t"] assms
    unfolding intruder_deduct_hom_def
    by blast
qed

lemma ideduct_hom_mono:
  "[⟨M; A; Sec⟩ ⊢hom t; M ⊆ M'] ⇒ ⟨M'; A; Sec⟩ ⊢hom t"
using ideduct_restricted_mono[of M _ t M']
unfolding intruder_deduct_hom_def
by fast

```

### 5.2.4 Lemmata: GSMP

```

lemma GSMP_disjoint_empty[simp]:
  "GSMP_disjoint {} A Sec" "GSMP_disjoint A {} Sec"
unfolding GSMP_disjoint_def GSMP_def by fastforce+

lemma GSMP_mono:
  assumes "N ⊆ M"
  shows "GSMP N ⊆ GSMP M"
using SMP_mono[OF assms] unfolding GSMP_def by fast

lemma GSMP_SMP_mono:
  assumes "SMP N ⊆ SMP M"
  shows "GSMP N ⊆ GSMP M"
using assms unfolding GSMP_def by fast

lemma GSMP_subterm:
  assumes "t ∈ GSMP M" "t' ⊆ t"
  shows "t' ∈ GSMP M"
using SMP.Subterm[of t M t'] ground_subterm[of t t'] assms unfolding GSMP_def by auto

lemma GSMP_subterms: "subtermsset (GSMP M) = GSMP M"
using GSMP_subterm[of _ M] by blast

lemma GSMP_Ana_key:
  assumes "t ∈ GSMP M" "Ana t = (K, T)" "k ∈ set K"
  shows "k ∈ GSMP M"
using SMP.Ana[of t M K T k] Ana_keys_fv[of t K T] assms unfolding GSMP_def by auto

lemma GSMP_append[simp]: "GSMP (trmslst (A@B)) = GSMP (trmslst A) ∪ GSMP (trmslst B)"
using SMP_union[of "trmslst A" "trmslst B"] trmslst_append[of A B] unfolding GSMP_def by auto

lemma GSMP_union: "GSMP (A ∪ B) = GSMP A ∪ GSMP B"

```

```

using SMP_union[of A B] unfolding GSMP_def by auto

lemma GSMP_Union: "GSMP (trmslst A) = (⋃ l. GSMP (trmsprojlst l A))"
proof -
  define P where "P ≡ (λ l. trmsprojlst l A)"
  define Q where "Q ≡ trmslst A"
  have "SMP (⋃ l. P l) = (⋃ l. SMP (P l))" "Q = (⋃ l. P l)"
    unfolding P_def Q_def by (metis SMP_Union, metis trmslst_union)
  hence "GSMP Q = (⋃ l. GSMP (P l))" unfolding GSMP_def by auto
  thus ?thesis unfolding P_def Q_def by metis
qed

lemma in_GSMP_in_proj: "t ∈ GSMP (trmslst A) ⇒ ∃ n. t ∈ GSMP (trmsprojlst n A)"
using GSMP_Union[of A] by blast

lemma in_proj_in_GSMP: "t ∈ GSMP (trmsprojlst n A) ⇒ t ∈ GSMP (trmslst A)"
using GSMP_Union[of A] by blast

lemma GSMP_disjointE:
  assumes A: "GSMP_disjoint (trmsprojlst n A) (trmsprojlst m A) Sec"
  shows "GSMP (trmsprojlst n A) ∩ GSMP (trmsprojlst m A) ⊆ Sec ∪ {m. {} ⊢c m}"
using assms unfolding GSMP_disjoint_def by auto

lemma GSMP_disjoint_term:
  assumes "GSMP_disjoint (trmsprojlst l A) (trmsprojlst l' A) Sec"
  shows "t ∉ GSMP (trmsprojlst l A) ∨ t ∉ GSMP (trmsprojlst l' A) ∨ t ∈ Sec ∨ {} ⊢c t"
using assms unfolding GSMP_disjoint_def by blast

lemma GSMP_wt_subst_subset:
  assumes "t ∈ GSMP (M ·set I)" "wtsubst I" "wftrms (substrange I)"
  shows "t ∈ GSMP M"
using SMP_wt_subst_subset[OF _ assms(2,3), of t M] assms(1) unfolding GSMP_def by simp

lemma GSMP_wt_substI:
  assumes "t ∈ M" "wtsubst I" "wftrms (substrange I)" "interpretationsubst I"
  shows "t · I ∈ GSMP M"
proof -
  have "t ∈ SMP M" using assms(1) by auto
  hence *: "t · I ∈ SMP M" using SMP.Substitution assms(2,3) wftrm_substrange_iff[of I] by simp
  moreover have "fv (t · I) = {}"
    using assms(1) interpretation_grounds_all'[OF assms(4)]
    by auto
  ultimately show ?thesis unfolding GSMP_def by simp
qed

lemma GSMP_disjoint_subset:
  assumes "GSMP_disjoint L R S" "L' ⊆ L" "R' ⊆ R"
  shows "GSMP_disjoint L' R' S"
using assms(1) SMP_mono[OF assms(2)] SMP_mono[OF assms(3)]
by (auto simp add: GSMP_def GSMP_disjoint_def)

lemma GSMP_disjoint_fst_specific_not_snd_specific:
  assumes "GSMP_disjoint (trmsprojlst l A) (trmsprojlst l' A) Sec" "l ≠ l'"
  and "projspecific l m A Sec"
  shows "¬projspecific l' m A Sec"
using assms by (fastforce simp add: GSMP_disjoint_def projspecific_def)

lemma GSMP_disjoint_snd_specific_not_fst_specific:
  assumes "GSMP_disjoint (trmsprojlst l A) (trmsprojlst l' A) Sec"
  and "projspecific l' m A Sec"
  shows "¬projspecific l m A Sec"
using assms by (auto simp add: GSMP_disjoint_def projspecific_def)

```

```

lemma GSMP_disjoint_intersection_not_specific:
  assumes "GSMP_disjoint (trms_proj_lst l A) (trms_proj_lst l' A) Sec"
  and "t ∈ Sec ∨ {} ⊢c t"
  shows "¬proj_specific l t A Sec" "¬proj_specific l' t A Sec"
using assms by (auto simp add: GSMP_disjoint_def proj_specific_def)

```

### 5.2.5 Lemmata: Intruder Knowledge and Declassification

```

lemma ik_proj_subst_GSMP_subset:
  assumes I: "wt_subst I" "wf_trms (subst_range I)" "interpretation_subst I"
  shows "ikst (proj_unl n A) ·set I ⊆ GSMP (trms_proj_lst n A)"
proof
  fix t assume "t ∈ ikst (proj_unl n A) ·set I"
  hence *: "t ∈ trms_proj_lst n A ·set I" by auto
  then obtain s where "s ∈ trms_proj_lst n A" "t = s · I" by auto
  hence "t ∈ SMP (trms_proj_lst n A)" using SMP_I I(1,2) wf_trm_subst_range_iff[of I] by simp
  moreover have "fv t = {}"
    using * interpretation_grounds_all'[OF I(3)]
    by auto
  ultimately show "t ∈ GSMP (trms_proj_lst n A)" unfolding GSMP_def by simp
qed

```

```

lemma declassified_proj_ik_subset: "declassifiedlst A I ⊆ ikst (proj_unl n A) ·set I"
proof (induction A)
  case (Cons a A) thus ?case
    using proj_ik_append[of n "[a]" A] by (auto simp add: declassifiedlst_def)
qed (simp add: declassifiedlst_def)

```

```

lemma declassified_proj_GSMP_subset:
  assumes I: "wt_subst I" "wf_trms (subst_range I)" "interpretation_subst I"
  shows "declassifiedlst A I ⊆ GSMP (trms_proj_lst n A)"
by (rule subset_trans[OF declassified_proj_ik_subset ik_proj_subst_GSMP_subset[OF I]])

```

```

lemma declassified_subterms_proj_GSMP_subset:
  assumes I: "wt_subst I" "wf_trms (subst_range I)" "interpretation_subst I"
  shows "subtermsset (declassifiedlst A I) ⊆ GSMP (trms_proj_lst n A)"
proof
  fix t assume t: "t ∈ subtermsset (declassifiedlst A I)"
  then obtain t' where t': "t' ∈ declassifiedlst A I" "t ⊆ t'" by moura
  hence "t' ∈ GSMP (trms_proj_lst n A)" using declassified_proj_GSMP_subset[OF assms] by blast
  thus "t ∈ GSMP (trms_proj_lst n A)"
    using SMP.Subterm[of t' "trms_proj_lst n A" t] ground_subterm[OF _ t'(2)] t'(2)
    unfolding GSMP_def by fast
qed

```

```

lemma declassified_secrets_subset:
  assumes A: "∀n m. n ≠ m → GSMP_disjoint (trms_proj_lst n A) (trms_proj_lst m A) Sec"
  and I: "wt_subst I" "wf_trms (subst_range I)" "interpretation_subst I"
  shows "declassifiedlst A I ⊆ Sec ∪ {m. {} ⊢c m}"
using declassified_proj_GSMP_subset[OF I] A at_least_2_labels
unfolding GSMP_disjoint_def by blast

```

```

lemma declassified_subterms_secrets_subset:
  assumes A: "∀n m. n ≠ m → GSMP_disjoint (trms_proj_lst n A) (trms_proj_lst m A) Sec"
  and I: "wt_subst I" "wf_trms (subst_range I)" "interpretation_subst I"
  shows "subtermsset (declassifiedlst A I) ⊆ Sec ∪ {m. {} ⊢c m}"
using declassified_subterms_proj_GSMP_subset[OF I, of A label_witness1]
declassified_subterms_proj_GSMP_subset[OF I, of A label_witness2]
A at_least_2_labels
unfolding GSMP_disjoint_def by fast

```

```

lemma declassified_proj_eq: "declassifiedlst A I = declassifiedlst (proj n A) I"
unfolding declassifiedlst_def proj_def by auto

```

```
lemma declassified_append: "declassifiedlst (A@B) I = declassifiedlst A I ∪ declassifiedlst B I"
unfolding declassifiedlst_def by auto
```

```
lemma declassified_prefix_subset: "prefix A B ⇒ declassifiedlst A I ⊆ declassifiedlst B I"
using declassified_append unfolding prefix_def by auto
```

### 5.2.6 Lemmata: Homogeneous and Heterogeneous Terms

```
lemma proj_specific_secrets_anti_mono:
  assumes "proj_specific l t A Sec" "Sec' ⊆ Sec"
  shows "proj_specific l t A Sec'"
using assms unfolding proj_specific_def by fast
```

```
lemma heterogeneous_secrets_anti_mono:
  assumes "heterogeneouslst t A Sec" "Sec' ⊆ Sec"
  shows "heterogeneouslst t A Sec'"
using assms proj_specific_secrets_anti_mono unfolding heterogeneouslst_def by metis
```

```
lemma homogeneous_secrets_mono:
  assumes "homogeneouslst t A Sec'" "Sec' ⊆ Sec"
  shows "homogeneouslst t A Sec"
using assms heterogeneous_secrets_anti_mono by blast
```

```
lemma heterogeneous_supterm:
  assumes "heterogeneouslst t A Sec" "t ⊆ t'"
  shows "heterogeneouslst t' A Sec"
proof -
  obtain l1 l2 s1 s2 where *:
    "l1 ≠ l2"
    "s1 ⊆ t" "proj_specific l1 s1 A Sec"
    "s2 ⊆ t" "proj_specific l2 s2 A Sec"
  using assms(1) unfolding heterogeneouslst_def by maura
  thus ?thesis
    using term.order_trans[OF *(2) assms(2)] term.order_trans[OF *(4) assms(2)]
    by (auto simp add: heterogeneouslst_def)
qed
```

```
lemma homogeneous_subterm:
  assumes "homogeneouslst t A Sec" "t' ⊆ t"
  shows "homogeneouslst t' A Sec"
by (metis assms heterogeneous_supterm)
```

```
lemma proj_specific_subterm:
  assumes "t ⊆ t'" "proj_specific l t' A Sec"
  shows "proj_specific l t A Sec ∨ t ∈ Sec ∨ {} ⊢c t"
using GSMP_subterm[OF _ assms(1)] assms(2) by (auto simp add: proj_specific_def)
```

```
lemma heterogeneous_term_is_Fun:
  assumes "heterogeneouslst t A S" shows "∃ f T. t = Fun f T"
using assms by (cases t) (auto simp add: GSMP_def heterogeneouslst_def proj_specific_def)
```

```
lemma proj_specific_is_homogeneous:
  assumes A: "∀ l l'. l ≠ l' ⇒ GSMP_disjoint (trms_projlst l A) (trms_projlst l' A) Sec"
  and t: "proj_specific l m A Sec"
  shows "homogeneouslst m A Sec"
proof
  assume "heterogeneouslst m A Sec"
  then obtain s l' where s: "s ∈ subterms m" "proj_specific l' s A Sec" "l ≠ l'"
  unfolding heterogeneouslst_def by maura
  hence "s ∈ GSMP (trms_projlst l A)" "s ∈ GSMP (trms_projlst l' A)"
  using t by (auto simp add: GSMP_def proj_specific_def)
  hence "s ∈ Sec ∨ {} ⊢c s"
```

```

using A s(3) by (auto simp add: GSMP_disjoint_def)
thus False using s(2) by (auto simp add: proj_specific_def)
qed

lemma deduct_synth_homogeneous:
  assumes "{} ⊢c t"
  shows "homogeneouslst t A Sec"
proof -
  have "∀ s ∈ subterms t. {} ⊢c s" using deduct_synth_subterm[OF assms] by auto
  thus ?thesis unfolding heterogeneouslst_def proj_specific_def by auto
qed

lemma GSMP_proj_is_homogeneous:
  assumes "∀ l l'. l ≠ l' → GSMP_disjoint (trms_projlst l A) (trms_projlst l' A) Sec"
  and "t ∈ GSMP (trms_projlst l A)" "t ∉ Sec"
  shows "homogeneouslst t A Sec"
proof
  assume "heterogeneouslst t A Sec"
  then obtain s l' where s: "s ∈ subterms t" "proj_specific l' s A Sec" "l ≠ l'"
    unfolding heterogeneouslst_def by maura
  hence "s ∈ GSMP (trms_projlst l A)" "s ∈ GSMP (trms_projlst l' A)"
    using assms by (auto simp add: GSMP_def proj_specific_def)
  hence "s ∈ Sec ∨ {} ⊢c s" using assms(1) s(3) by (auto simp add: GSMP_disjoint_def)
  thus False using s(2) by (auto simp add: proj_specific_def)
qed

lemma homogeneous_is_not_proj_specific:
  assumes "homogeneouslst m A Sec"
  shows "∃ l::'lbl. ¬proj_specific l m A Sec"
proof -
  let ?P = "λ l s. proj_specific l s A Sec"
  have "∀ l1 l2. ∀ s1 ∈ subterms m. ∀ s2 ∈ subterms m. (l1 ≠ l2 → (¬?P l1 s1 ∨ ¬?P l2 s2))"
    using assms heterogeneouslst_def by metis
  then obtain l1 l2 where "l1 ≠ l2" "¬?P l1 m ∨ ¬?P l2 m"
    by (metis term.order_refl at_least_2_labels)
  thus ?thesis by metis
qed

lemma secrets_are_homogeneous:
  assumes "∀ s ∈ Sec. P s → (∀ s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ Sec)" "s ∈ Sec" "P s"
  shows "homogeneouslst s A Sec"
using assms by (auto simp add: heterogeneouslst_def proj_specific_def)

lemma GSMP_is_homogeneous:
  assumes A: "∀ l l'. l ≠ l' → GSMP_disjoint (trms_projlst l A) (trms_projlst l' A) Sec"
  and t: "t ∈ GSMP (trmslst A)" "t ∉ Sec"
  shows "homogeneouslst t A Sec"
proof -
  obtain n where n: "t ∈ GSMP (trms_projlst n A)" using in_GSMP_in_proj[OF t(1)] by maura
  show ?thesis using GSMP_proj_is_homogeneous[OF A n t(2)] by metis
qed

lemma GSMP_intersection_is_homogeneous:
  assumes A: "∀ l l'. l ≠ l' → GSMP_disjoint (trms_projlst l A) (trms_projlst l' A) Sec"
  and t: "t ∈ GSMP (trms_projlst l A) ∩ GSMP (trms_projlst l' A)" "l ≠ l'"
  shows "homogeneouslst t A Sec"
proof -
  define M where "M ≡ GSMP (trms_projlst l A)"
  define M' where "M' ≡ GSMP (trms_projlst l' A)"

  have t_in: "t ∈ M ∩ M'" "t ∈ GSMP (trmslst A)"
    using t(1) in_proj_in_GSMP[of t _ A]
    unfolding M_def M'_def by blast+

```



```

have "M ∩ M' ⊆ Sec ∪ {m. {} ⊢c m}"
  using A GSMP_disjointE[of 1 A 1' Sec] t(2)
  unfolding M_def M'_def by presburger
moreover have "subtermsset (M ∩ M') = M ∩ M'"
  using GSMP_subterms unfolding M_def M'_def by blast
ultimately have *: "subtermsset (M ∩ M') ⊆ Sec ∪ {m. {} ⊢c m}"
  by blast

show ?thesis
proof (cases "t ∈ Sec")
  case True thus ?thesis
    using * secrets_are_homogeneous[of Sec "λt. t ∈ M ∩ M'", OF _ _ t_in(1)]
    by fast
  qed (metis GSMP_is_homogeneous[OF A t_in(2)])
qed

lemma GSMP_is_homogeneous':
  assumes A: "∀ l 1'. l ≠ 1' → GSMP_disjoint (trms_projlst l A) (trms_projlst 1' A) Sec"
  and t: "t ∈ GSMP (trmslst A)"
  "t ∉ Sec - ⋃ {GSMP (trms_projlst l1 A) ∩ GSMP (trms_projlst l2 A) | l1 l2. l1 ≠ l2}"
  shows "homogeneouslst t A Sec"
using GSMP_is_homogeneous[OF A t(1)] GSMP_intersection_is_homogeneous[OF A] t(2)
by blast

lemma declassified_secrets_are_homogeneous:
  assumes A: "∀ l 1'. l ≠ 1' → GSMP_disjoint (trms_projlst l A) (trms_projlst 1' A) Sec"
  and I: "wtsubst I" "wftrms (subst_range I)" "interpretationsubst I"
  and s: "s ∈ declassifiedlst A I"
  shows "homogeneouslst s A Sec"
proof -
  have s_in: "s ∈ GSMP (trmslst A)"
  using declassified_proj_GSMP_subset[OF I, of A label_witness1]
  in_proj_in_GSMP[of s label_witness1 A] s
  by blast

  show ?thesis
proof (cases "s ∈ Sec")
  case True thus ?thesis
    using declassified_subterms_secrets_subset[OF A I]
    secrets_are_homogeneous[of Sec "λs. s ∈ declassifiedlst A I", OF _ _ s]
    by fast
  qed (metis GSMP_is_homogeneous[OF A s_in])
qed

lemma Ana_keys_homogeneous:
  assumes A: "∀ l 1'. l ≠ 1' → GSMP_disjoint (trms_projlst l A) (trms_projlst 1' A) Sec"
  and t: "t ∈ GSMP (trmslst A)"
  and k: "Ana t = (K,T)" "k ∈ set K"
  "k ∉ Sec - ⋃ {GSMP (trms_projlst l1 A) ∩ GSMP (trms_projlst l2 A) | l1 l2. l1 ≠ l2}"
  shows "homogeneouslst k A Sec"
proof (cases "k ∈ ⋃ {GSMP (trms_projlst l1 A) ∩ GSMP (trms_projlst l2 A) | l1 l2. l1 ≠ l2}")
  case False
  hence "k ∉ Sec" using k(3) by fast
  moreover have "k ∈ GSMP (trmslst A)"
  using t SMP.Ana[OF _ k(1,2)] Ana_keys_fv[OF k(1)] k(2)
  unfolding GSMP_def by auto
  ultimately show ?thesis using GSMP_is_homogeneous[OF A, of k] by metis
qed (use GSMP_intersection_is_homogeneous[OF A] in blast)

```

### 5.2.7 Lemmata: Intruder Deduction Equivalences

```

lemma deduct_if_hom_deduct: "⟨M; A; S⟩ ⊢hom m ⇒ M ⊢ m"

```

```

using deduct_if_restricted_deduct unfolding intruder_deduct_hom_def by blast

lemma hom_deduct_if_hom_ik:
  assumes " $\langle M; A; Sec \rangle \vdash_{hom} m$ " " $\forall m \in M. \text{homogeneous}_{lst} m A Sec \wedge m \in GSMP (\text{trms}_{lst} A)$ "
  shows " $\text{homogeneous}_{lst} m A Sec \wedge m \in GSMP (\text{trms}_{lst} A)$ "
proof -
  let ?Q = " $\lambda m. \text{homogeneous}_{lst} m A Sec \wedge m \in GSMP (\text{trms}_{lst} A)$ "
  have "?Q t'" when "?Q t" " $t' \sqsubseteq t$ " for t t'
    using homogeneous_subterm[OF _ that(2)] GSMP_subterm[OF _ that(2)] that(1)
    by blast
  thus ?thesis
    using assms(1) restricted_deduct_if_restricted_ik[OF _ assms(2)]
    unfolding intruder_deduct_hom_def
    by blast
qed

lemma deduct_hom_if_synth:
  assumes hom: " $\text{homogeneous}_{lst} m A Sec$ " " $m \in GSMP (\text{trms}_{lst} A)$ "
  and m: " $M \vdash_c m$ "
  shows " $\langle M; A; Sec \rangle \vdash_{hom} m$ "
proof -
  let ?Q = " $\lambda m. \text{homogeneous}_{lst} m A Sec \wedge m \in GSMP (\text{trms}_{lst} A)$ "
  have "?Q t'" when "?Q t" " $t' \sqsubseteq t$ " for t t'
    using homogeneous_subterm[OF _ that(2)] GSMP_subterm[OF _ that(2)] that(1)
    by blast
  thus ?thesis
    using assms deduct_restricted_if_synth[of ?Q]
    unfolding intruder_deduct_hom_def
    by blast
qed

lemma hom_deduct_if_deduct:
  assumes A: " $\text{par\_comp } A Sec$ "
  and M: " $\forall m \in M. \text{homogeneous}_{lst} m A Sec \wedge m \in GSMP (\text{trms}_{lst} A)$ "
  and m: " $M \vdash m$ " " $m \in GSMP (\text{trms}_{lst} A)$ "
  shows " $\langle M; A; Sec \rangle \vdash_{hom} m$ "
proof -
  let ?P = " $\lambda x. \text{homogeneous}_{lst} x A Sec \wedge x \in GSMP (\text{trms}_{lst} A)$ "

  have GSMP_hom: " $\text{homogeneous}_{lst} t A Sec$ " when " $t \in GSMP (\text{trms}_{lst} A)$ " for t
    using A GSMP_is_homogeneous[of A Sec t]
    secrets_are_homogeneous[of Sec " $\lambda x. \text{True}$ " t A] that
    unfolding par_comp_def by blast

  have P_Ana: "?P k" when "?P t" " $\text{Ana } t = (K, T)$ " " $k \in \text{set } K$ " for t K T k
    using GSMP_Ana_key[OF _ that(2,3), of " $\text{trms}_{lst} A$ "] A that GSMP_hom
    by presburger

  have P_subterm: "?P t'" when "?P t" " $t' \sqsubseteq t$ " for t t'
    using GSMP_subterm[of _ " $\text{trms}_{lst} A$ "] homogeneous_subterm[of _ A Sec] that
    by blast

  have P_m: "?P m"
    using GSMP_hom[OF m(2)] m(2)
    by metis

  show ?thesis
    using restricted_deduct_if_deduct'[OF M _ m(1) P_m] P_Ana P_subterm
    unfolding intruder_deduct_hom_def
    by fast
qed

```

## 5.2.8 Lemmata: Deduction Reduction of Parallel Composable Constraints

```

lemma par_comp_hom_deduct:
  assumes A: "par_comp A Sec"
  and M: "∀l. ∀m ∈ M l. homogeneouslst m A Sec"
    "∀l. M l ⊆ GSMP (trms_projlst l A)"
    "∀l. Discl ⊆ M l"
    "Discl ⊆ Sec ∪ {m. {} ⊢c m}"
  and Sec: "∀l. ∀s ∈ Sec - Discl. ¬(⟨M l; A; Sec⟩ ⊢hom s)"
  and t: "⟨∪l. M l; A; Sec⟩ ⊢hom t"
  shows "t ∉ Sec - Discl" (is ?A)
    "∀l. t ∈ GSMP (trms_projlst l A) → ⟨M l; A; Sec⟩ ⊢hom t" (is ?B)
proof -
  have M': "∀l. ∀m ∈ M l. m ∈ GSMP (trmslst A)"
  proof (intro allI ballI)
    fix l m show "m ∈ M l ⇒ m ∈ GSMP (trmslst A)" using M(2) in_proj_in_GSMP[of m l A] by blast
  qed

  show ?A ?B using t
  proof (induction t rule: intruder_deduct_hom_induct)
    case (AxiomH t)
    then obtain lt where t_in_proj_ik: "t ∈ M lt" by moura
    show t_not_Sec: "t ∉ Sec - Discl"
    proof
      assume "t ∈ Sec - Discl"
      hence "∀l. ¬(⟨M l; A; Sec⟩ ⊢hom t)" using Sec by auto
      thus False using intruder_deduct_hom_AxiomH[OF t_in_proj_ik] by metis
    qed

    have 1: "∀l. t ∈ M l → t ∈ GSMP (trms_projlst l A)"
      using M(2,3) AxiomH by auto

    have 3: "∧l1 l2. l1 ≠ l2 ⇒ t ∈ GSMP (trms_projlst l1 A) ∩ GSMP (trms_projlst l2 A)
      ⇒ {} ⊢c t ∨ t ∈ Discl"
      using A t_not_Sec by (auto simp add: par_comp_def GSMP_disjoint_def)

    have 4: "homogeneouslst t A Sec" "t ∈ GSMP (trmslst A)" using M(1) M' t_in_proj_ik by auto

    { fix l assume "t ∈ Discl"
      hence "t ∈ M l" using M(3) by auto
      hence "⟨M l; A; Sec⟩ ⊢hom t" by auto
    } hence 5: "∀l. t ∈ Discl → ⟨M l; A; Sec⟩ ⊢hom t" by metis

    show "∀l. t ∈ GSMP (trms_projlst l A) → ⟨M l; A; Sec⟩ ⊢hom t"
      by (metis (lifting) Int_iff empty_subsetI
        1 3 4 5 t_in_proj_ik
        intruder_deduct_hom_AxiomH[of t _ A Sec]
        deduct_hom_if_synth[of t A Sec "{}"]
        ideduct_hom_mono[of "{}" A Sec t])

  next
    case (ComposeH T f)
    show "∀l. Fun f T ∈ GSMP (trms_projlst l A) → ⟨M l; A; Sec⟩ ⊢hom Fun f T"
    proof (intro allI impI)
      fix l
      assume "Fun f T ∈ GSMP (trms_projlst l A)"
      hence "∧t. t ∈ set T ⇒ t ∈ GSMP (trms_projlst l A)"
        using GSMP_subterm[OF _ subtermeqI''] by auto
      thus "⟨M l; A; Sec⟩ ⊢hom Fun f T"
        using ComposeH.IH(2) intruder_deduct_hom_ComposeH[OF ComposeH.hyps(1,2) _ ComposeH.hyps(4,5)]
        by simp
    qed
    thus "Fun f T ∉ Sec - Discl"
      using Sec ComposeH.hyps(5) trmslst_union[of A] GSMP_Union[of A]

```

```

    by (metis (no_types, lifting) UN_iff)
next
case (DecomposeH t K T ti)
have ti_subt: "ti ⊆ t" using Ana_subterm[OF DecomposeH.hyps(2)] ⟨ti ∈ set T⟩ by auto
have t: "homogeneouslst t A Sec" "t ∈ GSMP (trmslst A)"
  using DecomposeH.hyps(1) hom_deduct_if_hom_ik M(1) M'
  by auto
have ti: "homogeneouslst ti A Sec" "ti ∈ GSMP (trmslst A)"
  using intruder_deduct_hom_DecomposeH[OF DecomposeH.hyps] hom_deduct_if_hom_ik M(1) M' by auto
{ fix l assume *: "ti ∈ GSMP (trmsprojlst l A)" "t ∈ GSMP (trmsprojlst l A)"
  hence "∧k. k ∈ set K ⇒ ⟨M l; A; Sec⟩ ⊢hom k"
    using GSMP_Ana_key[OF _ DecomposeH.hyps(2)] DecomposeH.IH(4) by auto
  hence "⟨M l; A; Sec⟩ ⊢hom ti" "ti ∉ Sec - Discl"
    using Sec DecomposeH.IH(2) *(2)
    intruder_deduct_hom_DecomposeH[OF _ DecomposeH.hyps(2) _ ⟨ti ∈ set T⟩]
    by force+
} moreover {
fix l1 l2 assume *: "ti ∈ GSMP (trmsprojlst l1 A)" "t ∈ GSMP (trmsprojlst l2 A)" "l1 ≠ l2"
have "GSMP_disjoint (trmsprojlst l1 A) (trmsprojlst l2 A) Sec"
  using *(3) A by (simp add: par_comp_def)
hence "ti ∈ Sec ∪ {m. {} ⊢c m}"
  using GSMP_subterm[OF *(2) ti_subt] *(1) by (auto simp add: GSMP_disjoint_def)
moreover have "∧k. k ∈ set K ⇒ ⟨M l2; A; Sec⟩ ⊢hom k"
  using *(2) GSMP_Ana_key[OF _ DecomposeH.hyps(2)] DecomposeH.IH(4) by auto
ultimately have "ti ∉ Sec - Discl" "{ } ⊢c ti ∨ ti ∈ Discl"
  using Sec DecomposeH.IH(2) *(2)
  intruder_deduct_hom_DecomposeH[OF _ DecomposeH.hyps(2) _ ⟨ti ∈ set T⟩]
  by (metis (lifting), metis (no_types, lifting) DiffI Un_iff mem_Collect_eq)
hence "⟨M l1; A; Sec⟩ ⊢hom ti" "⟨M l2; A; Sec⟩ ⊢hom ti" "ti ∉ Sec - Discl"
  using M(3,4) deduct_hom_if_synth[THEN ideduct_hom_mono] ti
  by (meson intruder_deduct_hom_AxiomH empty_subsetI subsetCE)+
} moreover have
  "∃l. ti ∈ GSMP (trmsprojlst l A)"
  "∃l. t ∈ GSMP (trmsprojlst l A)"
  using in_GSMP_in_proj[of _ A] ti(2) t(2) by presburger+
ultimately show
  "ti ∉ Sec - Discl"
  "∀l. ti ∈ GSMP (trmsprojlst l A) ⇒ ⟨M l; A; Sec⟩ ⊢hom ti"
  by (metis (no_types, lifting))+
qed
qed

lemma par_comp_deduct_proj:
  assumes A: "par_comp A Sec"
  and M: "∀l. ∀m∈M l. homogeneouslst m A Sec"
  "∀l. M l ⊆ GSMP (trmsprojlst l A)"
  "∀l. Discl ⊆ M l"
  and t: "(⋃l. M l) ⊢ t" "t ∈ GSMP (trmsprojlst l A)"
  and Discl: "Discl ⊆ Sec ∪ {m. {} ⊢c m}"
  shows "M l ⊢ t ∨ (∃s ∈ Sec - Discl. ∃l. M l ⊢ s)"
using t
proof (induction t rule: intruder_deduct_induct)
case (Axiom t)
then obtain l' where t_in_ik_proj: "t ∈ M l'" by moura
show ?case
proof (cases "t ∈ Sec - Discl ∨ { } ⊢c t")
case True
note T = True
show ?thesis
proof (cases "t ∈ Sec - Discl")
case True thus ?thesis using intruder_deduct.Axiom[OF t_in_ik_proj] by metis
next
case False thus ?thesis using T ideduct_mono[of "{ }" t] by auto

```

```

qed
next
case False
hence "t ∉ Sec - Discl" "¬{ } ⊢c t" "t ∈ GSMP (trms_projlst 1 A)" using Axiom by auto
hence "(∀ l'. l ≠ l' → t ∉ GSMP (trms_projlst l' A)) ∨ t ∈ Discl"
  using A unfolding GSMP_disjoint_def par_comp_def by auto
hence "(∀ l'. l ≠ l' → t ∉ GSMP (trms_projlst l' A)) ∨ t ∈ M l ∨ { } ⊢c t" using M by auto
thus ?thesis using Axiom deduct_if_synth[THEN ideduct_mono] t_in_ik_proj
  by (metis (no_types, lifting) False M(2) intruder_deduct.Axiom subsetCE)
qed
next
case (Compose T f)
hence "Fun f T ∈ GSMP (trms_projlst 1 A)" using Compose.prem by auto
hence "∧ t. t ∈ set T ⇒ t ∈ GSMP (trms_projlst 1 A)" unfolding GSMP_def by auto
hence IH: "∧ t. t ∈ set T ⇒ M l ⊢ t ∨ (∃ s ∈ Sec - Discl. ∃ l. M l ⊢ s)"
  using Compose.IH by auto
show ?case
proof (cases "∀ t ∈ set T. M l ⊢ t")
  case True thus ?thesis by (metis intruder_deduct.Compose[OF Compose.hyps(1,2)])
qed (metis IH)
next
case (Decompose t K T ti)
have hom_ik: "∀ l. ∀ m ∈ M l. homogeneouslst m A Sec ∧ m ∈ GSMP (trmslst A)"
proof (intro allI ballI conjI)
  fix l m assume m: "m ∈ M l"
  thus "homogeneouslst m A Sec" using M(1) by simp
  show "m ∈ GSMP (trmslst A)" using in_proj_in_GSMP[of m l A] M(2) m by blast
qed

have par_comp_unfold:
  "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (trms_projlst l1 A) (trms_projlst l2 A) Sec"
  using A by (auto simp add: par_comp_def)

note ti_GSMP = in_proj_in_GSMP[OF Decompose.prem(1)]

have "(∪ l. M l; A; Sec) ⊢hom ti"
  using intruder_deduct.Decompose[OF Decompose.hyps]
  hom_deduct_if_deduct[OF A, of "∪ l. M l"] hom_ik ti_GSMP
  by blast
hence "(M l; A; Sec) ⊢hom ti ∨ (∃ s ∈ Sec - Discl. ∃ l. (M l; A; Sec) ⊢hom s)"
  using par_comp_hom_deduct(2)[OF A M Discl(1)] Decompose.prem(1)
  by blast
thus ?case using deduct_if_hom_deduct[of _ A Sec] by auto
qed

```

### 5.2.9 Theorem: Parallel Compositionality for Labeled Constraints

```

lemma par_comp_prefix: assumes "par_comp (A@B) M" shows "par_comp A M"
proof -
  let ?L = "λ l. trms_projlst l A ∪ trms_projlst l B"
  have "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (?L l1) (?L l2) M"
    using assms unfolding par_comp_def
    by (metis trmsst_append proj_append(2) unlabel_append)
  hence "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (trms_projlst l1 A) (trms_projlst l2 A) M"
    using SMP_union by (auto simp add: GSMP_def GSMP_disjoint_def)
  thus ?thesis using assms unfolding par_comp_def by blast
qed

theorem par_comp_constr_typed:
  assumes A: "par_comp A Sec"
  and I: "I ⊢ ⟨unlabel A⟩" "interpretationsubst I" "wtsubst I" "wftrms (subst_range I)"
  shows "(∀ l. (I ⊢ ⟨proj_unl l A⟩)) ∨ (∃ A'. prefix A' A ∧ (strand_leakslst A' Sec I))"
proof -

```

```

let ?L = "λA'. ∃ t ∈ Sec - declassifiedlst A' I. ∃ l. [{ }; proj_unl 1 A'@[Send t]]d I"
have "[{ }; unlabeled A]d I" using I by (simp add: constr_sem_d_def)
with A have "(∀ l. [{ }; proj_unl 1 A]d I) ∨ (∃ A'. prefix A' A ∧ ?L A)"
proof (induction "unlabel A" arbitrary: A rule: List.rev_induct)
  case Nil
  hence "A = []" using unlabel_nil_only_if_nil by simp
  thus ?case by auto
next
  case (snoc b B A)
  hence disj: "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (trms_projlst l1 A) (trms_projlst l2 A) Sec"
    by (auto simp add: par_comp_def)

  obtain a A n where a: "A = A@[a]" "a = (ln n, b) ∨ a = (*, b)"
    using unlabel_snoc_inv[OF snoc.hyps(2)[symmetric]] by moura
  hence A: "A = A@[(ln n, b)] ∨ A = A@[(*, b)]" by metis

  have 1: "B = unlabeled A" using a snoc.hyps(2) unlabel_append[of A "[a]"] by auto
  have 2: "par_comp A Sec" using par_comp_prefix snoc.prem(1) a by metis
  have 3: "[{ }; unlabeled A]d I" by (metis 1 snoc.prem(2) snoc.hyps(2) strand_sem_split(3))
  have IH: "(∀ l. [{ }; proj_unl 1 A]d I) ∨ (∃ A'. prefix A' A ∧ ?L A)"
    by (rule snoc.hyps(1)[OF 1 2 3])

  show ?case
  proof (cases "∀ l. [{ }; proj_unl 1 A]d I")
    case False
    then obtain A' where A': "prefix A' A" "?L A'" by (metis IH)
    hence "prefix A' (A@[a])" using a prefix_prefix[of _ A "[a]"] by simp
    thus ?thesis using A'(2) a by auto
  next
    case True
    note IH' = True
    show ?thesis
    proof (cases b)
      case (Send t)
      hence "ikst (unlabel A) ·set I ⊢ t · I"
        using a ([{ }; unlabeled A]d I) strand_sem_split(2)[of "{ }" "unlabel A" "unlabel [a]" I]
        unlabel_append[of A "[a]"]
        by auto
      hence *: "(⋃ l. (ikst (proj_unl 1 A) ·set I)) ⊢ t · I"
        using proj_ik_union_is_unlabel_ik image_UN by metis

      have "ikst (proj_unl 1 A) = ikst (proj_unl 1 A)" for l
        using Send A
        by (metis append_Nil2 ikst.sims(3) proj_unl_cons(3) proj_nil(2)
            singleton_lst_proj(1,2) proj_ik_append)
      hence **: "ikst (proj_unl 1 A) ·set I ⊆ GSMP (trms_projlst 1 A)" for l
        using ik_proj_subst_GSMP_subset[OF I(3,4,2), of _ A]
        by auto

      note Discl =
        declassified_proj_ik_subset[of A I]
        declassified_proj_GSMP_subset[OF I(3,4,2), of A]
        declassified_secrets_subset[OF disj I(3,4,2)]
        declassified_append[of A "[a]" I]

      have Sec: "ground Sec"
        using A by (auto simp add: par_comp_def)

      have "∀ m ∈ ikst (proj_unl 1 A) ·set I. homogeneouslst m A Sec ∨ m ∈ Sec-declassifiedlst A I"
        "∀ m ∈ ikst (proj_unl 1 A) ·set I. m ∈ GSMP (trmslst A)"
        "ikst (proj_unl 1 A) ·set I ⊆ GSMP (trms_projlst 1 A)"
        for l
        using declassified_secrets_are_homogeneous[OF disj I(3,4,2)]

```

```

    GSMP_proj_is_homogeneous[OF disj]
    ik_proj_subst_GSMP_subset[OF  $\mathcal{I}(3,4,2)$ , of _ A]
  apply (metis (no_types, lifting) Diff_iff Discl(4) UnCI a(1) subsetCE)
  using ik_proj_subst_GSMP_subset[OF  $\mathcal{I}(3,4,2)$ , of _ A]
    GSMP_Union[of A]
  by auto
moreover have "ikst (proj_unl 1 [a]) = {}" for 1
  using Send proj_ikst_is_proj_rcv_set[of _ "[a]" a(2) by auto
ultimately have M:
  " $\forall 1. \forall m \in ik_{st} \text{ (proj\_unl 1 A) } \cdot_{set} \mathcal{I}. \text{homogeneous}_{lst} m \mathcal{A} \text{ Sec} \vee m \in \text{Sec-declassified}_{lst} A$ "
 $\mathcal{I}$ "

  " $\forall 1. ik_{st} \text{ (proj\_unl 1 A) } \cdot_{set} \mathcal{I} \subseteq \text{GSMP (trms\_proj}_{lst} 1 A)$ "
  using a(1) proj_ik_append[of _ A "[a]" by auto

have prefix_A: "prefix A A" using A by auto

have "s ·  $\mathcal{I}$  = s"
  when "s ∈ Sec" for s
  using that Sec by auto
hence leakage_case: " $\llbracket \{\}; \text{proj\_unl 1 A} @ [\text{Send s}] \rrbracket_d \mathcal{I}$ "
  when "s ∈ Sec - declassifiedlst A  $\mathcal{I}$ " "ikst (proj_unl 1 A) ·set  $\mathcal{I} \vdash s$ " for 1 s
  using that strand_sem_append(2) IH' by auto

have proj_deduct_case_n:
  " $\forall m. m \neq n \longrightarrow \llbracket \{\}; \text{proj\_unl } m \text{ (A@[a])} \rrbracket_d \mathcal{I}$ "
  " $ik_{st} \text{ (proj\_unl } n \text{ A) } \cdot_{set} \mathcal{I} \vdash t \cdot \mathcal{I} \implies \llbracket \{\}; \text{proj\_unl } n \text{ (A@[a])} \rrbracket_d \mathcal{I}$ "
  when "a = (ln n, Send t)"
  using that IH' proj_append(2)[of _ A]
  by auto

have proj_deduct_case_star:
  " $\llbracket \{\}; \text{proj\_unl 1 (A@[a])} \rrbracket_d \mathcal{I}$ "
  when "a = (★, Send t)" "ikst (proj_unl 1 A) ·set  $\mathcal{I} \vdash t \cdot \mathcal{I}$ " for 1
  using that IH' proj_append(2)[of _ A]
  by auto

show ?thesis
proof (cases "∃ 1. ∃ m ∈ ikst (proj_unl 1 A) ·set  $\mathcal{I}. m \in \text{Sec - declassified}_{lst} A \mathcal{I}$ ")
  case True
  then obtain 1 s where ls: "s ∈ Sec - declassifiedlst A  $\mathcal{I}$ " "ikst (proj_unl 1 A) ·set  $\mathcal{I} \vdash s$ "
    using intruder_deduct.Axiom by metis
  thus ?thesis using leakage_case prefix_A by blast
next
  case False
  hence M': " $\forall 1. \forall m \in ik_{st} \text{ (proj\_unl 1 A) } \cdot_{set} \mathcal{I}. \text{homogeneous}_{lst} m \mathcal{A} \text{ Sec}$ " using M(1) by blast

  note deduct_proj_lemma =
    par_comp_deduct_proj[OF snoc.prem(1) M' M(2) _ *, of "declassifiedlst A  $\mathcal{I}$ " n]

  from a(2) show ?thesis
  proof
    assume "a = (ln n, b)"
    hence "a = (ln n, Send t)" "t ·  $\mathcal{I} \in \text{GSMP (trms\_proj}_{lst} n A)$ "
      using Send a(1) trms_projlst_append[of n A "[a]"
        GSMP_wt_substI[OF _  $\mathcal{I}(3,4,2)$ ]
      by (metis, force)
    hence
      "a = (ln n, Send t)"
      " $\forall m. m \neq n \longrightarrow \llbracket \{\}; \text{proj\_unl } m \text{ (A@[a])} \rrbracket_d \mathcal{I}$ "
      " $ik_{st} \text{ (proj\_unl } n \text{ A) } \cdot_{set} \mathcal{I} \vdash t \cdot \mathcal{I} \implies \llbracket \{\}; \text{proj\_unl } n \text{ (A@[a])} \rrbracket_d \mathcal{I}$ "
      "t ·  $\mathcal{I} \in \text{GSMP (trms\_proj}_{lst} n A)$ "
      using proj_deduct_case_n
      by auto
  end

```

```

hence "( $\forall l. \llbracket \{ \}; \text{proj\_unl } l \ A \rrbracket_d \mathcal{I}$ )  $\vee$ 
  ( $\exists s \in \text{Sec-declassified}_{lst} A \ \mathcal{I}. \exists l. ik_{st} (\text{proj\_unl } l \ A) \cdot_{set} \mathcal{I} \vdash s$ )"
  using deduct_proj_lemma A a Discl
  by fast
thus ?thesis using leakage_case prefix_A by metis
next
assume "a = (*, b)"
hence ***: "a = (*, Send t)" "t  $\cdot \mathcal{I} \in \text{GSMP}(\text{trms\_proj}_{lst} l \ A)"$  for l
  using Send a(1) GSMP_wt_substI[OF  $\_ \mathcal{I}(3,4,2)$ ]
  by (metis, force)
hence "t  $\cdot \mathcal{I} \in \text{Sec-declassified}_{lst} A \ \mathcal{I} \vee$ 
  t  $\cdot \mathcal{I} \in \text{declassified}_{lst} A \ \mathcal{I} \vee$ 
  t  $\cdot \mathcal{I} \in \{m. \{ \} \vdash_c m\}"$ 
  using snoc.premis(1) a(1) at_least_2_labels
  unfolding par_comp_def GSMP_disjoint_def
  by blast
thus ?thesis
proof (elim disjE)
  assume "t  $\cdot \mathcal{I} \in \text{Sec-declassified}_{lst} A \ \mathcal{I}"$ 
  hence " $\exists s \in \text{Sec-declassified}_{lst} A \ \mathcal{I}. \exists l. ik_{st} (\text{proj\_unl } l \ A) \cdot_{set} \mathcal{I} \vdash s$ "
    using deduct_proj_lemma ***(2) A a Discl
    by blast
  thus ?thesis using prefix_A leakage_case by blast
next
  assume "t  $\cdot \mathcal{I} \in \text{declassified}_{lst} A \ \mathcal{I}"$ 
  hence " $ik_{st} (\text{proj\_unl } l \ A) \cdot_{set} \mathcal{I} \vdash t \cdot \mathcal{I}$ " for l
    using intruder_deduct.Axiom Discl(1) by blast
  thus ?thesis using proj_deduct_case_star[OF ***(1)] a(1) by fast
next
  assume "t  $\cdot \mathcal{I} \in \{m. \{ \} \vdash_c m\}"$ 
  hence " $M \vdash t \cdot \mathcal{I}$ " for M using ideduct_mono[OF deduct_if_synth] by blast
  thus ?thesis using IH' a(1) ***(1) by fastforce
qed
qed
qed
next
case (Receive t)
hence " $\llbracket \{ \}; \text{proj\_unl } l \ A \rrbracket_d \mathcal{I}$ " for l
  using IH' a proj_append(2)[of l A "[a]"]
  unfolding unlabel_def proj_def by auto
thus ?thesis by metis
next
case (Equality ac t t')
hence *: " $\llbracket M; [\text{Equality ac t t'}] \rrbracket_d \mathcal{I}$ " for M
  using a  $\langle \{ \}; \text{unlabel } A \rrbracket_d \mathcal{I} \rangle$  unlabel_append[of A "[a]"]
  by auto
show ?thesis
  using a proj_append(2)[of  $\_ A$  "[a]"] Equality
    strand_sem_append(2)[OF  $\_ *$ ] IH'
  unfolding unlabel_def proj_def by auto
next
case (Inequality X F)
hence *: " $\llbracket M; [\text{Inequality X F}] \rrbracket_d \mathcal{I}$ " for M
  using a  $\langle \{ \}; \text{unlabel } A \rrbracket_d \mathcal{I} \rangle$  unlabel_append[of A "[a]"]
  by auto
show ?thesis
  using a proj_append(2)[of  $\_ A$  "[a]"] Inequality
    strand_sem_append(2)[OF  $\_ *$ ] IH'
  unfolding unlabel_def proj_def by auto
qed
qed
qed
thus ?thesis using  $\mathcal{I}(1)$  unfolding strand_leakslst_def by (simp add: constr_sem_d_def)

```



qed

theorem par\_comp\_constr:

assumes  $\mathcal{A}$ : "par\_comp  $\mathcal{A}$  Sec" "typing\_cond (unlabel  $\mathcal{A}$ )"and  $\mathcal{I}$ : " $\mathcal{I} \models \langle \text{unlabel } \mathcal{A} \rangle$ " "interpretation<sub>subst</sub>  $\mathcal{I}$ "shows " $\exists \mathcal{I}_\tau. \text{interpretation}_{\text{subst}} \mathcal{I}_\tau \wedge \text{wt}_{\text{subst}} \mathcal{I}_\tau \wedge \text{wf}_{\text{trms}} (\text{subst\_range } \mathcal{I}_\tau) \wedge (\mathcal{I}_\tau \models \langle \text{unlabel } \mathcal{A} \rangle) \wedge ((\forall l. (\mathcal{I}_\tau \models \langle \text{proj\_unl } l \mathcal{A} \rangle)) \vee (\exists \mathcal{A}'. \text{prefix } \mathcal{A}' \mathcal{A} \wedge (\text{strand\_leaks}_{\text{lst}} \mathcal{A}' \text{ Sec } \mathcal{I}_\tau)))$ "

proof -

from  $\mathcal{A}(2)$  have \*:"wf<sub>st</sub> {} (unlabel  $\mathcal{A}$ )""fv<sub>st</sub> (unlabel  $\mathcal{A}$ )  $\cap$  bvars<sub>st</sub> (unlabel  $\mathcal{A}$ ) = {}""tfr<sub>st</sub> (unlabel  $\mathcal{A}$ )""wf<sub>trms</sub> (trms<sub>st</sub> (unlabel  $\mathcal{A}$ ))""Ana\_invar\_subst (ik<sub>st</sub> (unlabel  $\mathcal{A}$ )  $\cup$  assignment\_rhs<sub>st</sub> (unlabel  $\mathcal{A}$ ))"unfolding typing\_cond\_def tfr<sub>st</sub>\_def by metis+obtain  $\mathcal{I}_\tau$  where  $\mathcal{I}_\tau$ : " $\mathcal{I}_\tau \models \langle \text{unlabel } \mathcal{A} \rangle$ " "interpretation<sub>subst</sub>  $\mathcal{I}_\tau$ " "wt<sub>subst</sub>  $\mathcal{I}_\tau$ " "wf<sub>trms</sub> (subst\_range  $\mathcal{I}_\tau$ )"using wt\_attack\_if\_tfr\_attack\_d[OF \*  $\mathcal{I}(2,1)$ ] by metisshow ?thesis using par\_comp\_constr\_typed[OF  $\mathcal{A}(1)$   $\mathcal{I}_\tau$ ]  $\mathcal{I}_\tau$  by auto

qed

### 5.2.10 Theorem: Parallel Compositionality for Labeled Protocols

#### Definitions: Labeled Protocols

We state our result on the level of protocol traces (i.e., the constraints reachable in a symbolic execution of the actual protocol). Hence, we do not need to convert protocol strands to intruder constraints in the following well-formedness definitions.

definition wf<sub>lsts</sub> :: "('fun, 'var, 'lbl) labeled\_strand set  $\Rightarrow$  bool" where"wf<sub>lsts</sub>  $\mathcal{S} \equiv (\forall \mathcal{A} \in \mathcal{S}. \text{wf}_{\text{lst}} \{\} \mathcal{A}) \wedge (\forall \mathcal{A} \in \mathcal{S}. \forall \mathcal{A}' \in \mathcal{S}. \text{fv}_{\text{lst}} \mathcal{A} \cap \text{bvars}_{\text{lst}} \mathcal{A}' = \{\})$ "definition wf<sub>lsts'</sub> :: "('fun, 'var, 'lbl) labeled\_strand set  $\Rightarrow$  ('fun, 'var, 'lbl) labeled\_strand  $\Rightarrow$  bool" where
$$\begin{aligned} \text{"wf}_{\text{lsts}'} \mathcal{S} \mathcal{A} \equiv & (\forall \mathcal{A}' \in \mathcal{S}. \text{wf}_{\text{st}} (\text{wf}_{\text{restrictedvars}_{\text{lst}}} \mathcal{A}) (\text{unlabel } \mathcal{A}')) \wedge \\ & (\forall \mathcal{A}' \in \mathcal{S}. \forall \mathcal{A}'' \in \mathcal{S}. \text{fv}_{\text{lst}} \mathcal{A}' \cap \text{bvars}_{\text{lst}} \mathcal{A}'' = \{\}) \wedge \\ & (\forall \mathcal{A}' \in \mathcal{S}. \text{fv}_{\text{lst}} \mathcal{A}' \cap \text{bvars}_{\text{lst}} \mathcal{A} = \{\}) \wedge \\ & (\forall \mathcal{A}' \in \mathcal{S}. \text{fv}_{\text{lst}} \mathcal{A} \cap \text{bvars}_{\text{lst}} \mathcal{A}' = \{\}) \end{aligned}$$

definition typing\_cond\_prot where

"typing\_cond\_prot  $\mathcal{P} \equiv$ wf<sub>lsts</sub>  $\mathcal{P} \wedge$ tfr<sub>set</sub> ( $\bigcup (\text{trms}_{\text{lst}} \text{ ' } \mathcal{P})) \wedge$ wf<sub>trms</sub> ( $\bigcup (\text{trms}_{\text{lst}} \text{ ' } \mathcal{P})) \wedge$ ( $\forall \mathcal{A} \in \mathcal{P}. \text{list\_all tfr}_{\text{stp}} (\text{unlabel } \mathcal{A})$ )  $\wedge$ Ana\_invar\_subst ( $\bigcup (\text{ik}_{\text{st}} \text{ ' } \text{unlabel ' } \mathcal{P}) \cup \bigcup (\text{assignment\_rhs}_{\text{st}} \text{ ' } \text{unlabel ' } \mathcal{P}))$ "

definition par\_comp\_prot where

"par\_comp\_prot  $\mathcal{P}$  Sec  $\equiv$ ( $\forall l1 \ l2. l1 \neq l2 \longrightarrow$ GSMP\_disjoint ( $\bigcup \mathcal{A} \in \mathcal{P}. \text{trms\_proj}_{\text{lst}} l1 \mathcal{A}$ ) ( $\bigcup \mathcal{A} \in \mathcal{P}. \text{trms\_proj}_{\text{lst}} l2 \mathcal{A}$ ) Sec)  $\wedge$ ground Sec  $\wedge (\forall s \in \text{Sec}. \forall s' \in \text{subterms } s. \{\} \vdash_c s' \vee s' \in \text{Sec}) \wedge$ typing\_cond\_prot  $\mathcal{P}$ "

#### Lemmata: Labeled Protocols

lemma wf<sub>lsts</sub>\_eqs\_wf<sub>lsts'</sub> [simp]: "wf<sub>lsts</sub>  $\mathcal{S} = \text{wf}_{\text{lsts}'} \mathcal{S} []$ "unfolding wf<sub>lsts</sub>\_def wf<sub>lsts'</sub>\_def unlabel\_def by auto

lemma par\_comp\_prot\_impl\_par\_comp:

assumes "par\_comp\_prot  $\mathcal{P}$  Sec" " $\mathcal{A} \in \mathcal{P}$ "shows "par\_comp  $\mathcal{A}$  Sec"

proof -

```

have *: "∀ l1 l2. l1 ≠ l2 →
  GSMP_disjoint (⋃ A ∈ P. trms_proj_lst l1 A) (⋃ A ∈ P. trms_proj_lst l2 A) Sec"
  using assms(1) unfolding par_comp_prot_def by metis
{ fix l1 l2::'lbl assume **: "l1 ≠ l2"
  hence ***: "GSMP_disjoint (⋃ A ∈ P. trms_proj_lst l1 A) (⋃ A ∈ P. trms_proj_lst l2 A) Sec"
    using * by auto
  have "GSMP_disjoint (trms_proj_lst l1 A) (trms_proj_lst l2 A) Sec"
    using GSMP_disjoint_subset[OF ***] assms(2) by auto
} hence "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (trms_proj_lst l1 A) (trms_proj_lst l2 A) Sec" by metis
thus ?thesis using assms unfolding par_comp_prot_def par_comp_def by metis
qed

```

lemma typing\_cond\_prot\_impl\_typing\_cond:

```

assumes "typing_cond_prot P" "A ∈ P"
shows "typing_cond (unlabel A)"

```

proof -

```

have 1: "wf_st {} (unlabel A)" "fv_lst A ∩ bvars_lst A = {}"
  using assms unfolding typing_cond_prot_def wf_lsts_def by auto

```

```

have "tfr_set (⋃ (trms_lst ' P))"
  "wf_trms (⋃ (trms_lst ' P))"
  "trms_lst A ⊆ ⋃ (trms_lst ' P)"
  "SMP (trms_lst A) - Var'V ⊆ SMP (⋃ (trms_lst ' P)) - Var'V"
  using assms SMP_mono[of "trms_lst A" "⋃ (trms_lst ' P)"]
  unfolding typing_cond_prot_def
  by (metis, metis, auto)
hence 2: "tfr_set (trms_lst A)" and 3: "wf_trms (trms_lst A)"
  unfolding tfr_set_def by (meson subsetD)+

```

```

have 4: "list_all tfr_stp (unlabel A)" using assms unfolding typing_cond_prot_def by auto

```

```

have "subterms_set (ik_st (unlabel A) ∪ assignment_rhs_st (unlabel A)) ⊆
  subterms_set (⋃ (ik_st ' unlabel ' P) ∪ ⋃ (assignment_rhs_st ' unlabel ' P))"
  using assms(2) by auto
hence 5: "Ana_invar_subst (ik_st (unlabel A) ∪ assignment_rhs_st (unlabel A))"
  using assms SMP_mono unfolding typing_cond_prot_def Ana_invar_subst_def by (meson subsetD)

```

```

show ?thesis using 1 2 3 4 5 unfolding typing_cond_def tfr_st_def by blast
qed

```

### Theorem: Parallel Compositionality for Labeled Protocols

definition component\_prot where

```

"component_prot n P ≡ (∀ l ∈ P. ∀ s ∈ set l. is_LabelN n s ∨ is_LabelS s)"

```

definition composed\_prot where

```

"composed_prot P_i ≡ {A. ∀ n. proj n A ∈ P_i n}"

```

definition component\_secure\_prot where

```

"component_secure_prot n P Sec attack ≡ (∀ A ∈ P. suffix [(ln n, Send (Fun attack []))] A →
  (∀ I_τ. (interpretation_subst I_τ ∧ wt_subst I_τ ∧ wf_trms (subst_range I_τ)) →
    ¬(I_τ ⊨ ⟨proj_unl n A⟩) ∧
    (∀ A'. prefix A' A →
      (∀ t ∈ Sec-declassified_lst A' I_τ. ¬(I_τ ⊨ ⟨proj_unl n A'@[Send t]⟩))))))"

```

definition component\_leaks where

```

"component_leaks n A Sec ≡ (∃ A' I_τ. interpretation_subst I_τ ∧ wt_subst I_τ ∧ wf_trms (subst_range I_τ)
  ∧
  prefix A' A ∧ (∃ t ∈ Sec - declassified_lst A' I_τ. (I_τ ⊨ ⟨proj_unl n A'@[Send t]⟩)))"

```

definition unsat where

```

"unsat A ≡ (∀ I. interpretation_subst I → ¬(I ⊨ ⟨unlabel A⟩))"

```

```

theorem par_comp_constr_prot:
  assumes P: "P = composed_prot Pi" "par_comp_prot P Sec" "∀n. component_prot n (Pi n)"
  and left_secure: "component_secure_prot n (Pi n) Sec attack"
  shows "∀A ∈ P. suffix [(ln n, Send (Fun attack []))] A →
    unsat A ∨ (∃m. n ≠ m ∧ component_leaks m A Sec)"
proof -
  { fix A A' assume A: "A = A'@[ln n, Send (Fun attack [])]" "A ∈ P"
    let ?P = "∃A'. Iτ. interpretationsubst Iτ ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ) ∧ prefix A' A
  }
  ∧
  (∃t ∈ Sec - declassifiedlst A' Iτ. ∃m. n ≠ m ∧ (Iτ ⊨ ⟨proj_unl m A'@[Send t]⟩))"
  have tcp: "typing_cond_prot P" using P(2) unfolding par_comp_prot_def by simp
  have par_comp: "par_comp A Sec" "typing_cond (unlabel A)"
    using par_comp_prot_impl_par_comp[OF P(2) A(2)]
    typing_cond_prot_impl_typing_cond[OF tcp A(2)]
    by metis+
  have "unlabel (proj n A) = proj_unl n A" "proj_unl n A = proj_unl n (proj n A)"
    "∧A. A ∈ Pi n ⇒ proj n A = A"
    "proj n A = (proj n A')@[ln n, Send (Fun attack [])]"
    using P(1,3) A by (auto simp add: proj_def unlabel_def component_prot_def composed_prot_def)
  moreover have "proj n A ∈ Pi n"
    using P(1) A unfolding composed_prot_def by blast
  moreover {
    fix A assume "prefix A A"
    hence *: "prefix (proj n A) (proj n A)" unfolding proj_def prefix_def by force
    hence "proj_unl n A = proj_unl n (proj n A)"
      "∀I. declassifiedlst A I = declassifiedlst (proj n A) I"
      unfolding proj_def declassifiedlst_def by auto
    hence "∃B. prefix B (proj n A) ∧ proj_unl n A = proj_unl n B ∧
      (∀I. declassifiedlst A I = declassifiedlst B I)"
      using * by metis
  }
  ultimately have *:
    "∀Iτ. interpretationsubst Iτ ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ) →
      ¬(Iτ ⊨ ⟨proj_unl n A⟩) ∧ (∀A'. prefix A' A →
        (∀t ∈ Sec - declassifiedlst A' Iτ. ¬(Iτ ⊨ ⟨proj_unl n A'@[Send t]⟩)))"
    using left_secure unfolding component_secure_prot_def composed_prot_def suffix_def by metis
  { fix I assume I: "interpretationsubst I" "I ⊨ ⟨unlabel A⟩"
    obtain Iτ where Iτ:
      "interpretationsubst Iτ" "wtsubst Iτ" "wftrms (subst_range Iτ)"
      "∃A'. prefix A' A ∧ (strand_leakslst A' Sec Iτ)"
      using par_comp_constr[OF par_comp I(2,1)] * by mouna
    hence "∃A'. prefix A' A ∧ (∃t ∈ Sec - declassifiedlst A' Iτ. ∃m.
      n ≠ m ∧ (Iτ ⊨ ⟨proj_unl m A'@[Send t]⟩))"
      using Iτ(4) * unfolding strand_leakslst_def by metis
    hence ?P using Iτ(1,2,3) by auto
  } hence "unsat A ∨ (∃m. n ≠ m ∧ component_leaks m A Sec)"
    by (metis unsat_def component_leaks_def)
  } thus ?thesis unfolding suffix_def by metis
qed
end

```

### 5.2.11 Automated GSMP Disjointness

```

locale labeled_typed_model' = typed_model' arity public Ana Γ +
  labeled_typed_model arity public Ana Γ label_witness1 label_witness2
  for arity::''fun ⇒ nat"
  and public::''fun ⇒ bool"
  and Ana::('fun, (('fun, 'atom::finite) term_type × nat)) term

```

```

    ⇒ (('fun, (('fun, 'atom) term_type × nat)) term list
      × (('fun, (('fun, 'atom) term_type × nat)) term list)"
  and Γ::"('fun, (('fun, 'atom) term_type × nat)) term ⇒ ('fun, 'atom) term_type"
  and label_witness1 label_witness2::'lbl
begin

lemma GSMP_disjointI:
  fixes A' A B B'::"('fun, ('fun, 'atom) term × nat) term list"
  defines "f ≡ λM. {t · δ | t δ. t ∈ M ∧ wtsubst δ ∧ wftrms (subst_range δ) ∧ fv (t · δ) = {}}"
    and "δ ≡ var_rename (max_var_set (fvset (set A)))"
  assumes A'_wf: "list_all (wftrm' arity) A'"
    and B'_wf: "list_all (wftrm' arity) B'"
    and A_inst: "has_all_wt_instances_of Γ (set A') (set A)"
    and B_inst: "has_all_wt_instances_of Γ (set B') (set (B ·list δ))"
    and A_SMP_repr: "finite_SMP_representation arity Ana Γ A"
    and B_SMP_repr: "finite_SMP_representation arity Ana Γ (B ·list δ)"
    and AB_trms_disj:
      "∀t ∈ set A. ∀s ∈ set (B ·list δ). Γ t = Γ s ∧ mgu t s ≠ None ⟶
        (intruder_synth' public arity {} t ∧ intruder_synth' public arity {} s) ∨
        ((∃u ∈ Sec. is_wt_instance_of_cond Γ t u) ∧ (∃u ∈ Sec. is_wt_instance_of_cond Γ s u))"
    and Sec_wf: "wftrms Sec"
  shows "GSMP_disjoint (set A') (set B') ((f Sec) - {m. {} ⊢c m})"
proof -
  have A_wf: "wftrms (set A)" and B_wf: "wftrms (set (B ·list δ))"
    and A'_wf': "wftrms (set A')" and B'_wf': "wftrms (set B')"
  using finite_SMP_representationD[OF A_SMP_repr]
    finite_SMP_representationD[OF B_SMP_repr]
    A'_wf B'_wf
  unfolding wftrms_code[symmetric] wftrm_code[symmetric] list_all_iff by blast+

  have AB_fv_disj: "fvset (set A) ∩ fvset (set (B ·list δ)) = {}"
    using var_rename_fv_set_disjoint'[of "set A" "set B", unfolded δ_def[symmetric]] by simp

  have "GSMP_disjoint (set A) (set (B ·list δ)) ((f Sec) - {m. {} ⊢c m})"
    using ground_SMP_disjointI[OF AB_fv_disj A_SMP_repr B_SMP_repr Sec_wf AB_trms_disj]
    unfolding GSMP_def GSMP_disjoint_def f_def by blast
  moreover have "SMP (set A') ⊆ SMP (set A)" "SMP (set B') ⊆ SMP (set (B ·list δ))"
    using SMP_I'[OF A'_wf' A_wf A_inst] SMP_SMP_subset[of "set A'" "set A"]
      SMP_I'[OF B'_wf' B_wf B_inst] SMP_SMP_subset[of "set B'" "set (B ·list δ)"]
    by blast+
  ultimately show ?thesis unfolding GSMP_def GSMP_disjoint_def by auto
qed

end

end

```

## 6 The Stateful Protocol Composition Result

In this chapter, we extend the compositionality result to stateful security protocols. This work is an extension of the work described in [4] and [1, chapter 5].

### 6.1 Labeled Stateful Strands (Labeled\_Stateful\_Strands)

```
theory Labeled_Stateful_Strands
imports Stateful_Strands Labeled_Strands
begin
```

#### 6.1.1 Definitions

Syntax for stateful strand labels

```
abbreviation Star_step ("⟨*, _⟩") where
  "⟨*, (s::('a,'b) stateful_strand_step)⟩ ≡ (*, s)"
```

```
abbreviation LabelN_step ("⟨_, _⟩") where
  "⟨(l::'a), (s::('b,'c) stateful_strand_step)⟩ ≡ (ln l, s)"
```

Database projection

```
abbreviation dbproj where "dbproj l D ≡ filter (λd. fst d = l) D"
```

The type of labeled stateful strands

```
type_synonym ('a,'b,'c) labeled_stateful_strand_step = "'c strand_label × ('a,'b)
stateful_strand_step"
type_synonym ('a,'b,'c) labeled_stateful_strand = "('a,'b,'c) labeled_stateful_strand_step list"
```

Dual strands

```
fun dual_lstp:: "('a,'b,'c) labeled_stateful_strand_step ⇒ ('a,'b,'c) labeled_stateful_strand_step"
where
  "dual_lstp (l,send⟨t⟩) = (l,receive⟨t⟩)"
| "dual_lstp (l,receive⟨t⟩) = (l,send⟨t⟩)"
| "dual_lstp x = x"
```

```
definition dual_lst:: "('a,'b,'c) labeled_stateful_strand ⇒ ('a,'b,'c) labeled_stateful_strand"
where
```

```
"dual_lst ≡ map dual_lstp"
```

Substitution application

```
fun subst_apply_labeled_stateful_strand_step::
  "('a,'b,'c) labeled_stateful_strand_step ⇒ ('a,'b) subst ⇒
  ('a,'b,'c) labeled_stateful_strand_step"
(infix "·lstp" 51) where
  "(l,s) ·lstp v = (l,s ·sstp v)"
```

```
definition subst_apply_labeled_stateful_strand::
  "('a,'b,'c) labeled_stateful_strand ⇒ ('a,'b) subst ⇒ ('a,'b,'c) labeled_stateful_strand"
(infix "·lst" 51) where
  "S ·lst v ≡ map (λx. x ·lstp v) S"
```

Definitions lifted from stateful strands

```
abbreviation wfrestrictedvars_lst where "wfrestrictedvars_lst S ≡ wfrestrictedvars_sst (unlabel S)"
```

```
abbreviation ik_lst where "ik_lst S ≡ ik_sst (unlabel S)"
```

abbreviation  $db_{lsst}$  where " $db_{lsst} S \equiv db_{sst} (\text{unlabel } S)$ "

abbreviation  $db'_{lsst}$  where " $db'_{lsst} S \equiv db'_{sst} (\text{unlabel } S)$ "

abbreviation  $trms_{lsst}$  where " $trms_{lsst} S \equiv trms_{sst} (\text{unlabel } S)$ "

abbreviation  $trms\_proj_{lsst}$  where " $trms\_proj_{lsst} n S \equiv trms_{sst} (\text{proj\_unl } n S)$ "

abbreviation  $vars_{lsst}$  where " $vars_{lsst} S \equiv vars_{sst} (\text{unlabel } S)$ "

abbreviation  $vars\_proj_{lsst}$  where " $vars\_proj_{lsst} n S \equiv vars_{sst} (\text{proj\_unl } n S)$ "

abbreviation  $bvars_{lsst}$  where " $bvars_{lsst} S \equiv bvars_{sst} (\text{unlabel } S)$ "

abbreviation  $fv_{lsst}$  where " $fv_{lsst} S \equiv fv_{sst} (\text{unlabel } S)$ "

Labeled set-operations

fun  $setops_{lsstp}$  where

" $setops_{lsstp} (i, \text{insert}(t, s)) = \{(i, t, s)\}$ "  
 / " $setops_{lsstp} (i, \text{delete}(t, s)) = \{(i, t, s)\}$ "  
 / " $setops_{lsstp} (i, \langle \_ : t \in s \rangle) = \{(i, t, s)\}$ "  
 / " $setops_{lsstp} (i, \forall \_ \langle \_ \neq \_ \vee \_ \notin F \rangle) = ((\lambda(t, s). (i, t, s)) \text{ ' set } F)$ "  
 / " $setops_{lsstp} \_ = \{\}$ "

definition  $setops_{lsst}$  where

" $setops_{lsst} S \equiv \bigcup (setops_{lsstp} \text{ ' set } S)$ "

### 6.1.2 Minor Lemmata

lemma  $subst\_lsst\_nil[simp]$ : " $[] \cdot_{lsst} \delta = []$ "

by (simp add: subst\_apply\_labeled\_stateful\_strand\_def)

lemma  $subst\_lsst\_cons$ : " $a \# A \cdot_{lsst} \delta = (a \cdot_{lsstp} \delta) \# (A \cdot_{lsst} \delta)$ "

by (simp add: subst\_apply\_labeled\_stateful\_strand\_def)

lemma  $subst\_lsst\_singleton$ : " $[(l, s)] \cdot_{lsst} \delta = [(l, s \cdot_{lsstp} \delta)]$ "

by (simp add: subst\_apply\_labeled\_stateful\_strand\_def)

lemma  $subst\_lsst\_append$ : " $A @ B \cdot_{lsst} \delta = (A \cdot_{lsst} \delta) @ (B \cdot_{lsst} \delta)$ "

by (simp add: subst\_apply\_labeled\_stateful\_strand\_def)

lemma  $subst\_lsst\_append\_inv$ :

assumes " $A \cdot_{lsst} \delta = B1 @ B2$ "

shows " $\exists A1 A2. A = A1 @ A2 \wedge A1 \cdot_{lsst} \delta = B1 \wedge A2 \cdot_{lsst} \delta = B2$ "

using *assms*

proof (induction A arbitrary: B1 B2)

case (Cons a A)

note *prems* = Cons.*prems*

note *IH* = Cons.*IH*

show ?case

proof (cases B1)

case Nil

then obtain b B3 where " $B2 = b \# B3$ " " $a \cdot_{lsstp} \delta = b$ " " $A \cdot_{lsst} \delta = B3$ "

using *prems*  $subst\_lsst\_cons$  by *fastforce*

thus ?thesis by (simp add: Nil subst\_apply\_labeled\_stateful\_strand\_def)

next

case (Cons b B3)

hence " $a \cdot_{lsstp} \delta = b$ " " $A \cdot_{lsst} \delta = B3 @ B2$ "

using *prems* by (simp\_all add:  $subst\_lsst\_cons$ )

thus ?thesis by (metis Cons\_eq\_appendI Cons *IH*  $subst\_lsst\_cons$ )

qed

qed (metis  $append\_is\_Nil\_conv$   $subst\_lsst\_nil$ )

lemma  $subst\_lsst\_member[intro]$ : " $x \in \text{set } A \implies x \cdot_{lsstp} \delta \in \text{set } (A \cdot_{lsst} \delta)$ "

by (metis  $image\_eqI$   $set\_map$   $subst\_apply\_labeled\_stateful\_strand\_def$ )

```

lemma subst_lsst_unlabel_cons: "unlabel ((l,b)#A ·lsst ∅) = (b ·sstp ∅)#(unlabel (A ·lsst ∅))"
by (simp add: subst_apply_labeled_stateful_strand_def)

lemma subst_lsst_unlabel: "unlabel (A ·lsst δ) = unlabel A ·sst δ"
proof (induction A)
  case (Cons a A)
  then obtain l b where "a = (l,b)" by (metis surj_pair)
  thus ?case
    using Cons
    by (simp add: subst_apply_labeled_stateful_strand_def subst_apply_stateful_strand_def)
qed simp

lemma subst_lsst_unlabel_member[intro]:
  assumes "x ∈ set (unlabel A)"
  shows "x ·sstp δ ∈ set (unlabel (A ·lsst δ))"
proof -
  obtain l where x: "(l,x) ∈ set A" using assms unfolding unlabel_def by moura
  thus ?thesis
    using subst_lsst_member
    by (metis unlabel_def in_set_zipE subst_apply_labeled_stateful_strand_step.simps zip_map_fst_snd)
qed

lemma subst_lsst_prefix:
  assumes "prefix B (A ·lsst ∅)"
  shows "∃ C. C ·lsst ∅ = B ∧ prefix C A"
using assms
proof (induction A rule: List.rev_induct)
  case (snoc a A) thus ?case
    proof (cases "B = A@[a] ·lsst ∅")
      case False thus ?thesis
        using snoc by (auto simp add: subst_lsst_append[of A] subst_lsst_cons)
    qed auto
  qed simp
qed simp

lemma dual_lsst_nil[simp]: "duallsst [] = []"
by (simp add: dual_lsst_def)

lemma dual_lsst_Cons[simp]:
  "duallsst ((l,send⟨t⟩)#A) = (l,receive⟨t⟩)#(duallsst A)"
  "duallsst ((l,receive⟨t⟩)#A) = (l,send⟨t⟩)#(duallsst A)"
  "duallsst ((l,⟨a: t ≐ s⟩)#A) = (l,⟨a: t ≐ s⟩)#(duallsst A)"
  "duallsst ((l,insert⟨t,s⟩)#A) = (l,insert⟨t,s⟩)#(duallsst A)"
  "duallsst ((l,delete⟨t,s⟩)#A) = (l,delete⟨t,s⟩)#(duallsst A)"
  "duallsst ((l,⟨a: t ∈ s⟩)#A) = (l,⟨a: t ∈ s⟩)#(duallsst A)"
  "duallsst ((l,∀X(∀≠: F ∨≠: G))#A) = (l,∀X(∀≠: F ∨≠: G))#(duallsst A)"
by (simp_all add: dual_lsst_def)

lemma dual_lsst_append[simp]: "duallsst (A@B) = duallsst A@duallsst B"
by (simp add: dual_lsst_def)

lemma dual_lsstp_subst: "duallsstp (s ·lsstp δ) = (duallsstp s) ·lsstp δ"
proof -
  obtain l x where s: "s = (l,x)" by moura
  thus ?thesis by (cases x) (auto simp add: subst_apply_labeled_stateful_strand_def)
qed

lemma dual_lsst_subst: "duallsst (S ·lsst δ) = (duallsst S) ·lsst δ"
proof (induction S)
  case (Cons s S) thus ?case
    using Cons dual_lsstp_subst[of s δ]
    by (simp add: dual_lsst_def subst_apply_labeled_stateful_strand_def)
  qed (simp add: dual_lsst_def subst_apply_labeled_stateful_strand_def)

```

```

lemma duallsst_subst_unlabel: "unlabel (duallsst (S ·lsst δ)) = unlabel (duallsst S) ·sst δ"
by (metis duallsst_subst subst_lsst_unlabel)

lemma duallsst_subst_cons: "duallsst (a#A ·lsst σ) = (duallsstp a ·lsstp σ)#(duallsst (A ·lsst σ))"
by (metis duallsst_subst list.simps(9) duallsst_def subst_apply_labeled_stateful_strand_def)

lemma duallsst_subst_append: "duallsst (A@B ·lsst σ) = (duallsst A@duallsst B) ·lsst σ"
by (metis (no_types) duallsst_subst duallsst_append)

lemma duallsst_subst_snoc: "duallsst (A@[a] ·lsst σ) = (duallsst A ·lsst σ)@[duallsstp a ·lsstp σ]"
by (metis duallsst_def duallsst_subst duallsst_subst_cons list.map(1) map_append
    subst_apply_labeled_stateful_strand_def)

lemma duallsst_memberD:
  assumes "(l,a) ∈ set (duallsst A)"
  shows "∃b. (l,b) ∈ set A ∧ duallsstp (l,b) = (l,a)"
  using assms
proof (induction A)
  case (Cons c A)
  hence "(l,a) ∈ set (duallsst A) ∨ duallsstp c = (l,a)" unfolding duallsst_def by force
  thus ?case
  proof
    assume "(l,a) ∈ set (duallsst A)" thus ?case using Cons.IH by auto
  next
    assume a: "duallsstp c = (l,a)"
    obtain i b where b: "c = (i,b)" by (metis surj_pair)
    thus ?case using a by (cases b) auto
  qed
qed simp

lemma duallsstp_inv:
  assumes "duallsstp (l, a) = (k, b)"
  shows "l = k"
  and "a = receive⟨t⟩ ⟹ b = send⟨t⟩"
  and "a = send⟨t⟩ ⟹ b = receive⟨t⟩"
  and "(∄t. a = receive⟨t⟩ ∨ a = send⟨t⟩) ⟹ b = a"
proof -
  show "l = k" using assms by (cases a) auto
  show "a = receive⟨t⟩ ⟹ b = send⟨t⟩" using assms by (cases a) auto
  show "a = send⟨t⟩ ⟹ b = receive⟨t⟩" using assms by (cases a) auto
  show "(∄t. a = receive⟨t⟩ ∨ a = send⟨t⟩) ⟹ b = a" using assms by (cases a) auto
qed

lemma duallsst_self_inverse: "duallsst (duallsst A) = A"
proof (induction A)
  case (Cons a A)
  obtain l b where "a = (l,b)" by (metis surj_pair)
  thus ?case using Cons by (cases b) auto
qed simp

lemma varssst_unlabel_duallsst_eq: "varslsst (duallsst A) = varslsst A"
proof (induction A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  thus ?case using Cons.IH by (cases b) auto
qed simp

lemma fvsst_unlabel_duallsst_eq: "fvlsst (duallsst A) = fvlsst A"
proof (induction A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  thus ?case using Cons.IH by (cases b) auto
qed simp

```



```

lemma bvarssst_unlabel_duallsst_eq: "bvarslsst (duallsst A) = bvarslsst A"
proof (induction A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  thus ?case using Cons.IH by (cases b) simp+
qed simp

lemma varssst_unlabel_Cons: "varslsst ((l,b)#A) = varssstp b ∪ varslsst A"
by (metis unlabel_Cons(1) varssst_Cons)

lemma fvsst_unlabel_Cons: "fvlsst ((l,b)#A) = fvsstp b ∪ fvlsst A"
by (metis unlabel_Cons(1) fvsst_Cons)

lemma bvarssst_unlabel_Cons: "bvarslsst ((l,b)#A) = set (bvarssstp b) ∪ bvarslsst A"
by (metis unlabel_Cons(1) bvarssst_Cons)

lemma bvarslsst_subst: "bvarslsst (A ·lsst δ) = bvarslsst A"
by (metis subst_lsst_unlabel bvarssst_subst)

lemma duallsst_member:
  assumes "(l,x) ∈ set A"
  and "¬is_Receive x" "¬is_Send x"
  shows "(l,x) ∈ set (duallsst A)"
using assms
proof (induction A)
  case (Cons a A) thus ?case using assms(2,3) by (cases x) (auto simp add: duallsst_def)
qed simp

lemma duallsst_unlabel_member:
  assumes "x ∈ set (unlabel A)"
  and "¬is_Receive x" "¬is_Send x"
  shows "x ∈ set (unlabel (duallsst A))"
using assms duallsst_member[of _ A]
by (meson unlabel_in unlabel_mem_has_label)

lemma duallsst_steps_iff:
  "(l,send⟨t⟩) ∈ set A ⟷ (l,receive⟨t⟩) ∈ set (duallsst A)"
  "(l,receive⟨t⟩) ∈ set A ⟷ (l,send⟨t⟩) ∈ set (duallsst A)"
  "(l,⟨c: t ≐ s⟩) ∈ set A ⟷ (l,⟨c: t ≐ s⟩) ∈ set (duallsst A)"
  "(l,insert⟨t,s⟩) ∈ set A ⟷ (l,insert⟨t,s⟩) ∈ set (duallsst A)"
  "(l,delete⟨t,s⟩) ∈ set A ⟷ (l,delete⟨t,s⟩) ∈ set (duallsst A)"
  "(l,⟨c: t ∈ s⟩) ∈ set A ⟷ (l,⟨c: t ∈ s⟩) ∈ set (duallsst A)"
  "(l,∀X(∀≠: F ∨≠: G)) ∈ set A ⟷ (l,∀X(∀≠: F ∨≠: G)) ∈ set (duallsst A)"
proof (induction A)
  case (Cons a A)
  obtain j b where a: "a = (j,b)" by (metis surj_pair)
  { case 1 thus ?case by (cases b) (simp_all add: Cons.IH(1) a duallsst_def) }
  { case 2 thus ?case by (cases b) (simp_all add: Cons.IH(2) a duallsst_def) }
  { case 3 thus ?case by (cases b) (simp_all add: Cons.IH(3) a duallsst_def) }
  { case 4 thus ?case by (cases b) (simp_all add: Cons.IH(4) a duallsst_def) }
  { case 5 thus ?case by (cases b) (simp_all add: Cons.IH(5) a duallsst_def) }
  { case 6 thus ?case by (cases b) (simp_all add: Cons.IH(6) a duallsst_def) }
  { case 7 thus ?case by (cases b) (simp_all add: Cons.IH(7) a duallsst_def) }
}
qed (simp_all add: duallsst_def)

lemma duallsst_unlabel_steps_iff:
  "send⟨t⟩ ∈ set (unlabel A) ⟷ receive⟨t⟩ ∈ set (unlabel (duallsst A))"
  "receive⟨t⟩ ∈ set (unlabel A) ⟷ send⟨t⟩ ∈ set (unlabel (duallsst A))"
  "⟨c: t ≐ s⟩ ∈ set (unlabel A) ⟷ ⟨c: t ≐ s⟩ ∈ set (unlabel (duallsst A))"
  "insert⟨t,s⟩ ∈ set (unlabel A) ⟷ insert⟨t,s⟩ ∈ set (unlabel (duallsst A))"
  "delete⟨t,s⟩ ∈ set (unlabel A) ⟷ delete⟨t,s⟩ ∈ set (unlabel (duallsst A))"
  "⟨c: t ∈ s⟩ ∈ set (unlabel A) ⟷ ⟨c: t ∈ s⟩ ∈ set (unlabel (duallsst A))"

```

```

"∀X(∀≠: F ∨≠: G) ∈ set (unlabel A) ↔ ∀X(∀≠: F ∨≠: G) ∈ set (unlabel (duallsst A))"
using duallsst_steps_iff(1,2)[of _ t A]
    duallsst_steps_iff(3,6)[of _ c t s A]
    duallsst_steps_iff(4,5)[of _ t s A]
    duallsst_steps_iff(7)[of _ X F G A]
by (meson unlabel_in unlabel_mem_has_label)+

lemma duallsst_list_all:
  "list_all is_Receive (unlabel A) ⇒ list_all is_Send (unlabel (duallsst A))"
  "list_all is_Send (unlabel A) ⇒ list_all is_Receive (unlabel (duallsst A))"
  "list_all is_Equality (unlabel A) ⇒ list_all is_Equality (unlabel (duallsst A))"
  "list_all is_Insert (unlabel A) ⇒ list_all is_Insert (unlabel (duallsst A))"
  "list_all is_Delete (unlabel A) ⇒ list_all is_Delete (unlabel (duallsst A))"
  "list_all is_InSet (unlabel A) ⇒ list_all is_InSet (unlabel (duallsst A))"
  "list_all is_NegChecks (unlabel A) ⇒ list_all is_NegChecks (unlabel (duallsst A))"
  "list_all is_Assignment (unlabel A) ⇒ list_all is_Assignment (unlabel (duallsst A))"
  "list_all is_Check (unlabel A) ⇒ list_all is_Check (unlabel (duallsst A))"
  "list_all is_Update (unlabel A) ⇒ list_all is_Update (unlabel (duallsst A))"
proof (induct A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  { case 1 thus ?case using Cons.hyps(1) a by (cases b) auto }
  { case 2 thus ?case using Cons.hyps(2) a by (cases b) auto }
  { case 3 thus ?case using Cons.hyps(3) a by (cases b) auto }
  { case 4 thus ?case using Cons.hyps(4) a by (cases b) auto }
  { case 5 thus ?case using Cons.hyps(5) a by (cases b) auto }
  { case 6 thus ?case using Cons.hyps(6) a by (cases b) auto }
  { case 7 thus ?case using Cons.hyps(7) a by (cases b) auto }
  { case 8 thus ?case using Cons.hyps(8) a by (cases b) auto }
  { case 9 thus ?case using Cons.hyps(9) a by (cases b) auto }
  { case 10 thus ?case using Cons.hyps(10) a by (cases b) auto }
qed simp_all

lemma duallsst_in_set_prefix_obtain:
  assumes "s ∈ set (unlabel (duallsst A))"
  shows "∃ l B s'. (l,s) = duallsstp (l,s') ∧ prefix (B@[l,s']) A"
  using assms
proof (induction A rule: List.rev_induct)
  case (snoc a A)
  obtain i b where a: "a = (i,b)" by (metis surj_pair)
  show ?case using snoc
  proof (cases "s ∈ set (unlabel (duallsst A))")
    case False thus ?thesis
      using a snoc.prem1 unlabel_append[of "duallsst A" "duallsst [a]" duallsst_append[of A "[a]"]
      by (cases b) (force simp add: unlabel_def duallsst_def)+
    qed auto
  qed simp
qed simp

lemma duallsst_in_set_prefix_obtain_subst:
  assumes "s ∈ set (unlabel (duallsst (A ·lsst ∅)))"
  shows "∃ l B s'. (l,s) = duallsstp ((l,s') ·lsstp ∅) ∧ prefix ((B ·lsst ∅)@[l,s']) (A ·lsst ∅)"
proof -
  obtain B l s' where B: "(l,s) = duallsstp (l,s') ∧ prefix (B@[l,s']) (A ·lsst ∅)"
  using duallsst_in_set_prefix_obtain[OF assms] by mouna

  obtain C where C: "C ·lsst ∅ = B@[l,s']"
  using substlsst_prefix[OF B(2)] by mouna

  obtain D u where D: "C = D@[l,u]" "D ·lsst ∅ = B" "[l,u] ·lsst ∅ = [(l, s')]"
  using substlsst_prefix[OF B(2)] substlsst_append_inv[OF C(1)]
  by (auto simp add: subst_apply_labeled_stateful_strand_def)

```

```

show ?thesis
  using B D subst_lsst_cons subst_lsst_singleton
  by (metis (no_types, lifting) nth_append_length)
qed

lemma trms_sst_unlabel_dual_lsst_eq: "trms_lsst (dual_lsst A) = trms_lsst A"
proof (induction A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  thus ?case using Cons.IH by (cases b) auto
qed simp

lemma trms_sst_unlabel_subst_cons:
  "trms_lsst ((l,b)#A ·_lsst δ) = trms_sstp (b ·_sstp δ) ∪ trms_lsst (A ·_lsst δ)"
by (metis subst_lsst_unlabel trms_sst_subst_cons unlabel_Cons(1))

lemma trms_sst_unlabel_subst:
  assumes "bvars_lsst S ∩ subst_domain ∅ = {}"
  shows "trms_lsst (S ·_lsst ∅) = trms_lsst S ·_set ∅"
by (metis trms_sst_subst[OF assms] subst_lsst_unlabel)

lemma trms_sst_unlabel_subst':
  fixes t:: "('a,'b) term" and δ:: "('a,'b) subst"
  assumes "t ∈ trms_lsst (S ·_lsst δ)"
  shows "∃ s ∈ trms_lsst S. ∃ X. set X ⊆ bvars_lsst S ∧ t = s · rm_vars (set X) δ"
using assms
proof (induction S)
  case (Cons a S)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  hence "t ∈ trms_lsst (S ·_lsst δ) ∨ t ∈ trms_sstp (b ·_sstp δ)"
  using Cons.prem1 trms_sst_unlabel_subst_cons by fast
  thus ?case
  proof
    assume *: "t ∈ trms_sstp (b ·_sstp δ)"
    show ?thesis using trms_sstp_subst'[OF *] a by auto
  next
    assume *: "t ∈ trms_lsst (S ·_lsst δ)"
    show ?thesis using Cons.IH[OF *] a by auto
  qed
qed simp

lemma trms_sst_unlabel_subst'':
  fixes t:: "('a,'b) term" and δ ∅:: "('a,'b) subst"
  assumes "t ∈ trms_lsst (S ·_lsst δ) ·_set ∅"
  shows "∃ s ∈ trms_lsst S. ∃ X. set X ⊆ bvars_lsst S ∧ t = s · rm_vars (set X) δ ∘_s ∅"
proof -
  obtain s where s: "s ∈ trms_lsst (S ·_lsst δ)" "t = s · ∅" using assms by mouna
  show ?thesis using trms_sst_unlabel_subst'[OF s(1)] s(2) by auto
qed

lemma trms_sst_unlabel_dual_subst_cons:
  "trms_lsst (dual_lsst (a#A ·_lsst σ)) = (trms_sstp (snd a ·_sstp σ)) ∪ (trms_lsst (dual_lsst (A ·_lsst σ)))"
proof -
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  thus ?thesis using a dual_lsst_subst_cons[of a A σ] by (cases b) auto
qed

lemma dual_lsst_funs_term:
  "⋃ (funs_term ' (trms_sst (unlabel (dual_lsst S)))) = ⋃ (funs_term ' (trms_sst (unlabel S)))"
using trms_sst_unlabel_dual_lsst_eq by fast

lemma dual_lsst_db_lsst:
  "db'_lsst (dual_lsst A) = db'_lsst A"

```

```

proof (induction A)
  case (Cons a A)
    obtain l b where "a = (l,b)" by (metis surj_pair)
    thus ?case using Cons by (cases b) auto
qed simp

lemma db_sst_unlabel_append:
  "db'_lsst (A@B) I D = db'_lsst B I (db'_lsst A I D)"
by (metis db_sst_append unlabel_append)

lemma db_sst_dual_lsst:
  "db'_sst (unlabel (dual_lsst (T ·lsst δ))) I D = db'_sst (unlabel (T ·lsst δ)) I D"
proof (induction T arbitrary: D)
  case (Cons x T)
    obtain l s where "x = (l,s)" by moura
    thus ?case
      using Cons
      by (cases s) (simp_all add: unlabel_def dual_lsst_def subst_apply_labeled_stateful_strand_def)
qed (simp add: unlabel_def dual_lsst_def subst_apply_labeled_stateful_strand_def)

lemma labeled_list_insert_eq_cases:
  "d ∉ set (unlabel D) ⟹ List.insert d (unlabel D) = unlabel (List.insert (i,d) D)"
  "(i,d) ∈ set D ⟹ List.insert d (unlabel D) = unlabel (List.insert (i,d) D)"
unfolding unlabel_def
by (metis (no_types, hide_lams) List.insert_def image_eqI list.simps(9) set_map snd_conv,
    metis in_set_insert set_zip_rightD zip_map_fst_snd)

lemma labeled_list_insert_eq_ex_cases:
  "List.insert d (unlabel D) = unlabel (List.insert (i,d) D) ∨
  (∃ j. (j,d) ∈ set D ∧ List.insert d (unlabel D) = unlabel (List.insert (j,d) D))"
using labeled_list_insert_eq_cases unfolding unlabel_def
by (metis in_set_impl_in_set_zip2 length_map zip_map_fst_snd)

lemma proj_subst: "proj l (A ·lsst δ) = proj l A ·lsst δ"
proof (induction A)
  case (Cons a A)
    obtain l b where "a = (l,b)" by (metis surj_pair)
    thus ?case using Cons unfolding proj_def subst_apply_labeled_stateful_strand_def by force
qed simp

lemma proj_set_subset[simp]:
  "set (proj n A) ⊆ set A"
unfolding proj_def by auto

lemma proj_proj_set_subset[simp]:
  "set (proj n (proj m A)) ⊆ set (proj n A)"
  "set (proj n (proj m A)) ⊆ set (proj m A)"
  "set (proj_unl n (proj m A)) ⊆ set (proj_unl n A)"
  "set (proj_unl n (proj m A)) ⊆ set (proj_unl m A)"
unfolding unlabel_def proj_def by auto

lemma proj_in_set_iff:
  "(ln i, d) ∈ set (proj i D) ⟷ (ln i, d) ∈ set D"
  "(*, d) ∈ set (proj i D) ⟷ (*, d) ∈ set D"
unfolding proj_def by auto

lemma proj_list_insert:
  "proj i (List.insert (ln i,d) D) = List.insert (ln i,d) (proj i D)"
  "proj i (List.insert (*,d) D) = List.insert (*,d) (proj i D)"
  "i ≠ j ⟹ proj i (List.insert (ln j,d) D) = proj i D"
unfolding List.insert_def proj_def by auto

lemma proj_filter: "proj i [d ← D. d ∉ set Di] = [d ← proj i D. d ∉ set Di]"

```

```

by (simp_all add: proj_def conj_commute)

lemma proj_list_Cons:
  "proj i ((ln i,d)#D) = (ln i,d)#proj i D"
  "proj i ((*,d)#D) = (*,d)#proj i D"
  "i ≠ j ⇒ proj i ((ln j,d)#D) = proj i D"
unfolding List.insert_def proj_def by auto

lemma proj_duallsst:
  "proj l (duallsst A) = duallsst (proj l A)"
proof (induction A)
  case (Cons a A)
  obtain k b where "a = (k,b)" by (metis surj_pair)
  thus ?case using Cons unfolding duallsst_def proj_def by (cases b) auto
qed simp

lemma proj_instance_ex:
  assumes B: "∀b ∈ set B. ∃a ∈ set A. ∃δ. b = a ·lsst δ ∧ P δ"
  and b: "b ∈ set (proj l B)"
  shows "∃a ∈ set (proj l A). ∃δ. b = a ·lsst δ ∧ P δ"
proof -
  obtain a δ where a: "a ∈ set A" "b = a ·lsst δ" "P δ" using B b proj_set_subset by fast
  obtain k b' where b': "b = (k, b')" "k = (ln l) ∨ k = *" using b proj_in_setD by metis
  obtain a' where a': "a = (k, a')" using b'(1) a(2) by (cases a) simp_all
  show ?thesis using a a' b'(2) unfolding proj_def by auto
qed

lemma proj_dbproj:
  "dbproj (ln i) (proj i D) = dbproj (ln i) D"
  "dbproj * (proj i D) = dbproj * D"
  "i ≠ j ⇒ dbproj (ln j) (proj i D) = []"
unfolding proj_def by (induct D) auto

lemma dbproj_Cons:
  "dbproj i ((i,d)#D) = (i,d)#dbproj i D"
  "i ≠ j ⇒ dbproj j ((i,d)#D) = dbproj j D"
by auto

lemma dbproj_subset[simp]:
  "set (unlabel (dbproj i D)) ⊆ set (unlabel D)"
unfolding unlabel_def by auto

lemma dbproj_subseq:
  assumes "Di ∈ set (subseqs (dbproj k D))"
  shows "dbproj k Di = Di" (is ?A)
  and "i ≠ k ⇒ dbproj i Di = []" (is "i ≠ k ⇒ ?B")
proof -
  have *: "set Di ⊆ set (dbproj k D)" using subseqs_powset[of "dbproj k D"] assms by auto
  thus ?A by (metis filter_True filter_set member_filter subsetCE)

  have "∧j d. (j,d) ∈ set Di ⇒ j = k" using * by auto
  moreover have "∧j d. (j,d) ∈ set (dbproj i Di) ⇒ j = i" by auto
  moreover have "∧j d. (j,d) ∈ set (dbproj i Di) ⇒ (j,d) ∈ set Di" by auto
  ultimately show "i ≠ k ⇒ ?B" by (metis set_empty subrelI subset_empty)
qed

lemma dbproj_subseq_subset:
  assumes "Di ∈ set (subseqs (dbproj i D))"
  shows "set Di ⊆ set D"
by (metis Pow_iff assms filter_set image_eqI member_filter subseqs_powset subsetCE subsetI)

lemma dbproj_subseq_in_subseqs:
  assumes "Di ∈ set (subseqs (dbproj i D))"

```

```

shows "Di ∈ set (subseqs D)"
using assms in_set_subseqs subseq_filter_left subseq_order.dual_order.trans by blast

lemma proj_subseq:
  assumes "Di ∈ set (subseqs (dbproj (ln j) D))" "j ≠ i"
  shows "[d ← proj i D. d ∉ set Di] = proj i D"
proof -
  have "set Di ⊆ set (dbproj (ln j) D)" using subseqs_powset[of "dbproj (ln j) D"] assms by auto
  hence "∧k d. (k,d) ∈ set Di ⟹ k = ln j" by auto
  moreover have "∧k d. (k,d) ∈ set (proj i D) ⟹ k ≠ ln j"
    using assms(2) unfolding proj_def by auto
  ultimately have "∧d. d ∈ set (proj i D) ⟹ d ∉ set Di" by auto
  thus ?thesis by simp
qed

lemma unlabel_subseqsD:
  assumes "A ∈ set (subseqs (unlabel B))"
  shows "∃C ∈ set (subseqs B). unlabel C = A"
using assms map_subseqs unfolding unlabel_def by (metis imageE set_map)

lemma unlabel_filter_eq:
  assumes "∀(j, p) ∈ set A ∪ B. ∀(k, q) ∈ set A ∪ B. p = q ⟹ j = k" (is "?P (set A)")
  shows "[d ← unlabel A. d ∉ snd ' B] = unlabel [d ← A. d ∉ B]"
using assms unfolding unlabel_def
proof (induction A)
  case (Cons a A)
  have "set A ⊆ set (a#A)" "{a} ⊆ set (a#A)" by auto
  hence *: "?P (set A)" "?P {a}" using Cons.prem by fast+
  hence IH: "[d ← map snd A . d ∉ snd ' B] = map snd [d ← A . d ∉ B]" using Cons.IH by auto

  { assume "snd a ∈ snd ' B"
    then obtain b where b: "b ∈ B" "snd a = snd b" by mouna
    hence "fst a = fst b" using *(2) by auto
    hence "a ∈ B" using b by (metis surjective_pairing)
  } hence **: "a ∉ B ⟹ snd a ∉ snd ' B" by metis

  show ?case by (cases "a ∈ B") (simp add: ** IH)+
qed simp

lemma subseqs_mem_dbproj:
  assumes "Di ∈ set (subseqs D)" "list_all (λd. fst d = i) Di"
  shows "Di ∈ set (subseqs (dbproj i D))"
using assms
proof (induction D arbitrary: Di)
  case (Cons di D)
  obtain d j where di: "di = (j,d)" by (metis surj_pair)
  show ?case
  proof (cases "Di ∈ set (subseqs D)")
    case True
    hence "Di ∈ set (subseqs (dbproj i D))" using Cons.IH Cons.prem by auto
    thus ?thesis using subseqs_Cons by auto
  next
    case False
    then obtain Di' where Di': "Di = di#Di'" using Cons.prem(1)
      by (metis (mono_tags, lifting) Un_iff imageE set_append set_map subseqs.simps(2))
    hence "Di' ∈ set (subseqs D)" using Cons.prem(1) False
      by (metis (no_types, lifting) UnE imageE list.inject set_append set_map subseqs.simps(2))
    hence "Di' ∈ set (subseqs (dbproj i D))" using Cons.IH Cons.prem Di' by auto
    moreover have "i = j" using Di' di Cons.prem(2) by auto
    hence "dbproj i (di#D) = di#dbproj i D" by (simp add: di)
    ultimately show ?thesis using Di'
      by (metis (no_types, lifting) UnCI image_eqI set_append set_map subseqs.simps(2))
  qed
qed

```

qed simp

lemma unlabel\_subst: "unlabel  $S \cdot_{sst} \delta$  = unlabel ( $S \cdot_{lsst} \delta$ )"  
 unfolding unlabel\_def subst\_apply\_stateful\_strand\_def subst\_apply\_labeled\_stateful\_strand\_def  
 by auto

lemma subterms\_subst\_lsst:  
 assumes " $\forall x \in fv_{set} (trms_{lsst} S). (\exists f. \sigma x = \text{Fun } f []) \vee (\exists y. \sigma x = \text{Var } y)$ "  
 and " $bvars_{lsst} S \cap \text{subst\_domain } \sigma = \{\}$ "  
 shows " $\text{subterms}_{set} (trms_{lsst} (S \cdot_{lsst} \sigma)) = \text{subterms}_{set} (trms_{lsst} S) \cdot_{set} \sigma$ "  
 using subterms\_subst'' [OF assms(1)] trms\_sst\_subst [OF assms(2)] unlabel\_subst [of  $S \sigma$ ]  
 by simp

lemma subterms\_subst\_lsst\_ik:  
 assumes " $\forall x \in fv_{set} (ik_{lsst} S). (\exists f. \sigma x = \text{Fun } f []) \vee (\exists y. \sigma x = \text{Var } y)$ "  
 shows " $\text{subterms}_{set} (ik_{lsst} (S \cdot_{lsst} \sigma)) = \text{subterms}_{set} (ik_{lsst} S) \cdot_{set} \sigma$ "  
 using subterms\_subst'' [OF assms(1)] ik\_sst\_subst [of "unlabel  $S$ "  $\sigma$ ] unlabel\_subst [of  $S \sigma$ ]  
 by simp

lemma labeled\_stateful\_strand\_subst\_comp:  
 assumes " $\text{range\_vars } \delta \cap bvars_{lsst} S = \{\}$ "  
 shows " $S \cdot_{lsst} \delta \circ_s \vartheta = (S \cdot_{lsst} \delta) \cdot_{lsst} \vartheta$ "  
 using assms  
 proof (induction  $S$ )  
 case (Cons  $s S$ )  
 obtain  $l x$  where  $s = (l, x)$  by (metis surj\_pair)  
 hence IH: " $S \cdot_{lsst} \delta \circ_s \vartheta = (S \cdot_{lsst} \delta) \cdot_{lsst} \vartheta$ " using Cons by auto  
  
 have " $x \cdot_{sstp} \delta \circ_s \vartheta = (x \cdot_{sstp} \delta) \cdot_{sstp} \vartheta$ "  
 using  $s$  Cons.premis stateful\_strand\_step\_subst\_comp [of  $\delta x \vartheta$ ] by auto  
 thus ?case using  $s$  IH by (simp add: subst\_apply\_labeled\_stateful\_strand\_def)  
 qed simp

lemma sst\_vars\_proj\_subset[simp]:  
 " $fv_{sst} (\text{proj\_unl } n A) \subseteq fv_{sst} (\text{unlabel } A)$ "  
 " $bvars_{sst} (\text{proj\_unl } n A) \subseteq bvars_{sst} (\text{unlabel } A)$ "  
 " $vars_{sst} (\text{proj\_unl } n A) \subseteq vars_{sst} (\text{unlabel } A)$ "  
 using vars\_sst\_is\_fv\_sst\_bvars\_sst [of "unlabel  $A$ "]  
 vars\_sst\_is\_fv\_sst\_bvars\_sst [of "proj\_unl  $n A$ "]  
 unfolding unlabel\_def proj\_def by auto

lemma trms\_sst\_proj\_subset[simp]:  
 " $trms_{sst} (\text{proj\_unl } n A) \subseteq trms_{sst} (\text{unlabel } A)$ " (is ?A)  
 " $trms_{sst} (\text{proj\_unl } m (\text{proj } n A)) \subseteq trms_{sst} (\text{proj\_unl } n A)$ " (is ?B)  
 " $trms_{sst} (\text{proj\_unl } m (\text{proj } n A)) \subseteq trms_{sst} (\text{proj\_unl } m A)$ " (is ?C)  
 proof -  
 show ?A unfolding unlabel\_def proj\_def by auto  
 show ?B using trms\_sst\_mono [OF proj\_proj\_set\_subset(4)] by metis  
 show ?C using trms\_sst\_mono [OF proj\_proj\_set\_subset(3)] by metis  
 qed

lemma trms\_sst\_unlabel\_prefix\_subset:  
 " $trms_{sst} (\text{unlabel } A) \subseteq trms_{sst} (\text{unlabel } (A@B))$ " (is ?A)  
 " $trms_{sst} (\text{proj\_unl } n A) \subseteq trms_{sst} (\text{proj\_unl } n (A@B))$ " (is ?B)  
 using trms\_sst\_mono [of "proj\_unl  $n A$ " "proj\_unl  $n (A@B)"]$   
 unfolding unlabel\_def proj\_def by auto

lemma trms\_sst\_unlabel\_suffix\_subset:  
 " $trms_{sst} (\text{unlabel } B) \subseteq trms_{sst} (\text{unlabel } (A@B))$ "  
 " $trms_{sst} (\text{proj\_unl } n B) \subseteq trms_{sst} (\text{proj\_unl } n (A@B))$ "  
 using trms\_sst\_mono [of "proj\_unl  $n B$ " "proj\_unl  $n (A@B)"]$   
 unfolding unlabel\_def proj\_def by auto

```

lemma setopslsstpD:
  assumes p: "p ∈ setopslsstp a"
  shows "fst p = fst a" (is ?P)
    and "is_Update (snd a) ∨ is_InSet (snd a) ∨ is_NegChecks (snd a)" (is ?Q)
proof -
  obtain l k p' a' where a: "p = (l,p')" "a = (k,a')" by (metis surj_pair)
  show ?P using p a by (cases a') auto
  show ?Q using p a by (cases a') auto
qed

lemma setopslsst_nil[simp]:
  "setopslsst [] = {}"
by (simp add: setopslsst_def)

lemma setopslsst_cons[simp]:
  "setopslsst (x#S) = setopslsstp x ∪ setopslsst S"
by (simp add: setopslsst_def)

lemma setopssst_proj_subset:
  "setopssst (proj_unl n A) ⊆ setopssst (unlabel A)"
  "setopssst (proj_unl m (proj n A)) ⊆ setopssst (proj_unl n A)"
  "setopssst (proj_unl m (proj n A)) ⊆ setopssst (proj_unl m A)"
unfolding unlabel_def proj_def
proof (induction A)
  case (Cons a A)
  obtain l b where lb: "a = (l,b)" by moura
  { case 1 thus ?case using Cons.IH lb by (cases b) (auto simp add: setopssst_def) }
  { case 2 thus ?case using Cons.IH lb by (cases b) (auto simp add: setopssst_def) }
  { case 3 thus ?case using Cons.IH lb by (cases b) (auto simp add: setopssst_def) }
qed simp_all

lemma setopssst_unlabel_prefix_subset:
  "setopssst (unlabel A) ⊆ setopssst (unlabel (A@B))"
  "setopssst (proj_unl n A) ⊆ setopssst (proj_unl n (A@B))"
unfolding unlabel_def proj_def
proof (induction A)
  case (Cons a A)
  obtain l b where lb: "a = (l,b)" by moura
  { case 1 thus ?case using Cons.IH lb by (cases b) (auto simp add: setopssst_def) }
  { case 2 thus ?case using Cons.IH lb by (cases b) (auto simp add: setopssst_def) }
qed (simp_all add: setopssst_def)

lemma setopssst_unlabel_suffix_subset:
  "setopssst (unlabel B) ⊆ setopssst (unlabel (A@B))"
  "setopssst (proj_unl n B) ⊆ setopssst (proj_unl n (A@B))"
unfolding unlabel_def proj_def
proof (induction A)
  case (Cons a A)
  obtain l b where lb: "a = (l,b)" by moura
  { case 1 thus ?case using Cons.IH lb by (cases b) (auto simp add: setopssst_def) }
  { case 2 thus ?case using Cons.IH lb by (cases b) (auto simp add: setopssst_def) }
qed simp_all

lemma setopslsst_proj_subset:
  "setopslsst (proj n A) ⊆ setopslsst A"
  "setopslsst (proj m (proj n A)) ⊆ setopslsst (proj n A)"
unfolding proj_def setopslsst_def by auto

lemma setopslsst_prefix_subset:
  "setopslsst A ⊆ setopslsst (A@B)"
  "setopslsst (proj n A) ⊆ setopslsst (proj n (A@B))"
unfolding proj_def setopslsst_def by auto

```



```

lemma setopslsst_suffix_subset:
  "setopslsst B ⊆ setopslsst (A@B)"
  "setopslsst (proj n B) ⊆ setopslsst (proj n (A@B))"
unfolding proj_def setopslsst_def by auto

lemma setopslsst_mono:
  "set M ⊆ set N ⟹ setopslsst M ⊆ setopslsst N"
by (auto simp add: setopslsst_def)

lemma trmssst_unlabel_subset_if_no_label:
  "¬list_ex (is_LabelN 1) A ⟹ trmslsst (proj 1 A) ⊆ trmslsst (proj 1' A)"
by (rule trmssst_mono[OF proj_subset_if_no_label(2)[of 1 A 1']])

lemma setopssst_unlabel_subset_if_no_label:
  "¬list_ex (is_LabelN 1) A ⟹ setopssst (proj_unl 1 A) ⊆ setopssst (proj_unl 1' A)"
by (rule setopssst_mono[OF proj_subset_if_no_label(2)[of 1 A 1']])

lemma setopslsst_proj_subset_if_no_label:
  "¬list_ex (is_LabelN 1) A ⟹ setopslsst (proj 1 A) ⊆ setopslsst (proj 1' A)"
by (rule setopslsst_mono[OF proj_subset_if_no_label(1)[of 1 A 1']])

lemma setopslsstp_subst_cases[simp]:
  "setopslsstp ((1, send⟨t⟩) ·lsstp δ) = {}"
  "setopslsstp ((1, receive⟨t⟩) ·lsstp δ) = {}"
  "setopslsstp ((1, ⟨ac: s ≐ t⟩) ·lsstp δ) = {}"
  "setopslsstp ((1, insert⟨t, s⟩) ·lsstp δ) = {(1, t · δ, s · δ)}"
  "setopslsstp ((1, delete⟨t, s⟩) ·lsstp δ) = {(1, t · δ, s · δ)}"
  "setopslsstp ((1, ⟨ac: t ∈ s⟩) ·lsstp δ) = {(1, t · δ, s · δ)}"
  "setopslsstp ((1, ∀X⟨≠: F ∨ ≠: F'⟩) ·lsstp δ) =
    ((λ(t, s). (1, t · rm_vars (set X) δ, s · rm_vars (set X) δ)) ' set F')" (is "?A = ?B")
proof -
  have "?A = (λ(t, s). (1, t, s)) ' set (F' ·pairs rm_vars (set X) δ)" by auto
  thus "?A = ?B" unfolding subst_apply_pairs_def by auto
qed simp_all

lemma setopslsstp_subst:
  assumes "set (bvarssstp (snd a)) ∩ subst_domain ϑ = {}"
  shows "setopslsstp (a ·lsstp ϑ) = (λp. (fst a, snd p ·p ϑ)) ' setopslsstp a"
proof -
  obtain 1 a' where a: "a = (1, a')" by (metis surj_pair)
  show ?thesis
  proof (cases a')
    case (NegChecks X F G)
    hence *: "rm_vars (set X) ϑ = ϑ" using a assms rm_vars_apply'[of ϑ "set X"] by auto
    have "setopslsstp (a ·lsstp ϑ) = (λp. (fst a, p)) ' set (G ·pairs ϑ)"
    using * NegChecks a by auto
    moreover have "setopslsstp a = (λp. (fst a, p)) ' set G" using NegChecks a by simp
    hence "(λp. (fst a, snd p ·p ϑ)) ' setopslsstp a = (λp. (fst a, p ·p ϑ)) ' set G"
    by (metis (mono_tags, lifting) image_cong image_image snd_conv)
    hence "(λp. (fst a, snd p ·p ϑ)) ' setopslsstp a = (λp. (fst a, p)) ' (set G ·pset ϑ)"
    unfolding case_prod_unfold by auto
    ultimately show ?thesis by (simp add: subst_apply_pairs_def)
  qed (use a in simp_all)
qed

lemma setopslsstp_subst':
  assumes "set (bvarssstp (snd a)) ∩ subst_domain ϑ = {}"
  shows "setopslsstp (a ·lsstp ϑ) = (λ(i, p). (i, p ·p ϑ)) ' setopslsstp a"
using setopslsstp_subst[OF assms] setopslsstpD(1) unfolding case_prod_unfold
by (metis (mono_tags, lifting) image_cong)

lemma setopslsst_subst:
  assumes "bvarslsst S ∩ subst_domain ϑ = {}"

```

```

shows "setopslsst (S ·lsst ∅) = (λp. (fst p, snd p ·p ∅)) ' setopslsst S"
using assms
proof (induction S)
  case (Cons a S)
  have "bvarslsst S ∩ subst_domain ∅ = {}" and *: "set (bvarssstp (snd a)) ∩ subst_domain ∅ = {}"
    using Cons.prem by auto
  hence IH: "setopslsst (S ·lsst ∅) = (λp. (fst p, snd p ·p ∅)) ' setopslsst S"
    using Cons.IH by auto
  show ?case
    using setopslsstp-subst'[OF *] IH
    unfolding setopslsst-def case_prod_unfold subst_lsst_cons
    by auto
qed (simp add: setopssstp-def)

lemma setopslsstp-in_subst:
  assumes p: "p ∈ setopslsstp (a ·lsstp δ)"
  shows "∃q ∈ setopslsstp a. fst p = fst q ∧ snd p = snd q ·p rm_vars (set (bvarssstp (snd a))) δ"
    (is "∃q ∈ setopslsstp a. ?P q")
proof -
  obtain l b where a: "a = (l,b)" by (metis surj_pair)

  show ?thesis
  proof (cases b)
    case (NegChecks X F F')
    hence "p ∈ (λ(t, s). (l, t · rm_vars (set X) δ, s · rm_vars (set X) δ)) ' set F'"
      using p a setopslsstp-subst_cases(7)[of l X F F' δ] by blast
    then obtain s t where st:
      "(t,s) ∈ set F'" "p = (l, t · rm_vars (set X) δ, s · rm_vars (set X) δ)"
    by auto
    hence "(l,t,s) ∈ setopslsstp a" "fst p = fst (l,t,s)"
      "snd p = snd (l,t,s) ·p rm_vars (set X) δ"
    using a NegChecks by fastforce+
    moreover have "bvarssstp (snd a) = X" using NegChecks a by auto
    ultimately show ?thesis by blast
  qed (use p a in auto)
qed

lemma setopslsst-in_subst:
  assumes "p ∈ setopslsst (A ·lsst δ)"
  shows "∃q ∈ setopslsst A. fst p = fst q ∧ (∃X ⊆ bvarslsst A. snd p = snd q ·p rm_vars X δ)"
    (is "∃q ∈ setopslsst A. ?P A q")
  using assms
proof (induction A)
  case (Cons a A)
  note 0 = unlabel_Cons(2)[of a A] bvarssst-Cons[of "snd a" "unlabel A"]
  show ?case
  proof (cases "p ∈ setopslsst (A ·lsst δ)")
    case False
    hence "p ∈ setopslsstp (a ·lsstp δ)"
      using Cons.prem setopslsst-cons[of "a ·lsstp δ" "A ·lsst δ"] subst_lsst_cons[of a A δ] by auto
    moreover have "(set (bvarssstp (snd a))) ⊆ bvarslsst (a#A)" using 0 by simp
    ultimately have "∃q ∈ setopslsstp a. ?P (a#A) q" using setopslsstp-in_subst[of p a δ] by blast
    thus ?thesis by auto
  qed (use Cons.IH 0 in auto)
qed simp

lemma setopslsst-duallsst-eq:
  "setopslsst (duallsst A) = setopslsst A"
proof (induction A)
  case (Cons a A)
  obtain l b where "a = (l,b)" by (metis surj_pair)
  thus ?case using Cons unfolding setopslsst-def duallsst-def by (cases b) auto
qed simp

```

end

## 6.2 Stateful Protocol Compositionality (Stateful\_Compositionality)

```
theory Stateful_Compositionality
imports Stateful_Typing Parallel_Compositionality Labeled_Stateful_Strands
begin
```

### 6.2.1 Small Lemmata

```
lemma (in typed_model) wt_subst_sstp_vars_type_subset:
  fixes a::('fun,'var) stateful_strand_step
  assumes "wt_subst δ"
  and "∀t ∈ subst_range δ. fv t = {} ∨ (∃x. t = Var x)"
  shows "Γ ' Var ' fv_sstp (a `sstp δ) ⊆ Γ ' Var ' fv_sstp a" (is ?A)
  and "Γ ' Var ' set (bvars_sstp (a `sstp δ)) = Γ ' Var ' set (bvars_sstp a)" (is ?B)
  and "Γ ' Var ' vars_sstp (a `sstp δ) ⊆ Γ ' Var ' vars_sstp a" (is ?C)
proof -
  show ?A
  proof
    fix τ assume τ: "τ ∈ Γ ' Var ' fv_sstp (a `sstp δ)"
    then obtain x where x: "x ∈ fv_sstp (a `sstp δ)" "Γ (Var x) = τ" by moura

    show "τ ∈ Γ ' Var ' fv_sstp a"
  proof (cases "x ∈ fv_sstp a")
    case False
    hence "∃y ∈ fv_sstp a. δ y = Var x"
    proof (cases a)
    case (NegChecks X F G)
    hence *: "x ∈ fv_pairs (F `pairs rm_vars (set X) δ) ∪ fv_pairs (G `pairs rm_vars (set X) δ)"
      "x ∉ set X"
    using fv_sstp_NegCheck(1)[of X "F `pairs rm_vars (set X) δ" "G `pairs rm_vars (set X) δ"]
      fv_sstp_NegCheck(1)[of X F G] False x(1)
    by fastforce+

    obtain y where y: "y ∈ fv_pairs F ∪ fv_pairs G" "x ∈ fv (rm_vars (set X) δ y)"
    using fv_pairs_subst_obtain_var[of _ _ "rm_vars (set X) δ"]
      fv_pairs_subst_obtain_var[of _ _ "rm_vars (set X) δ"]
      *(1)
    by blast

    have "fv (rm_vars (set X) δ z) = {} ∨ (∃u. rm_vars (set X) δ z = Var u)" for z
    using assms(2) rm_vars_img_subset[of "set X" δ] by blast
    hence "rm_vars (set X) δ y = Var x" using y(2) by fastforce
    hence "∃y ∈ fv_sstp a. rm_vars (set X) δ y = Var x"
    using y fv_sstp_NegCheck(1)[of X F G] NegChecks *(2) by fastforce
    thus ?thesis by (metis (full_types) *(2) term.inject(1))
  qed (use assms(2) x(1) subst_apply_img_var' [of x _ δ] in fastforce)+
  then obtain y where y: "y ∈ fv_sstp a" "δ y = Var x" by moura
  hence "Γ (Var y) = τ" using x(2) assms(1) by (simp add: wt_subst_def)
  thus ?thesis using y(1) by auto
qed (use x in auto)

show ?B by (metis bvars_sstp_subst)

show ?C
proof
  fix τ assume τ: "τ ∈ Γ ' Var ' vars_sstp (a `sstp δ)"
  then obtain x where x: "x ∈ vars_sstp (a `sstp δ)" "Γ (Var x) = τ" by moura
```

```

show "τ ∈ Γ ' Var ' varssstp a"
proof (cases "x ∈ varssstp a")
  case False
  hence "∃ y ∈ varssstp a. δ y = Var x"
  proof (cases a)
    case (NegChecks X F G)
    hence *: "x ∈ fvpairs (F ·pairs rm_vars (set X) δ) ∪ fvpairs (G ·pairs rm_vars (set X) δ)"
      "x ∉ set X"
    using varssstp_NegCheck[of X "F ·pairs rm_vars (set X) δ" "G ·pairs rm_vars (set X) δ"]
      varssstp_NegCheck[of X F G] False x(1)
    by (fastforce, blast)

  obtain y where y: "y ∈ fvpairs F ∪ fvpairs G" "x ∈ fv (rm_vars (set X) δ y)"
    using fvpairs_subst_obtain_var[of _ _ "rm_vars (set X) δ"]
      fvpairs_subst_obtain_var[of _ _ "rm_vars (set X) δ"]
      *(1)
    by blast

  have "fv (rm_vars (set X) δ z) = {} ∨ (∃ u. rm_vars (set X) δ z = Var u)" for z
    using assms(2) rm_vars_img_subset[of "set X" δ] by blast
  hence "rm_vars (set X) δ y = Var x" using y(2) by fastforce
  hence "∃ y ∈ varssstp a. rm_vars (set X) δ y = Var x"
    using y varssstp_NegCheck[of X F G] NegChecks by blast
  thus ?thesis by (metis (full_types) *(2) term.inject(1))
qed (use assms(2) x(1) subst_apply_img_var'[of x _ δ] in fastforce)+
then obtain y where y: "y ∈ varssstp a" "δ y = Var x" by moura
hence "Γ (Var y) = τ" using x(2) assms(1) by (simp add: wtsubst_def)
thus ?thesis using y(1) by auto
qed (use x in auto)
qed
qed

```

```

lemma (in typed_model) wt_subst_lsst_vars_type_subset:
  fixes A::('fun,'var,'a) labeled_stateful_strand"
  assumes "wtsubst δ"
  and "∀ t ∈ subst_range δ. fv t = {} ∨ (∃ x. t = Var x)"
  shows "Γ ' Var ' fvlsst (A ·lsst δ) ⊆ Γ ' Var ' fvlsst A" (is ?A)
  and "Γ ' Var ' bvarslsst (A ·lsst δ) = Γ ' Var ' bvarslsst A" (is ?B)
  and "Γ ' Var ' varslsst (A ·lsst δ) ⊆ Γ ' Var ' varslsst A" (is ?C)
proof -
  have "varslsst (a#A ·lsst δ) = varssstp (b ·sstp δ) ∪ varslsst (A ·lsst δ)"
    "varslsst (a#A) = varssstp b ∪ varslsst A"
    "fvlsst (a#A ·lsst δ) = fvsstp (b ·sstp δ) ∪ fvlsst (A ·lsst δ)"
    "fvlsst (a#A) = fvsstp b ∪ fvlsst A"
    "bvarslsst (a#A ·lsst δ) = set (bvarssstp (b ·sstp δ)) ∪ bvarslsst (A ·lsst δ)"
    "bvarslsst (a#A) = set (bvarssstp b) ∪ bvarslsst A"
  when "a = (1,b)" for a l b and A::('fun,'var,'a) labeled_stateful_strand"
  using that unlabel_Cons(1)[of l b A] unlabel_subst[of "a#A" δ]
    subst_lsst_cons[of a A δ] subst_sst_cons[of b "unlabel A" δ]
    subst_apply_labeled_stateful_strand_step.simps(1)[of l b δ]
    varssstp_unlabel_Cons[of l b A] varssstp_unlabel_Cons[of l "b ·sstp δ" "A ·lsst δ"]
    fvsstp_unlabel_Cons[of l b A] fvsstp_unlabel_Cons[of l "b ·sstp δ" "A ·lsst δ"]
    bvarssstp_unlabel_Cons[of l b A] bvarssstp_unlabel_Cons[of l "b ·sstp δ" "A ·lsst δ"]
  by simp_all
  hence *: "Γ ' Var ' varslsst (a#A ·lsst δ) =
    Γ ' Var ' varssstp (b ·sstp δ) ∪ Γ ' Var ' varslsst (A ·lsst δ)"
    "Γ ' Var ' varslsst (a#A) = Γ ' Var ' varssstp b ∪ Γ ' Var ' varslsst A"
    "Γ ' Var ' fvlsst (a#A ·lsst δ) =
    Γ ' Var ' fvsstp (b ·sstp δ) ∪ Γ ' Var ' fvlsst (A ·lsst δ)"
    "Γ ' Var ' fvlsst (a#A) = Γ ' Var ' fvsstp b ∪ Γ ' Var ' fvlsst A"
    "Γ ' Var ' bvarslsst (a#A ·lsst δ) =
    Γ ' Var ' set (bvarssstp (b ·sstp δ)) ∪ Γ ' Var ' bvarslsst (A ·lsst δ)"
    "Γ ' Var ' bvarslsst (a#A) = Γ ' Var ' set (bvarssstp b) ∪ Γ ' Var ' bvarslsst A"

```

```

when "a = (l,b)" for a l b and A::('fun,'var,'a) labeled_stateful_strand"
using that by fast+

have "?A ∧ ?B ∧ ?C"
proof (induction A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)

  show ?case
    using Cons.IH wt_subst_sstp_vars_type_subset[OF assms, of b] * [OF a, of A]
    by (metis Un_mono)
qed simp
thus ?A ?B ?C by metis+
qed

lemma (in stateful_typed_model) fv_pair_fv_pairs_subset:
  assumes "d ∈ set D"
  shows "fv (pair (snd d)) ⊆ fv_pairs (unlabel D)"
using assms unfolding pair_def by (induct D) (auto simp add: unlabel_def)

lemma (in stateful_typed_model) labeled_sat_ineq_lift:
  assumes "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))])_st) [d←dbproj i D. d ∉ set Di]]_d I"
  (is "?R1 D")
  and "∀(j,p) ∈ {(i,t,s)} ∪ set D ∪ set Di. ∀(k,q) ∈ {(i,t,s)} ∪ set D ∪ set Di.
    (∃δ. Unifier δ (pair p) (pair q)) ⟶ j = k" (is "?R2 D")
  shows "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))])_st) [d←D. d ∉ set Di]]_d I"
using assms
proof (induction D)
  case (Cons dl D)
  obtain d l where dl: "dl = (l,d)" by (metis surj_pair)

  have 1: "?R1 D"
  proof (cases "i = 1")
    case True thus ?thesis using Cons.prem1 dl by (cases "dl ∈ set Di") auto
  next
    case False thus ?thesis using Cons.prem1 dl by auto
  qed

  have "set D ⊆ set (dl#D)" by auto
  hence 2: "?R2 D" using Cons.prem2 by blast

  have "i ≠ 1 ∨ dl ∈ set Di ∨ [M; [∀X(∀≠: [(pair (t,s), pair (snd dl))])_st]]_d I"
  using Cons.prem1 dl by (auto simp add: ineq_model_def)
  moreover have "∃δ. Unifier δ (pair (t,s)) (pair d) ⟶ i = 1"
  using Cons.prem2 dl by force
  ultimately have 3: "dl ∈ set Di ∨ [M; [∀X(∀≠: [(pair (t,s), pair (snd dl))])_st]]_d I"
  using strand_sem_not_unif_is_sat_ineq[of "pair (t,s)" "pair d"] dl by fastforce

  show ?case using Cons.IH[OF 1 2] 3 dl by auto
qed simp

lemma (in stateful_typed_model) labeled_sat_ineq_dbproj:
  assumes "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))])_st) [d←D. d ∉ set Di]]_d I"
  (is "?P D")
  shows "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))])_st) [d←dbproj i D. d ∉ set Di]]_d I"
  (is "?Q D")
using assms
proof (induction D)
  case (Cons di D)
  obtain d j where di: "di = (j,d)" by (metis surj_pair)

  have "?P D" using Cons.prem by (cases "di ∈ set Di") auto
  hence IH: "?Q D" by (metis Cons.IH)

```

```

show ?case using di IH
proof (cases "i = j  $\wedge$  di  $\notin$  set Di")
  case True
    have 1: "[M;  $\forall X \langle \forall \neq: [(pair (t,s), pair (snd di))] \rangle_{st} ] ]_d \mathcal{I}$ "
      using Cons.prem True by auto
    have 2: "dbproj i (di#D) = di#dbproj i D" using True dbproj_Cons(1) di by auto
    show ?thesis using 1 2 IH by auto
  qed auto
qed simp

lemma (in stateful_typed_model) labeled_sat_ineq_dbproj_sem_equiv:
  assumes " $\forall (j,p) \in ((\lambda(t, s). (i, t, s)) 'set F') \cup set D.$ "
    " $\forall (k,q) \in ((\lambda(t, s). (i, t, s)) 'set F') \cup set D.$ "
    " $(\exists \delta. \text{Unifier } \delta (pair p) (pair q)) \longrightarrow j = k$ "
  and "fvpairs (map snd D)  $\cap$  set X = {}"
  shows "[M; map ( $\lambda G. \forall X \langle \forall \neq: (F@G) \rangle_{st} \rangle (tr_{pairs} F' (map snd D)) ] ]_d \mathcal{I} \longleftrightarrow$ 
    [M; map ( $\lambda G. \forall X \langle \forall \neq: (F@G) \rangle_{st} \rangle (tr_{pairs} F' (map snd (dbproj i D))) ] ]_d \mathcal{I}$ "
proof -
  let ?A = "set (map snd D)  $\cdot_{pset} \mathcal{I}$ "
  let ?B = "set (map snd (dbproj i D))  $\cdot_{pset} \mathcal{I}$ "
  let ?C = "set (map snd D) - set (map snd (dbproj i D))"
  let ?F = " $(\lambda(t, s). (i, t, s)) 'set F'$ "
  let ?P = " $\lambda \delta. \text{subst\_domain } \delta = set X \wedge \text{ground (subst\_range } \delta)$ "

  have 1: " $\forall (t, t') \in set (map snd D). (fv t \cup fv t') \cap set X = \{ \}$ "
    " $\forall (t, t') \in set (map snd (dbproj i D)). (fv t \cup fv t') \cap set X = \{ \}$ "
    using assms(2) dbproj_subset[of i D] unfolding unlabel_def by force+

  have 2: "?B  $\subseteq$  ?A" by auto

  have 3: " $\neg \text{Unifier } \delta (pair f) (pair d)$ "
    when f: "f  $\in$  set F'" and d: "d  $\in$  set (map snd D) - set (map snd (dbproj i D))"
    for f d and  $\delta::('fun, 'var) \text{subst}$ 
  proof -
    obtain k where k: "(k,d)  $\in$  set D - set (dbproj i D)"
      using d by force

    have "(i,f)  $\in ((\lambda(t, s). (i, t, s)) 'set F') \cup set D$ "
      "(k,d)  $\in ((\lambda(t, s). (i, t, s)) 'set F') \cup set D$ "
      using f k by auto
    hence "i = k" when "Unifier  $\delta (pair f) (pair d)$ " for  $\delta$ 
      using assms(1) that by blast
    moreover have "k  $\neq$  i" using k d by simp
    ultimately show ?thesis by metis
  qed

  have "f  $\cdot_p \delta \neq d \cdot_p \delta$ "
    when "f  $\in$  set F'" "d  $\in$  ?C" for f d and  $\delta::('fun, 'var) \text{subst}$ 
    by (metis fun_pair_eq_subst 3[OF that])
  hence "f  $\cdot_p (\delta \circ_s \mathcal{I}) \notin ?C \cdot_{pset} (\delta \circ_s \mathcal{I})$ "
    when "f  $\in$  set F'" for f and  $\delta::('fun, 'var) \text{subst}$ 
    using that by blast
  moreover have "?C  $\cdot_{pset} \delta \cdot_{pset} \mathcal{I} = ?C \cdot_{pset} \mathcal{I}$ "
    when "?P  $\delta$ " for  $\delta$ 
    using assms(2) that pairs_substI[of  $\delta$  "(set (map snd D) - set (map snd (dbproj i D)))"]
    by blast
  ultimately have 4: "f  $\cdot_p (\delta \circ_s \mathcal{I}) \notin ?C \cdot_{pset} \mathcal{I}$ "
    when "f  $\in$  set F'" "?P  $\delta$ " for f and  $\delta::('fun, 'var) \text{subst}$ 
    by (metis that subst_pairs_compose)

  { fix f and  $\delta::('fun, 'var) \text{subst}$ 
    assume "f  $\in$  set F'" "?P  $\delta$ "

```

```

  hence "f ·p (δ ∘s I) ∉ ?C ·pset I" by (metis 4)
  hence "f ·p (δ ∘s I) ∉ ?A - ?B" by force
} hence 5: "∀ f ∈ set F'. ∀ δ. ?P δ ⟶ f ·p (δ ∘s I) ∉ ?A - ?B" by metis

```

```
show ?thesis
```

```

  using negchecks_model_db_subset[OF 2]
      negchecks_model_db_supset[OF 2 5]
      tr_pairs_sem_equiv[OF 1(1)]
      tr_pairs_sem_equiv[OF 1(2)]
      tr_NegChecks_constr_iff(1)
      strand_sem_eq_defs(2)
  by (metis (no_types, lifting))

```

```
qed
```

```
lemma (in stateful_typed_model) labeled_sat_eqs_list_all:
```

```

  assumes "∀ (j, p) ∈ {(i,t,s)} ∪ set D. ∀ (k,q) ∈ {(i,t,s)} ∪ set D.
    (∃ δ. Unifier δ (pair p) (pair q)) ⟶ j = k" (is "?P D")
  and "[M; map (λd. ⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st) D]_d I" (is "?Q D")
  shows "list_all (λd. fst d = i) D"

```

```
using assms
```

```
proof (induction D rule: List.rev_induct)
```

```

  case (snoc di D)
  obtain d j where di: "di = (j,d)" by (metis surj_pair)
  have "pair (t,s) · I = pair d · I" using di snoc.prems(2) by auto
  hence "∃ δ. Unifier δ (pair (t,s)) (pair d)" by auto
  hence 1: "i = j" using snoc.prems(1) di by fastforce

```

```

  have "set D ⊆ set (D@[di])" by auto
  hence 2: "?P D" using snoc.prems(1) by blast

```

```
have 3: "?Q D" using snoc.prems(2) by auto
```

```
show ?case using di 1 snoc.IH[OF 2 3] by simp
```

```
qed simp
```

```
lemma (in stateful_typed_model) labeled_sat_eqs_subseqs:
```

```

  assumes "Di ∈ set (subseqs D)"
  and "∀ (j, p) ∈ {(i,t,s)} ∪ set D. ∀ (k, q) ∈ {(i,t,s)} ∪ set D.
    (∃ δ. Unifier δ (pair p) (pair q)) ⟶ j = k" (is "?P D")
  and "[M; map (λd. ⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st) Di]_d I"
  shows "Di ∈ set (subseqs (dbproj i D))"

```

```
proof -
```

```

  have "set Di ⊆ set D" by (rule subseqs_subset[OF assms(1)])
  hence "?P Di" using assms(2) by blast

```

```
thus ?thesis using labeled_sat_eqs_list_all[OF _ assms(3)] subseqs_mem_dbproj[OF assms(1)] by simp
```

```
qed
```

```
lemma (in stateful_typed_model) duallsst_tfrsstp:
```

```

  assumes "list_all tfrsstp (unlabel S)"
  shows "list_all tfrsstp (unlabel (duallsst S))"

```

```
using assms
```

```
proof (induction S)
```

```

  case (Cons a S)
  have prems: "tfrsstp (snd a)" "list_all tfrsstp (unlabel S)"
  using Cons.prems unlabel_Cons(2)[of a S] by simp_all
  hence IH: "list_all tfrsstp (unlabel (duallsst S))" by (metis Cons.IH)

```

```
obtain l b where a: "a = (l,b)" by (metis surj_pair)
```

```
with Cons show ?case
```

```
proof (cases b)
```

```

  case (Equality c t t')
  hence "duallsst (a#S) = a#duallsst S" by (metis duallsst_Cons(3) a)
  thus ?thesis using a IH prems by fastforce

```

```

next
  case (NegChecks X F G)
  hence "duallsst (a#S) = a#duallsst S" by (metis duallsst_Cons(7) a)
  thus ?thesis using a IH prems by fastforce
qed auto
qed simp

lemma (in stateful_typed_model) setopssst_unlabel_duallsst_eq:
  "setopssst (unlabel (duallsst A)) = setopssst (unlabel A)"
proof (induction A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)
  thus ?case using Cons.IH by (cases b) (simp_all add: setopssst_def)
qed simp

```

## 6.2.2 Locale Setup and Definitions

```

locale labeled_stateful_typed_model =
  stateful_typed_model arity public Ana  $\Gamma$  Pair
+ labeled_typed_model arity public Ana  $\Gamma$  label_witness1 label_witness2
  for arity::"'fun  $\Rightarrow$  nat"
  and public::"'fun  $\Rightarrow$  bool"
  and Ana::"'(fun,'var) term  $\Rightarrow$  (('fun,'var) term list  $\times$  ('fun,'var) term list)"
  and  $\Gamma$ ::"'(fun,'var) term  $\Rightarrow$  ('fun,'atom::finite) term_type"
  and Pair::"'fun"
  and label_witness1::"'lbl"
  and label_witness2::"'lbl"
begin

definition lpair where
  "lpair lp  $\equiv$  case lp of (i,p)  $\Rightarrow$  (i,pair p)"

lemma setopslsstp_pair_image[simp]:
  "lpair ' (setopslsstp (i,send<t>)) = {}"
  "lpair ' (setopslsstp (i,receive<t>)) = {}"
  "lpair ' (setopslsstp (i,<ac: t  $\doteq$  t'>)) = {}"
  "lpair ' (setopslsstp (i,insert<t,s>)) = {(i, pair (t,s))}"
  "lpair ' (setopslsstp (i,delete<t,s>)) = {(i, pair (t,s))}"
  "lpair ' (setopslsstp (i,<ac: t  $\in$  s>)) = {(i, pair (t,s))}"
  "lpair ' (setopslsstp (i, $\forall X(\forall \neq: F \vee \notin: F')$ )) = (( $\lambda(t,s).$  (i, pair (t,s))) ' set F')"
```

unfolding lpair\_def by force+

```

definition par_complsst where
  "par_complsst (A::('fun,'var,'lbl) labeled_stateful_strand) (Secrets::('fun,'var) terms)  $\equiv$ 
    ( $\forall$  l1 l2. l1  $\neq$  l2  $\longrightarrow$ 
      GSMP_disjoint (trmssst (proj_unl l1 A)  $\cup$  pair ' setopssst (proj_unl l1 A))
        (trmssst (proj_unl l2 A)  $\cup$  pair ' setopssst (proj_unl l2 A)) Secrets)  $\wedge$ 
    ground Secrets  $\wedge$  ( $\forall s \in$  Secrets.  $\forall s' \in$  subterms s.  $\{s'\} \vdash_c s' \vee s' \in$  Secrets)  $\wedge$ 
    ( $\forall (i,p) \in$  setopslsst A.  $\forall (j,q) \in$  setopslsst A.
      ( $\exists \delta.$  Unifier  $\delta$  (pair p) (pair q))  $\longrightarrow$  i = j)"

definition declassifiedlsst where
  "declassifiedlsst A  $\mathcal{I} \equiv$  {t. <*, receive<t>>  $\in$  set A}  $\cdot_{set}$   $\mathcal{I}$ "

definition strand_leakslsst ("_ leaks _ under _") where
  "(A::('fun,'var,'lbl) labeled_stateful_strand) leaks Secrets under  $\mathcal{I} \equiv$ 
    ( $\exists t \in$  Secrets - declassifiedlsst A  $\mathcal{I}$ .  $\exists n.$   $\mathcal{I} \models_s$  (proj_unl n A@[send<t>]))"
```

```

definition typing_condsst where
  "typing_condsst A  $\equiv$  wfsst A  $\wedge$  wftrms (trmssst A)  $\wedge$  tfrsst A"

type_synonym ('a,'b,'c) labeleddbstate = "('c strand_label  $\times$  (('a,'b) term  $\times$  ('a,'b) term)) set"
type_synonym ('a,'b,'c) labeleddbstatelist = "('c strand_label  $\times$  (('a,'b) term  $\times$  ('a,'b) term))

```



list"

For proving the compositionality theorem for stateful constraints the idea is to first define a variant of the reduction technique that was used to establish the stateful typing result. This variant performs database-state projections, and it allows us to reduce the compositionality problem for stateful constraints to ordinary constraints.

```

fun trpc::
  "('fun,'var,'lbl) labeled_stateful_strand ⇒ ('fun,'var,'lbl) labeleddbstatelist
  ⇒ ('fun,'var,'lbl) labeled_strand list"
where
  "trpc [] D = [[]]"
  | "trpc ((i,send⟨t⟩)#A) D = map ((#) (i,send⟨t⟩st)) (trpc A D)"
  | "trpc ((i,receive⟨t⟩)#A) D = map ((#) (i,receive⟨t⟩st)) (trpc A D)"
  | "trpc ((i,⟨ac: t ≐ t'⟩)#A) D = map ((#) (i,⟨ac: t ≐ t'⟩st)) (trpc A D)"
  | "trpc ((i,insert⟨t,s⟩)#A) D = trpc A (List.insert (i,(t,s)) D)"
  | "trpc ((i,delete⟨t,s⟩)#A) D = (
    concat (map (λDi. map (λB. (map (λd. (i,⟨check: (pair (t,s)) ≐ (pair (snd d))⟩st)) Di)@
      (map (λd. (i,∀ []⟨V≠: [(pair (t,s), pair (snd d))⟩st))
        [d←dbproj i D. d ∉ set Di])@B)
      (trpc A [d←D. d ∉ set Di])))
    (subseqs (dbproj i D)))"
  | "trpc ((i,⟨ac: t ∈ s⟩)#A) D =
    concat (map (λB. map (λd. (i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#B) (dbproj i D)) (trpc A D))"
  | "trpc ((i,∀X⟨V≠: F V≠: F'⟩)#A) D =
    map ((@) (map (λG. (i,∀X⟨V≠: (F@G)⟩st)) (trpairs F' (map snd (dbproj i D))))) (trpc A D)"

```

### 6.2.3 Small Lemmata

```

lemma par_complsst_nil:
  assumes "ground Sec" "∀s ∈ Sec. ∀s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ Sec"
  shows "par_complsst [] Sec"
using assms unfolding par_complsst_def by simp

```

```

lemma par_complsst_subset:
  assumes A: "par_complsst A Sec"
  and BA: "set B ⊆ set A"
  shows "par_complsst B Sec"
proof -
  let ?L = "λn A. trmssst (proj_unl n A) ∪ pair ' setopssst (proj_unl n A)"
  have "?L n B ⊆ ?L n A" for n
    using trmssst_mono[OF proj_set_mono(2)[OF BA]] setopssst_mono[OF proj_set_mono(2)[OF BA]]
    by blast
  hence "GSMP_disjoint (?L m B) (?L n B) Sec" when nm: "m ≠ n" for n m::'lbl
    using GSMP_disjoint_subset[of "?L m A" "?L n A" Sec "?L m B" "?L n B"] A nm
    unfolding par_complsst_def by simp
  thus "par_complsst B Sec"
    using A setopslsst_mono[OF BA]
    unfolding par_complsst_def by blast
qed

```

```

lemma par_complsst_split:
  assumes "par_complsst (A@B) Sec"
  shows "par_complsst A Sec" "par_complsst B Sec"
using par_complsst_subset[OF assms] by simp_all

```

```

lemma par_complsst_proj:
  assumes "par_complsst A Sec"
  shows "par_complsst (proj n A) Sec"
using par_complsst_subset[OF assms] by simp

```

```

lemma par_complsst_duallsst:
  assumes A: "par_complsst A S"

```

```

shows "par_complsst (duallsst A) S"
proof (unfold par_complsst_def case_prod_unfold; intro conjI)
  show "ground S" "∀ s ∈ S. ∀ s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ S"
    using A unfolding par_complsst_def by fast+

  let ?M = "λ l B. (trmslsst (proj l B) ∪ pair ' setopssst (proj_unl l B))"
  let ?P = "λ B. ∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (?M l1 B) (?M l2 B) S"
  let ?Q = "λ B. ∀ p ∈ setopslsst B. ∀ q ∈ setopslsst B.
    (∃ δ. Unifier δ (pair (snd p)) (pair (snd q))) → fst p = fst q"

  have "?P A" "?Q A" using A unfolding par_complsst_def case_prod_unfold by blast+
  thus "?P (duallsst A)" "?Q (duallsst A)"
    by (metis setopssst_unlabel_duallsst_eq trmssst_unlabel_duallsst_eq proj_duallsst,
        metis setopslsst_duallsst_eq)
qed

lemma par_complsst_subst:
  assumes A: "par_complsst A S"
  and δ: "wtsubst δ" "wftrms (subst_range δ)" "subst_domain δ ∩ bvarslsst A = {}"
  shows "par_complsst (A ·lsst δ) S"
proof (unfold par_complsst_def case_prod_unfold; intro conjI)
  show "ground S" "∀ s ∈ S. ∀ s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ S"
    using A unfolding par_complsst_def by fast+

  let ?N = "λ l B. trmslsst (proj l B) ∪ pair ' setopssst (proj_unl l B)"
  define M where "M ≡ λ l (B::('fun,'var,'lbl) labeled_stateful_strand). ?N l B"
  let ?P = "λ p q. ∃ δ. Unifier δ (pair (snd p)) (pair (snd q))"
  let ?Q = "λ B. ∀ p ∈ setopslsst B. ∀ q ∈ setopslsst B. ?P p q → fst p = fst q"
  let ?R = "λ B. ∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (?N l1 B) (?N l2 B) S"

  have d: "bvarslsst (proj l A) ∩ subst_domain δ = {}" for l
    using δ(3) unfolding proj_def bvarssst_def unlabel_def by auto

  have "GSMP_disjoint (M l1 A) (M l2 A) S" when l: "l1 ≠ l2" for l1 l2
    using l A unfolding par_complsst_def M_def by presburger
  moreover have "M l (A ·lsst δ) = (M l A) ·set δ" for l
    using fun_pair_subst_set[of δ "setopssst (proj_unl l A)", symmetric]
      trmssst_subst[OF d[of l]] setopssst_subst[OF d[of l]] proj_subst[of l A δ]
    unfolding M_def unlabel_subst by auto
  ultimately have "GSMP_disjoint (M l1 (A ·lsst δ)) (M l2 (A ·lsst δ)) S" when l: "l1 ≠ l2" for l1 l2
    using l GSMP_wt_subst_subset[OF _ δ(1,2), of _ "M l1 A"]
      GSMP_wt_subst_subset[OF _ δ(1,2), of _ "M l2 A"]
    unfolding GSMP_disjoint_def by fastforce
  thus "?R (A ·lsst δ)" unfolding M_def by blast

  have "?Q A" using A unfolding par_complsst_def by force
  thus "?Q (A ·lsst δ)" using δ(3)
proof (induction A)
  case (Cons a A)
  obtain l b where a: "a = (l,b)" by (metis surj_pair)

  have 0: "bvarslsst (a#A) = set (bvarssstp (snd a)) ∪ bvarslsst A"
    unfolding bvarssst_def unlabel_def by simp

  have "?Q A" "subst_domain δ ∩ bvarslsst A = {}"
    using Cons.prem0 unfolding setopslsst_def by auto
  hence IH: "?Q (A ·lsst δ)" using Cons.IH unfolding setopslsst_def by blast

  have 1: "fst p = fst q"
    when p: "p ∈ setopslsstp (a ·lsstp δ)"
    and q: "q ∈ setopslsstp (a ·lsstp δ)"
    and pq: "?P p q"
    for p q

```

```

using a p q pq by (cases b) auto

have 2: "fst p = fst q"
  when p: "p ∈ setopslsst (A ·lsst δ)"
    and q: "q ∈ setopslsstp (a ·lsstp δ)"
    and pq: "?P p q"
  for p q
proof -
  obtain p' X where p':
    "p' ∈ setopslsst A" "fst p = fst p'"
    "X ⊆ bvarslsst (a#A)" "snd p = snd p' ·p rm_vars X δ"
    using setopslsst_in_subst[OF p] 0 by blast

  obtain q' Y where q':
    "q' ∈ setopslsstp a" "fst q = fst q'"
    "Y ⊆ bvarslsst (a#A)" "snd q = snd q' ·p rm_vars Y δ"
    using setopslsstp_in_subst[OF q] 0 by blast

  have "pair (snd p) = pair (snd p') · δ"
    "pair (snd q) = pair (snd q') · δ"
    using fun_pair_subst[of "snd p'" "rm_vars X δ"] fun_pair_subst[of "snd q'" "rm_vars Y δ"]
      p'(3,4) q'(3,4) Cons.prems(2) rm_vars_apply'[of δ X] rm_vars_apply'[of δ Y]
    by fastforce+
  hence "∃ δ. Unifier δ (pair (snd p')) (pair (snd q'))"
    using pq Unifier_comp' by metis
  thus ?thesis using Cons.prems p'(1,2) q'(1,2) by simp
qed

show ?case by (metis 1 2 IH Un_iff setopslsst_cons subst_lsst_cons)
qed simp
qed

lemma wf_pair_negchecks_map':
  assumes "wfst X (unlabel A)"
  shows "wfst X (unlabel (map (λG. (i, ∀ Y (V ≠: (F@G))st)) M@A))"
  using assms by (induct M) auto

lemma wf_pair_eqs_ineqs_map':
  fixes A::('fun, 'var, 'lbl) labeled_strand"
  assumes "wfst X (unlabel A)"
    "Di ∈ set (subseqs (dbproj i D))"
    "fvpairs (unlabel D) ⊆ X"
  shows "wfst X (unlabel (
    (map (λd. (i, ⟨check: (pair (t,s)) ÷ (pair (snd d))⟩st)) Di)@
    (map (λd. (i, ∀ [] (V ≠: [(pair (t,s), pair (snd d))]st)) [d ← dbproj i D. d ∉ set Di])@A))"

proof -
  let ?f = "[d ← dbproj i D. d ∉ set Di]"
  define c1 where c1: "c1 = map (λd. (i, ⟨check: (pair (t,s)) ÷ (pair (snd d))⟩st)) Di"
  define c2 where c2: "c2 = map (λd. (i, ∀ [] (V ≠: [(pair (t,s), pair (snd d))]st)) ?f)"
  define c3 where c3: "c3 = map (λd. ⟨check: (pair (t,s)) ÷ (pair d)⟩st (unlabel Di))"
  define c4 where c4: "c4 = map (λd. ∀ [] (V ≠: [(pair (t,s), pair d)]st (unlabel ?f))"
  have ci_eqs: "c3 = unlabel c1" "c4 = unlabel c2" unfolding c1 c2 c3 c4 unlabel_def by auto
  have 1: "wfst X (unlabel (c2@A))"
    using wf_fun_pair_ineqs_map[OF assms(1)] ci_eqs(2) unlabel_append[of c2 A] c4
    by metis
  have 2: "fvpairs (unlabel Di) ⊆ X"
    using assms(3) subseqs_set_subset(1)[OF assms(2)]
    unfolding unlabel_def
    by fastforce
  { fix B::('fun, 'var) strand" assume "wfst X B"
    hence "wfst X (unlabel c1@B)" using 2 unfolding c1 unlabel_def by (induct Di) auto
  } thus ?thesis using 1 unfolding c1 c2 unlabel_def by simp
qed

```

```

lemma trmssst_setopssst_wt_instance_ex:
  defines "M  $\equiv$   $\lambda A$ . trmslsst A  $\cup$  pair ' setopssst (unlabel A) "
  assumes B: " $\forall b \in \text{set } B. \exists a \in \text{set } A. \exists \delta. b = a \cdot_{lsstp} \delta \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta)$ "
  shows " $\forall t \in M B. \exists s \in M A. \exists \delta. t = s \cdot \delta \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta)$ "
proof
  let ?P = " $\lambda \delta. \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta)$ "

  fix t assume "t  $\in M B$ "
  then obtain b where b: "b  $\in \text{set } B$ " "t  $\in \text{trms}_{sstp} (\text{snd } b) \cup \text{pair ' setops}_{sstp} (\text{snd } b)$ "
    unfolding M_def unfolding unlabel_def trmssst_def setopssst_def by auto
  then obtain a  $\delta$  where a: "a  $\in \text{set } A$ " "b = a  $\cdot_{lsstp} \delta$ " and  $\delta$ : " $\text{wt}_{subst} \delta$ " " $\text{wf}_{trms} (\text{subst\_range } \delta)$ "
    using B by meson

  note  $\delta' = \text{wt\_subst\_rm\_vars}[OF \delta(1)] \text{ wf\_trms\_subst\_rm\_vars}'[OF \delta(2)]$ 

  have "t  $\in M (A \cdot_{lsst} \delta)$ "
    using b(2) a
    unfolding M_def subst_apply_labeled_stateful_strand_def unlabel_def trmssst_def setopssst_def
    by auto
  moreover have " $\exists s \in M A. \exists \delta. t = s \cdot \delta \wedge ?P \delta$ " when "t  $\in \text{trms}_{lsst} (A \cdot_{lsst} \delta)$ "
    using trmssst_unlabel_subst'[OF that]  $\delta'$  unfolding M_def by blast
  moreover have " $\exists s \in M A. \exists \delta. t = s \cdot \delta \wedge ?P \delta$ " when t: "t  $\in \text{pair ' setops}_{sst} (\text{unlabel } A \cdot_{sst} \delta)$ "
  proof -
    obtain p where p: "p  $\in \text{setops}_{sst} (\text{unlabel } A \cdot_{sst} \delta)$ " "t = pair p" using t by blast
    then obtain q X where q: "q  $\in \text{setops}_{sst} (\text{unlabel } A)$ " "p = q  $\cdot_p \text{rm\_vars} (\text{set } X) \delta$ "
      using setopssst_subst'[OF p(1)] by blast
    hence "t = pair q  $\cdot \text{rm\_vars} (\text{set } X) \delta$ "
      using fun_pair_subst[of q "rm_vars (set X)  $\delta$ "] p(2) by presburger
    thus ?thesis using  $\delta'$ [of "set X"] q(1) unfolding M_def by blast
  qed
  ultimately show " $\exists s \in M A. \exists \delta. t = s \cdot \delta \wedge ?P \delta$ " unfolding M_def unlabel_subst by fast
qed

lemma setopslsst_wt_instance_ex:
  assumes B: " $\forall b \in \text{set } B. \exists a \in \text{set } A. \exists \delta. b = a \cdot_{lsstp} \delta \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta)$ "
  shows " $\forall p \in \text{setops}_{lsst} B. \exists q \in \text{setops}_{lsst} A. \exists \delta. \text{fst } p = \text{fst } q \wedge \text{snd } p = \text{snd } q \cdot_p \delta \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta)$ "
proof
  let ?P = " $\lambda \delta. \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta)$ "

  fix p assume "p  $\in \text{setops}_{lsst} B$ "
  then obtain b where b: "b  $\in \text{set } B$ " "p  $\in \text{setops}_{lsstp} b$ " unfolding setopslsst_def by blast
  then obtain a  $\delta$  where a: "a  $\in \text{set } A$ " "b = a  $\cdot_{lsstp} \delta$ " and  $\delta$ : " $\text{wt}_{subst} \delta$ " " $\text{wf}_{trms} (\text{subst\_range } \delta)$ "
    using B by meson
  hence p: "p  $\in \text{setops}_{lsst} (A \cdot_{lsst} \delta)$ "
    using b(2) unfolding setopslsst_def subst_apply_labeled_stateful_strand_def by auto

  obtain X q where q:
    "q  $\in \text{setops}_{lsst} A$ " "fst p = fst q" "snd p = snd q  $\cdot_p \text{rm\_vars } X \delta$ "
    using setopslsst_in_subst[OF p] by blast

  show " $\exists q \in \text{setops}_{lsst} A. \exists \delta. \text{fst } p = \text{fst } q \wedge \text{snd } p = \text{snd } q \cdot_p \delta \wedge ?P \delta$ "
    using q wt_subst_rm_vars[OF  $\delta(1)$ ] wf_trms_subst_rm_vars'[OF  $\delta(2)$ ] by blast
qed

```

#### 6.2.4 Lemmata: Properties of the Constraint Translation Function

```

lemma tr_par_labeled_rcv_iff:
  "B  $\in \text{set } (\text{tr}_{pc} A D) \implies (i, \text{receive}(t)_{st}) \in \text{set } B \longleftrightarrow (i, \text{receive}(t)) \in \text{set } A$ "
by (induct A D arbitrary: B rule: trpc.induct) auto

```

```

lemma tr_par_declassified_eq:

```

```

"B ∈ set (trpc A D) ⇒ declassifiedlst B I = declassifiedlst A I"
using tr_par_labeled_rcv_iff unfolding declassifiedlst_def declassifiedlst_def by simp

lemma tr_par_ik_eq:
  assumes "B ∈ set (trpc A D)"
  shows "ikst (unlabel B) = ikst (unlabel A)"
proof -
  have "{t. ∃ i. (i, receive⟨t⟩st) ∈ set B} = {t. ∃ i. (i, receive⟨t⟩) ∈ set A}"
    using tr_par_labeled_rcv_iff[OF assms] by simp
  moreover have
    "∧ C. {t. ∃ i. (i, receive⟨t⟩st) ∈ set C} = {t. receive⟨t⟩st ∈ set (unlabel C)}"
    "∧ C. {t. ∃ i. (i, receive⟨t⟩) ∈ set C} = {t. receive⟨t⟩ ∈ set (unlabel C)}"
  unfolding unlabel_def by force+
  ultimately show ?thesis by (metis ikst_def ikst_is_rcv_set)
qed

lemma tr_par_deduct_iff:
  assumes "B ∈ set (trpc A D)"
  shows "ikst (unlabel B) ·set I ⊢ t ↔ ikst (unlabel A) ·set I ⊢ t"
using tr_par_ik_eq[OF assms] by metis

lemma tr_par_vars_subset:
  assumes "A' ∈ set (trpc A D)"
  shows "fvlst A' ⊆ fvsst (unlabel A) ∪ fvpairs (unlabel D)" (is ?P)
  and "bvarslst A' ⊆ bvarssst (unlabel A)" (is ?Q)
proof -
  show ?P using assms
proof (induction "unlabel A" arbitrary: A A' D rule: strand_sem_stateful_induct)
  case (ConsIn A' D ac t s AA A')
  then obtain i B where iB: "A = (i, ⟨ac: t ∈ s⟩)#B" "AA = unlabel B"
    unfolding unlabel_def by maura
  then obtain A'' d where *:
    "d ∈ set (dbproj i D)"
    "A' = (i, ⟨ac: (pair (t,s)) ÷ (pair (snd d))⟩st)#A'"
    "A'' ∈ set (trpc B D)"
  using ConsIn.premis(1) by maura
  hence "fvlst A'' ⊆ fvsst (unlabel B) ∪ fvpairs (unlabel D)"
    "fv (pair (snd d)) ⊆ fvpairs (unlabel D)"
  apply (metis ConsIn.hyps(1)[OF iB(2)])
  using fvpairs_mono[OF dbproj_subset[of i D]]
    fv_pair_fvpairs_subset[OF *(1)]
  by blast
  thus ?case using * iB unfolding pair_def by auto
next
  case (ConsDel A' D t s AA A')
  then obtain i B where iB: "A = (i, delete⟨t,s⟩)#B" "AA = unlabel B"
    unfolding unlabel_def by maura

define fltD1 where "fltD1 = (λDi. filter (λd. d ∉ set Di) D)"
define fltD2 where "fltD2 = (λDi. filter (λd. d ∉ set Di) (dbproj i D))"
define constr where "constr =
  (λDi. (map (λd. (i, ⟨check: (pair (t,s)) ÷ (pair (snd d))⟩st)) Di)@
    (map (λd. (i, ∀ [] [V≠: [(pair (t,s), pair (snd d))]st)) (fltD2 Di)))"

from iB obtain A'' Di where *:
  "Di ∈ set (subseqs (dbproj i D))" "A' = (constr Di)@A'" "A'' ∈ set (trpc B (fltD1 Di))"
  using ConsDel.premis(1) unfolding constr_def fltD1_def fltD2_def by maura
  hence "fvlst A'' ⊆ fvsst AA ∪ fvpairs (unlabel (fltD1 Di))"
  unfolding constr_def fltD1_def by (metis ConsDel.hyps(1) iB(2))
  hence 1: "fvlst A'' ⊆ fvsst AA ∪ fvpairs (unlabel D)"
  using fvpairs_mono[of "unlabel (fltD1 Di)" "unlabel D"]
  unfolding unlabel_def fltD1_def by force

```

```

have 2: "fv_pairs (unlabel Di) ∪ fv_pairs (unlabel (fltD1 Di)) ⊆ fv_pairs (unlabel D)"
  using subseqs_set_subset(1)[OF *(1)]
  unfolding fltD1_def unlabel_def
  by auto

have 5: "fv_lst A' = fv_lst (constr Di) ∪ fv_lst A'" using * unfolding unlabel_def by force

have "fv_lst (constr Di) ⊆ fv t ∪ fv s ∪ fv_pairs (unlabel Di) ∪ fv_pairs (unlabel (fltD1 Di))"
  unfolding unlabel_def constr_def fltD1_def fltD2_def pair_def by auto
hence 3: "fv_lst (constr Di) ⊆ fv t ∪ fv s ∪ fv_pairs (unlabel D)" using 2 by blast

have 4: "fv_sst (unlabel A) = fv t ∪ fv s ∪ fv_sst AA" using iB by auto

have "fv_st (unlabel A') ⊆ fv_sst (unlabel A) ∪ fv_pairs (unlabel D)" using 1 3 4 5 by blast
thus ?case by metis
next
case (ConsNegChecks A' D X F F' AA A')
then obtain i B where iB: "A = (i, NegChecks X F F')#B" "AA = unlabel B"
  unfolding unlabel_def by maura

define D' where "D' ≡ ⋃ (fv_pairs ' set (tr_pairs F' (unlabel (dbproj i D))))"
define constr where "constr = map (λG. (i, ∀X(≠: (F@G)<_st)) (tr_pairs F' (map snd (dbproj i D))))"

from iB obtain A'' where *: "A'' ∈ set (tr_pc B D)" "A' = constr@A'"
  using ConsNegChecks.prem(1) unfolding constr_def by maura
hence "fv_lst A'' ⊆ fv_sst AA ∪ fv_pairs (unlabel D)"
  by (metis ConsNegChecks.hyps(1) iB(2))
hence **: "fv_lst A'' ⊆ fv_sst AA ∪ fv_pairs (unlabel D)" by auto

have 1: "fv_lst constr ⊆ (D' ∪ fv_pairs F) - set X"
  unfolding D'_def constr_def unlabel_def by auto

have "set (unlabel (dbproj i D)) ⊆ set (unlabel D)" unfolding unlabel_def by auto
hence 2: "D' ⊆ fv_pairs F' ∪ fv_pairs (unlabel D)"
  using tr_pairs_vars_subset'[of F' "unlabel (dbproj i D)"] fv_pairs_mono
  unfolding D'_def by blast

have 3: "fv_lst A' ⊆ ((fv_pairs F' ∪ fv_pairs F) - set X) ∪ fv_pairs (unlabel D) ∪ fv_lst A'"
  using 1 2 *(2) unfolding unlabel_def by fastforce

have 4: "fv_sst AA ⊆ fv_sst (unlabel A)" by (metis ConsNegChecks.hyps(2) fv_sst_cons_subset)

have 5: "fv_pairs F' ∪ fv_pairs F - set X ⊆ fv_sst (unlabel A)"
  using ConsNegChecks.hyps(2) unfolding unlabel_def by force

show ?case using ** 3 4 5 by blast
qed (fastforce simp add: unlabel_def)+

show ?Q using assms
  apply (induct "unlabel A" arbitrary: A A' D rule: strand_sem_stateful_induct)
  by (fastforce simp add: unlabel_def)+
qed

lemma tr_par_vars_disj:
  assumes "A' ∈ set (tr_pc A D)" "fv_pairs (unlabel D) ∩ bvars_sst (unlabel A) = {}"
  and "fv_sst (unlabel A) ∩ bvars_sst (unlabel A) = {}"
  shows "fv_lst A' ∩ bvars_lst A' = {}"
using assms tr_par_vars_subset by fast

lemma tr_par_trms_subset:
  assumes "A' ∈ set (tr_pc A D)"
  shows "trms_lst A' ⊆ trms_sst (unlabel A) ∪ pair ' setops_sst (unlabel A) ∪ pair ' snd ' set D"
using assms

```

```

proof (induction A D arbitrary: A' rule: trpc.induct)
  case 1 thus ?case by simp
next
  case (2 i t A D)
  then obtain A'' where A'': "A' = (i, send⟨t⟩st)#A'" "A'' ∈ set (trpc A D)" by moura
  hence "trmslst A'' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪ pair ' snd ' set D"
    by (metis "2.IH")
  thus ?case using A'' by (auto simp add: setopssst_def)
next
  case (3 i t A D)
  then obtain A'' where A'': "A' = (i, receive⟨t⟩st)#A'" "A'' ∈ set (trpc A D)"
    by moura
  hence "trmslst A'' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪ pair ' snd ' set D"
    by (metis "3.IH")
  thus ?case using A'' by (auto simp add: setopssst_def)
next
  case (4 i ac t t' A D)
  then obtain A'' where A'': "A' = (i, ⟨ac: t ≐ t'⟩st)#A'" "A'' ∈ set (trpc A D)"
    by moura
  hence "trmslst A'' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪ pair ' snd ' set D"
    by (metis "4.IH")
  thus ?case using A'' by (auto simp add: setopssst_def)
next
  case (5 i t s A D)
  hence "A' ∈ set (trpc A (List.insert (i, t, s) D))" by simp
  hence "trmslst A' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪
    pair ' snd ' set (List.insert (i, t, s) D)"
    by (metis "5.IH")
  thus ?case by (auto simp add: setopssst_def)
next
  case (6 i t s A D)
  from 6 obtain Di A'' B C where A'':
    "Di ∈ set (subseqs (dbproj i D))" "A'' ∈ set (trpc A [d ← D. d ∉ set Di])" "A' = (B@C)@A'"
    "B = map (λd. (i, ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st)) Di"
    "C = map (λd. (i, ∀ [] (V ≠: [(pair (t, s), pair (snd d))]st)) [d ← dbproj i D. d ∉ set Di])"
    by moura
  hence "trmslst A'' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪
    pair ' snd ' set [d ← D. d ∉ set Di]"
    by (metis "6.IH")
  moreover have "set [d ← D. d ∉ set Di] ⊆ set D" using set_filter by auto
  ultimately have
    "trmslst A'' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪ pair ' snd ' set D"
    by blast
  hence "trmslst A'' ⊆ trmssst (unlabel ((i, delete⟨t, s⟩)#A)) ∪
    pair ' setopssst (unlabel ((i, delete⟨t, s⟩)#A)) ∪
    pair ' snd ' set D"
    using setopssst_cons_subset trmssst_cons
    by (auto simp add: setopssst_def)
  moreover have "set Di ⊆ set D" "set [d ← dbproj i D. d ∉ set Di] ⊆ set D"
    using subseqs_set_subset[OF A''(1)] by auto
  hence "trmsst (unlabel B) ⊆ insert (pair (t, s)) (pair ' snd ' set D)"
    "trmsst (unlabel C) ⊆ insert (pair (t, s)) (pair ' snd ' set D)"
    using A''(4,5) unfolding unlabel_def by auto
  hence "trmsst (unlabel (B@C)) ⊆ insert (pair (t, s)) (pair ' snd ' set D)"
    using unlabel_append[of B C] by auto
  moreover have "pair (t, s) ∈ pair ' setopssst (delete⟨t, s⟩#unlabel A)" by (simp add: setopssst_def)
  ultimately show ?case
    using A''(3) trmsst_append[of "unlabel (B@C)" "unlabel A'"] unlabel_append[of "B@C" A'']
    by (auto simp add: setopssst_def)
next
  case (7 i ac t s A D)
  from 7 obtain d A'' where A'':
    "d ∈ set (dbproj i D)" "A'' ∈ set (trpc A D)"

```

```

    "A' = (i, ⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#A'''"
  by maura
hence "trmslst A'' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪
      pair ' snd ' set D"
  by (metis "7.IH")
moreover have "trmsst (unlabel A') = {pair (t,s), pair (snd d)} ∪ trmsst (unlabel A'')"
  using A''(1,3) by auto
ultimately show ?case using A''(1) by (auto simp add: setopssst_def)
next
case (8 i X F F' A D)
define constr where "constr = map (λG. (i, ∀X⟨V≠: (F@G)⟩st)) (trpairs F' (map snd (dbproj i D)))"
define B where "B ≡ ⋃ (trmspairs ' set (trpairs F' (map snd (dbproj i D))))"

from 8 obtain A'' where A'':
  "A'' ∈ set (trpc A D)" "A' = constr@A''"
  unfolding constr_def by maura

have "trmsst (unlabel A'') ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A) ∪ pair'snd'set D"
  by (metis A''(1) "8.IH")
moreover have "trmsst (unlabel constr) ⊆ B ∪ trmspairs F ∪ pair ' snd ' set D"
  unfolding unlabel_def constr_def B_def by auto
ultimately have "trmsst (unlabel A') ⊆ B ∪ trmspairs F ∪ trmssst (unlabel A) ∪
      pair ' setopssst (unlabel A) ∪ pair ' snd ' set D"
  using A'' unlabel_append[of constr A''] by auto
moreover have "set (dbproj i D) ⊆ set D" by auto
hence "B ⊆ pair ' set F' ∪ pair ' snd ' set D"
  using trpairs-trms_subset'[of F' "map snd (dbproj i D)"]
  unfolding B_def by force
moreover have
  "pair ' setopssst (unlabel ((i, ∀X⟨V≠: F ∨ ∉: F')#A)) =
   pair ' set F' ∪ pair ' setopssst (unlabel A)"
  by auto
ultimately show ?case by (auto simp add: setopssst_def)
qed

lemma tr_par_wf_trms:
  assumes "A' ∈ set (trpc A [])" "wftrms (trmssst (unlabel A))"
  shows "wftrms (trmslst A'')"
using tr_par_trms_subset[OF assms(1)] setopssst-wftrms(2)[OF assms(2)]
by auto

lemma tr_par_wf':
  assumes "fvpairs (unlabel D) ∩ bvarssst (unlabel A) = {}"
  and "fvpairs (unlabel D) ⊆ X"
  and "wf'sst X (unlabel A)" "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}"
  and "A' ∈ set (trpc A D)"
  shows "wflst X A'"
proof -
  define P where
    "P = (λ(D::('fun,'var,'lbl) labeleddbstatelist) (A::('fun,'var,'lbl) labeled_stateful_strand).
      (fvpairs (unlabel D) ∩ bvarssst (unlabel A) = {}) ∧
      fvsst (unlabel A) ∩ bvarssst (unlabel A) = {})"

  have "P D A" using assms(1,4) by (simp add: P_def)
  with assms(5,3,2) show ?thesis
  proof (induction A arbitrary: X A' D)
    case Nil thus ?case by simp
  next
    case (Cons a A)
    obtain i s where i: "a = (i,s)" by (metis surj_pair)
    note prems = Cons.prems
    note IH = Cons.IH
    show ?case

```



```

proof (cases s)
  case (Receive t)
    note si = Receive i
    then obtain A'' where A'': "A' = (i, receive⟨t⟩st)#A''" "A'' ∈ set (trpc A D)" "fv t ⊆ X"
      using prems unlabel_Cons(1)[of i s A] by moura
    have *: "wf'sst X (unlabel A)"
      "fvpairs (unlabel D) ⊆ X"
      "P D A"
      using prems si apply (force, force)
      using prems(4) si unfolding P_def by fastforce
    show ?thesis using IH[OF A''(2) *] A''(1,3) by simp
next
  case (Send t)
    note si = Send i
    then obtain A'' where A'': "A' = (i, send⟨t⟩st)#A''" "A'' ∈ set (trpc A D)"
      using prems by moura
    have *: "wf'sst (X ∪ fv t) (unlabel A)"
      "fvpairs (unlabel D) ⊆ X ∪ fv t"
      "P D A"
      using prems si apply (force, force)
      using prems(4) si unfolding P_def by fastforce
    show ?thesis using IH[OF A''(2) *] A''(1) by simp
next
  case (Equality ac t t')
    note si = Equality i
    then obtain A'' where A'':
      "A' = (i, ⟨ac: t ≐ t'⟩st)#A''" "A'' ∈ set (trpc A D)"
      "ac = Assign ⟹ fv t' ⊆ X"
      using prems unlabel_Cons(1)[of i s] by moura
    have *: "ac = Assign ⟹ wf'sst (X ∪ fv t) (unlabel A)"
      "ac = Check ⟹ wf'sst X (unlabel A)"
      "ac = Assign ⟹ fvpairs (unlabel D) ⊆ X ∪ fv t"
      "ac = Check ⟹ fvpairs (unlabel D) ⊆ X"
      "P D A"
      using prems si apply (force, force, force, force)
      using prems(4) si unfolding P_def by fastforce
    show ?thesis
      using IH[OF A''(2) *(1,3,5)] IH[OF A''(2) *(2,4,5)] A''(1,3)
      by (cases ac) simp_all
next
  case (Insert t t')
    note si = Insert i
    hence A': "A' ∈ set (trpc A (List.insert (i, t, t') D))" "fv t ⊆ X" "fv t' ⊆ X"
      using prems by auto
    have *: "wf'sst X (unlabel A)" "fvpairs (unlabel (List.insert (i, t, t') D)) ⊆ X"
      using prems si by (auto simp add: unlabel_def)
    have **: "P (List.insert (i, t, t') D) A"
      using prems(4) si
      unfolding P_def unlabel_def
      by fastforce
    show ?thesis using IH[OF A'(1) * **] A'(2,3) by simp
next
  case (Delete t t')
    note si = Delete i
    define constr where "constr = (λDi.
      (map (λd. (i, ⟨check: (pair (t, t')) ≐ (pair (snd d))⟩st)) Di)@
      (map (λd. (i, ∀ [] ⟨≠: [(pair (t, t'), pair (snd d))]⟩st)) [d ← dbproj i D. d ∉ set Di]))"
    from prems si obtain Di A'' where A'':
      "A' = constr Di@A''" "A'' ∈ set (trpc A [d ← D. d ∉ set Di])"
      "Di ∈ set (subseqs (dbproj i D))"
      unfolding constr_def by auto
    have *: "wf'sst X (unlabel A)"
      "fvpairs (unlabel (filter (λd. d ∉ set Di) D)) ⊆ X"

```

```

using prems si apply simp
using prems si by (fastforce simp add: unlabel_def)

have "fv_pairs (unlabel (filter (λd. d ∉ set Di) D)) ⊆ fv_pairs (unlabel D)"
  by (auto simp add: unlabel_def)
hence **: "P [d ← D. d ∉ set Di] A"
  using prems si unfolding P_def
  by fastforce

have ***: "fv_pairs (unlabel D) ⊆ X" using prems si by auto
show ?thesis
  using IH[OF A''(2) * **] A''(1) wf_pair_eqs_ineqs_map'[OF _ A''(3) ***]
  unfolding constr_def by simp
next
case (InSet ac t t')
note si = InSet i
then obtain d A'' where A'':
  "A' = (i, ⟨ac: (pair (t, t')) ÷ (pair (snd d))⟩st)#A'"
  "A'' ∈ set (trpc A D)"
  "d ∈ set D"
  using prems by maura
have *:
  "ac = Assign ⇒ wf'sst (X ∪ fv t ∪ fv t') (unlabel A)"
  "ac = Check ⇒ wf'sst X (unlabel A)"
  "ac = Assign ⇒ fv_pairs (unlabel D) ⊆ X ∪ fv t ∪ fv t'"
  "ac = Check ⇒ fv_pairs (unlabel D) ⊆ X"
  "P D A"
  using prems si apply (force, force, force, force)
  using prems(4) si unfolding P_def by fastforce
have **: "fv (pair (snd d)) ⊆ X"
  using A''(3) prems(3) fv_pair_fv_pairs_subset
  by fast
have ***: "fv (pair (t, t')) = fv t ∪ fv t'" unfolding pair_def by auto
show ?thesis
  using IH[OF A''(2) *(1,3,5)] IH[OF A''(2) *(2,4,5)] A''(1) ** ***
  by (cases ac) (simp_all add: Un_assoc)
next
case (NegChecks Y F F')
note si = NegChecks i
then obtain A'' where A'':
  "A' = (map (λG. (i, ∀Y⟨V≠: (F@G)⟩st)) (tr_pairs F' (map snd (dbproj i D))))@A'"
  "A'' ∈ set (trpc A D)"
  using prems by maura

have *: "wf'sst X (unlabel A)" "fv_pairs (unlabel D) ⊆ X" using prems si by auto

have "bvarssst (unlabel A) ⊆ bvarssst (unlabel ((i, ∀Y⟨V≠: F ∨ ∉: F')⟩#A))"
  "fvsst (unlabel A) ⊆ fvsst (unlabel ((i, ∀Y⟨V≠: F ∨ ∉: F')⟩#A))"
  by auto
have **: "P D A" using prems si unfolding P_def by blast

show ?thesis using IH[OF A''(2) * **] A''(1) wf_pair_negchecks_map' by simp
qed
qed
qed

lemma tr_par_wf:
  assumes "A' ∈ set (trpc A [])"
  and "wfsst (unlabel A)"
  and "wftrms (trmslst A)"
  shows "wflst {} A'"
  and "wftrms (trmslst A')"
  and "fvlst A' ∩ bvarslst A' = {}"

```

```

using tr_par_wf'[OF _ _ _ assms(1)]
  tr_par_wf_trms[OF assms(1,3)]
  tr_par_vars_disj[OF assms(1)]
  assms(2)
by fastforce+

lemma tr_par_tfrsstp:
  assumes "A' ∈ set (trpc A D)" "list_all tfrsstp (unlabel A)"
  and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}" (is "?P0 A D")
  and "fvpairs (unlabel D) ∩ bvarssst (unlabel A) = {}" (is "?P1 A D")
  and "∀t ∈ pair ' setopssst (unlabel A) ∪ pair ' snd ' set D.
    ∀t' ∈ pair ' setopssst (unlabel A) ∪ pair ' snd ' set D.
      (∃δ. Unifier δ t t') ⟶ Γ t = Γ t'" (is "?P3 A D")
  shows "list_all tfrsstp (unlabel A)"
proof -
  have sublm: "list_all tfrsstp (unlabel A)" "?P0 A D" "?P1 A D" "?P3 A D"
  when p: "list_all tfrsstp (unlabel (a#A))" "?P0 (a#A) D" "?P1 (a#A) D" "?P3 (a#A) D"
  for a A D
  proof -
    show "list_all tfrsstp (unlabel A)" using p(1) by (simp add: unlabel_def tfrsstp_def)
    show "?P0 A D" using p(2) fvsst_cons_subset unfolding unlabel_def by fastforce
    show "?P1 A D" using p(3) bvarssst_cons_subset unfolding unlabel_def by fastforce
    have "setopssst (unlabel A) ⊆ setopssst (unlabel (a#A))"
      using setopssst_cons_subset unfolding unlabel_def by auto
    thus "?P3 A D" using p(4) by blast
  qed

show ?thesis using assms
proof (induction A D arbitrary: A' rule: trpc.induct)
  case 1 thus ?case by simp
next
  case (2 i t A D)
  note prems = "2.prems"
  note IH = "2.IH"
  from prems(1) obtain A'' where A'': "A' = (i, send⟨t⟩st)#A'" "A'' ∈ set (trpc A D)" by moura
  have "list_all tfrsstp (unlabel A'')"
    using IH[OF A''(2)] prems(5) sublm[OF prems(2,3,4,5)]
    by meson
  thus ?case using A''(1) by simp
next
  case (3 i t A D)
  note prems = "3.prems"
  note IH = "3.IH"
  from prems(1) obtain A'' where A'': "A' = (i, receive⟨t⟩st)#A'" "A'' ∈ set (trpc A D)" by moura
  have "list_all tfrsstp (unlabel A'')"
    using IH[OF A''(2)] prems(5) sublm[OF prems(2,3,4,5)]
    by meson
  thus ?case using A''(1) by simp
next
  case (4 i ac t t' A D)
  note prems = "4.prems"
  note IH = "4.IH"
  from prems(1) obtain A'' where A'': "A' = (i, ⟨ac: t ≐ t'⟩st)#A'" "A'' ∈ set (trpc A D)" by
moura
  have "list_all tfrsstp (unlabel A'')"
    using IH[OF A''(2)] prems(5) sublm[OF prems(2,3,4,5)]
    by meson
  thus ?case using A''(1) prems(2) by simp
next
  case (5 i t s A D)
  note prems = "5.prems"
  note IH = "5.IH"
  from prems(1) have A': "A' ∈ set (trpc A (List.insert (i,t,s) D))" by simp

```

```

have 1: "list_all tfrsstp (unlabel A)" using sublm[OF prems(2,3,4,5)] by simp

have "pair ' setopssst (unlabel ((i,insert⟨t,s⟩)#A)) ∪ pair'snd'set D =
  pair ' setopssst (unlabel A) ∪ pair'snd'set (List.insert (i,t,s) D)"
  by (auto simp add: setopssst_def)
hence 3: "?P3 A (List.insert (i,t,s) D)" using prems(5) by metis
moreover have "?P1 A (List.insert (i,t,s) D)"
  using prems(3,4) bvarssst_cons_subset[of "unlabel A" "insert⟨t,s⟩"]
  unfolding unlabel_def
  by fastforce
ultimately have "list_all tfrstp (unlabel A)"
  using IH[OF A' sublm(1,2)[OF prems(2,3,4,5)] _ 3] by metis
thus ?case using A'(1) by auto
next
case (6 i t s A D)
note prems = "6.prems"
note IH = "6.IH"

define constr where constr: "constr ≡ (λDi.
  (map (λd. (i,⟨check: (pair (t,s)) ≐ (pair (snd d))⟩st)) Di)@
  (map (λd. (i,⟨∇ [] ∇≠: [(pair (t,s), pair (snd d))]⟩st)) (filter (λd. d ∉ set Di) (dbproj i
D)))))"

from prems(1) obtain Di A'' where A'':
  "A' = constr Di@A''" "A'' ∈ set (trpc A (filter (λd. d ∉ set Di) D))"
  "Di ∈ set (subseqs (dbproj i D))"
  unfolding constr by fastforce

define Q1 where "Q1 ≡ (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀x ∈ (fvpairs F) - set X. ∃a. Γ (Var x) = TAtom a)"
define Q2 where "Q2 ≡ (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀f T. Fun f T ∈ subtermsset (trmspairs F) ⟶ T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X))"

have "pair ' setopssst (unlabel A) ∪ pair'snd'set [d←D. d ∉ set Di]
  ⊆ pair ' setopssst (unlabel ((i,delete⟨t,s⟩)#A)) ∪ pair'snd'set D"
  using subseqs_set_subset[OF A''(3)] by (force simp add: setopssst_def)
moreover have "∀a∈M. ∀b∈M. P a b"
  when "M ⊆ N" "∀a∈N. ∀b∈N. P a b"
  for M N:: "('fun, 'var) terms" and P
  using that by blast
ultimately have *: "?P3 A (filter (λd. d ∉ set Di) D)"
  using prems(5) by presburger

have **: "?P1 A (filter (λd. d ∉ set Di) D)"
  using prems(4) bvarssst_cons_subset[of "unlabel A" "delete⟨t,s⟩"]
  unfolding unlabel_def by fastforce

have 1: "list_all tfrstp (unlabel A'')"
  using IH[OF A''(3,2) sublm(1,2)[OF prems(2,3,4,5)] ** *]
  by metis

have 2: "⟨ac: u ≐ u'⟩st ∈ set (unlabel A'') ∨
  (∃d ∈ set Di. u = pair (t,s) ∧ u' = pair (snd d))"
  when "⟨ac: u ≐ u'⟩st ∈ set (unlabel A'')" for ac u u'
  using that A''(1) unfolding constr unlabel_def by force
have 3:
  "∀X⟨∇≠: u⟩st ∈ set (unlabel A'') ∨
  (∃d ∈ set (filter (λd. d ∉ set Di) D). u = [(pair (t,s), pair (snd d))] ∧ Q2 u X)"
  when "∀X⟨∇≠: u⟩st ∈ set (unlabel A'')" for X u
  using that A''(1) unfolding Q2_def constr unlabel_def by force
have 4: "∀d∈set D. (∃δ. Unifier δ (pair (t,s)) (pair (snd d)))
  ⟶ Γ (pair (t,s)) = Γ (pair (snd d))"

```

```

using prems(5) by (simp add: setopssst_def)

{ fix ac u u'
  assume a: " $\langle ac: u \dot{=} u' \rangle_{st} \in \text{set } (\text{unlabel } A')$ " " $\exists \delta. \text{Unifier } \delta \text{ u u'}$ "
  hence " $\langle ac: u \dot{=} u' \rangle_{st} \in \text{set } (\text{unlabel } A'')$ "  $\vee$  ( $\exists d \in \text{set } Di. u = \text{pair } (t,s) \wedge u' = \text{pair } (\text{snd } d)$ )
d))"
  using 2 by metis
  moreover {
    assume " $\langle ac: u \dot{=} u' \rangle_{st} \in \text{set } (\text{unlabel } A'')$ "
    hence " $\text{tfr}_{stp} (\langle ac: u \dot{=} u' \rangle_{st})$ "
      using 1 Ball_set_list_all[of " $\text{unlabel } A''$ "  $\text{tfr}_{stp}$ ]
      by fast
  } moreover {
    fix d assume " $d \in \text{set } Di$ " " $u = \text{pair } (t,s)$ " " $u' = \text{pair } (\text{snd } d)$ "
    hence " $\exists \delta. \text{Unifier } \delta \text{ u u'} \implies \Gamma u = \Gamma u'$ "
      using 4 dbproj_subseq_subset  $A''(3)$ 
      by fast
    hence " $\text{tfr}_{stp} (\langle ac: u \dot{=} u' \rangle_{st})$ "
      using Ball_set_list_all[of " $\text{unlabel } A''$ "  $\text{tfr}_{stp}$ ]
      by simp
    hence " $\Gamma u = \Gamma u'$ " using  $\text{tfr}_{stp\_list\_all\_alt\_def}$ [of " $\text{unlabel } A''$ "]
      using a(2) unfolding unlabel_def by auto
  } ultimately have " $\Gamma u = \Gamma u'$ "
    using  $\text{tfr}_{stp\_list\_all\_alt\_def}$ [of " $\text{unlabel } A''$ "] a(2)
    unfolding unlabel_def by auto
  } moreover {
    fix u U
    assume " $\forall U (\forall \neq: u)_{st} \in \text{set } (\text{unlabel } A')$ "
    hence " $\forall U (\forall \neq: u)_{st} \in \text{set } (\text{unlabel } A'')$ "  $\vee$ 
      ( $\exists d \in \text{set } (\text{filter } (\lambda d. d \notin \text{set } Di) D). u = [(\text{pair } (t,s), \text{pair } (\text{snd } d))] \wedge Q2 \text{ u } U$ )
    using 3 by metis
    hence " $Q1 \text{ u } U \vee Q2 \text{ u } U$ "
      using 1 4 subseqs_set_subset[OF  $A''(3)$ ]  $\text{tfr}_{stp\_list\_all\_alt\_def}$ [of " $\text{unlabel } A''$ "]
      unfolding Q1_def Q2_def
      by blast
  } ultimately show ?case
    using  $\text{tfr}_{stp\_list\_all\_alt\_def}$ [of " $\text{unlabel } A''$ "] unfolding Q1_def Q2_def unlabel_def by blast
next
case (7 i ac t s A D)
note prems = "7.prems"
note IH = "7.IH"

from prems(1) obtain d A'' where A'':
  " $A' = (i, \langle ac: (\text{pair } (t,s)) \dot{=} (\text{pair } (\text{snd } d)) \rangle_{st}) \# A''$ "
  " $A'' \in \text{set } (\text{tr}_{pc} A D)$ "
  " $d \in \text{set } (\text{dbproj } i D)$ "
by moura

have 1: " $\text{list\_all } \text{tfr}_{stp} (\text{unlabel } A'')$ "
  using IH[OF  $A''(2)$ ] sublm(1,2,3)[OF prems(2,3,4,5)] sublm(4)[OF prems(2,3,4,5)]
  by metis

have 2: " $\Gamma (\text{pair } (t,s)) = \Gamma (\text{pair } (\text{snd } d))$ "
  when " $\exists \delta. \text{Unifier } \delta (\text{pair } (t,s)) (\text{pair } (\text{snd } d))$ "
  using that prems(2,5)  $A''(3)$  unfolding  $\text{tfr}_{sst\_def}$  by (simp add: setopssst_def)

show ?case using  $A''(1)$  1 2 by fastforce
next
case (8 i X F F' A D)
note prems = "8.prems"
note IH = "8.IH"

define constr where

```

```

"constr = map (λG. (i, ∀X(∀≠: (F@G))st)) (trpairs F' (map snd (dbproj i D)))"

define Q1 where "Q1 ≡ (λ(F::('fun,'var) term × ('fun,'var) term) list) X.
  ∀x ∈ (fvpairs F) - set X. ∃a. Γ (Var x) = TAtom a)"

define Q2 where "Q2 ≡ (λ(M::('fun,'var) terms) X.
  ∀f T. Fun f T ∈ subtermsset M ⟶ T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X))"

have Q2_subset: "Q2 M' X" when "M' ⊆ M" "Q2 M X" for X M M'
  using that unfolding Q2_def by auto

have Q2_supset: "Q2 (M ∪ M') X" when "Q2 M X" "Q2 M' X" for X M M'
  using that unfolding Q2_def by auto

from prems obtain A'' where A'': "A' = constr@A'" "A'' ∈ set (trpc A D)"
  using constr_def by moura

have 0: "constr = [(i, ∀X(∀≠: F)st)]" when "F' = []" using that unfolding constr_def by simp

have 1: "list_all tfrstp (unlabel A'')"
  using IH[OF A''(2) sublmm(1,2,3)[OF prems(2,3,4,5)] sublmm(4)[OF prems(2,3,4,5)]]
  by metis

have 2: "(F' = [] ∧ Q1 F X) ∨ Q2 (trmspairs F ∪ pair ' set F') X"
  using prems(2) unfolding Q1_def Q2_def by simp

have 3: "F' = [] ⟹ Q1 F X ⟹ list_all tfrstp (unlabel constr)"
  using 0 2 tfrstp_list_all_alt_def[of "unlabel constr"] unfolding Q1_def by auto

{ fix c assume "c ∈ set (unlabel constr)"
  hence "∃G ∈ set (trpairs F' (map snd (dbproj i D))). c = ∀X(∀≠: (F@G))st"
    unfolding constr_def unlabel_def by force
} moreover {
  fix G
  assume G: "G ∈ set (trpairs F' (map snd (dbproj i D)))"
  and c: "∀X(∀≠: (F@G))st ∈ set (unlabel constr)"
  and e: "Q2 (trmspairs F ∪ pair ' set F') X"

  have d_Q2: "Q2 (pair ' set (map snd D)) X" unfolding Q2_def
  proof (intro allI impI)
    fix f T assume "Fun f T ∈ subtermsset (pair ' set (map snd D))"
    then obtain d where d: "d ∈ set (map snd D)" "Fun f T ∈ subterms (pair d)" by force
    hence "fv (pair d) ∩ set X = {}"
      using prems(4) unfolding pair_def by (force simp add: unlabel_def)
    thus "T = [] ∨ (∃s ∈ set T. s ∉ Var ' set X)"
      by (metis fv_disj_Fun_subterm_param_cases d(2))
  qed

  have "trmspairs (F@G) ⊆ trmspairs F ∪ pair ' set F' ∪ pair ' set (map snd D)"
    using trpairs_trms_subset[OF G] by force
  hence "Q2 (trmspairs (F@G)) X" using Q2_subset[OF _ Q2_supset[OF e d_Q2]] by metis
  hence "tfrstp (∀X(∀≠: (F@G))st)" by (metis Q2_def tfrstp.simps(2))
} ultimately have 4:
  "Q2 (trmspairs F ∪ pair ' set F') X ⟹ list_all tfrstp (unlabel constr)"
  using Ball_set by blast

have 5: "list_all tfrstp (unlabel constr)" using 2 3 4 by metis

show ?case using 1 5 A''(1) by (simp add: unlabel_def)
qed
qed
lemma tr_par_tfr:

```

```

assumes "A' ∈ set (trpc A [])" and "tfrsst (unlabel A)"
and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}"
shows "tfrst (unlabel A')"
proof -
  have *: "trmslst A' ⊆ trmssst (unlabel A) ∪ pair ' setopssst (unlabel A)"
    using tr_par_trms_subset[OF assms(1)] by simp
  hence "SMP (trmslst A') ⊆ SMP (trmssst (unlabel A) ∪ pair ' setopssst (unlabel A))"
    using SMP_mono by simp
  moreover have "tfrset (trmssst (unlabel A) ∪ pair ' setopssst (unlabel A))"
    using assms(2) unfolding tfrsst_def by fast
  ultimately have 1: "tfrset (trmslst A')" by (metis tfr_subset(2)[OF _ *])

  have **: "list_all tfrsttp (unlabel A)" using assms(2) unfolding tfrsst_def by fast
  have "pair ' setopssst (unlabel A) ⊆
    SMP (trmssst (unlabel A) ∪ pair ' setopssst (unlabel A)) - Var'\"
    using setopssst_are_pairs unfolding pair_def by auto
  hence "Γ t = Γ t'"
    when "∃δ. Unifier δ t t'" "t ∈ pair ' setopssst (unlabel A)" "t' ∈ pair ' setopssst (unlabel A)"
    for t t'
    using that assms(2) unfolding tfrsst_def tfrset_def by blast
  moreover have "fvpairs (unlabel []) = {}" "pair ' snd ' set [] = {}" by auto
  ultimately have 2: "list_all tfrsttp (unlabel A')"
    using tr_par_tfrsttp[OF assms(1) ** assms(3)] by simp

  show ?thesis by (metis 1 2 tfrst_def)
qed

lemma tr_par_proj:
  assumes "B ∈ set (trpc A D)"
  shows "proj n B ∈ set (trpc (proj n A) (proj n D))"
using assms
proof (induction A D arbitrary: B rule: trpc.induct)
  case (5 i t s S D)
  note prems = "5.prems"
  note IH = "5.IH"
  have IH': "proj n B ∈ set (trpc (proj n S) (proj n (List.insert (i,t,s) D)))"
    using prems IH by auto
  show ?case
  proof (cases "(i = ln n) ∨ (i = *)" )
    case True thus ?thesis
      using IH' proj_list_insert(1,2)[of n "(t,s)" D] proj_list_Cons(1,2)[of n _ S]
      by auto
    next
    case False
    then obtain m where "i = ln m" "n ≠ m" by (cases i) simp_all
    thus ?thesis
      using IH' proj_list_insert(3)[of n _ "(t,s)" D] proj_list_Cons(3)[of n _ "insert(t,s)" S]
      by auto
  qed
next
  case (6 i t s S D)
  note prems = "6.prems"
  note IH = "6.IH"
  define constr where "constr = (λDi D.
    (map (λd. (i,⟨check: (pair (t,s)) ≐ (pair (snd d))⟩st)) Di)@
    (map (λd. (i,⟨[ ] ∨ ≠: [(pair (t,s), pair (snd d))]⟩st)) [d←dbproj i D. d ∉ set Di]))"

  obtain Di B' where B':
    "B = constr Di D@B'"
    "Di ∈ set (subseqs (dbproj i D))"
    "B' ∈ set (trpc S [d←D. d ∉ set Di])"
    using prems constr_def by fastforce
  hence "proj n B' ∈ set (trpc (proj n S) (proj n [d←D. d ∉ set Di]))" using IH by simp

```

```

hence IH': "proj n B' ∈ set (trpc (proj n S) [d←proj n D. d ∉ set Di])" by (metis proj_filter)
show ?case
proof (cases "(i = ln n) ∨ (i = ★)")
  case True
  hence "proj n B = constr Di D@proj n B'" "Di ∈ set (subseqs (dbproj i (proj n D)))"
    using B'(1,2) proj_dbproj(1,2)[of n D] unfolding proj_def constr_def by auto
  moreover have "constr Di (proj n D) = constr Di D"
    using True proj_dbproj(1,2)[of n D] unfolding constr_def by presburger
  ultimately have "proj n B ∈ set (trpc ((i, delete(t,s))#proj n S) (proj n D))"
    using IH' unfolding constr_def by force
  thus ?thesis by (metis proj_list_Cons(1,2) True)
next
  case False
  then obtain m where m: "i = ln m" "n ≠ m" by (cases i) simp_all
  hence *: "(ln n) ≠ i" by simp
  have "proj n B = proj n B'" using B'(1) False unfolding constr_def proj_def by auto
  moreover have "[d←proj n D. d ∉ set Di] = proj n D"
    using proj_subseq[OF _ m(2)[symmetric]] m(1) B'(2) by simp
  ultimately show ?thesis using m(1) IH' proj_list_Cons(3)[OF m(2), of _ S] by auto
qed
next
  case (7 i ac t s S D)
  note prems = "7.prems"
  note IH = "7.IH"
  define constr where "constr = (
    λd::'lbl strand_label × ('fun,'var) term × ('fun,'var) term.
    (i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st))"

  obtain d B' where B':
    "B = constr d#B'"
    "d ∈ set (dbproj i D)"
    "B' ∈ set (trpc S D)"
  using prems constr_def by fastforce
  hence IH': "proj n B' ∈ set (trpc (proj n S) (proj n D))" using IH by auto

  show ?case
  proof (cases "(i = ln n) ∨ (i = ★)")
    case True
    hence "proj n B = constr d#proj n B'" "d ∈ set (dbproj i (proj n D))"
      using B' proj_list_Cons(1,2)[of n _ B']
      unfolding constr_def
      by (force, metis proj_dbproj(1,2))
    hence "proj n B ∈ set (trpc ((i, InSet ac t s)#proj n S) (proj n D))"
      using IH' unfolding constr_def by auto
    thus ?thesis using proj_list_Cons(1,2)[of n _ S] True by metis
  next
    case False
    then obtain m where m: "i = ln m" "n ≠ m" by (cases i) simp_all
    hence "proj n B = proj n B'" using B'(1) proj_list_Cons(3) unfolding constr_def by auto
    thus ?thesis
      using IH' m proj_list_Cons(3)[OF m(2), of "InSet ac t s" S]
      unfolding constr_def
      by auto
  qed
next
  case (8 i X F F' S D)
  note prems = "8.prems"
  note IH = "8.IH"

  define constr where
    "constr = (λD. map (λG. (i,∀X(∀≠: (F@G)⟩st)) (trpairs F' (map snd (dbproj i D)))))"

  obtain B' where B':

```



```

    "B = constr D@B'"
    "B' ∈ set (trpc S D)"
    using prems constr_def by fastforce
  hence IH': "proj n B' ∈ set (trpc (proj n S) (proj n D))" using IH by auto

  show ?case
  proof (cases "(i = ln n) ∨ (i = *)" )
    case True
    hence "proj n B = constr (proj n D)@proj n B'"
      using B'(1,2) proj_dbproj(1,2)[of n D] unfolding proj_def constr_def by auto
    hence "proj n B ∈ set (trpc ((i, NegChecks X F F')#proj n S) (proj n D))"
      using IH' unfolding constr_def by auto
    thus ?thesis using proj_list_Cons(1,2)[of n _ S] True by metis
  next
    case False
    then obtain m where m: "i = ln m" "n ≠ m" by (cases i) simp_all
    hence "proj n B = proj n B'" using B'(1) unfolding constr_def proj_def by auto
    thus ?thesis
      using IH' m proj_list_Cons(3)[OF m(2), of "NegChecks X F F'" S]
      unfolding constr_def
      by auto
  qed
qed (force simp add: proj_def)+

lemma tr_par_preserves_typing_cond:
  assumes "par_complsst A Sec" "typing_condsst (unlabel A)" "A' ∈ set (trpc A [])"
  shows "typing_cond (unlabel A')"
proof -
  have "wf'sst {} (unlabel A)"
    "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}"
    "wftrms (trmssst (unlabel A))"
  using assms(2) unfolding typing_condsst_def by simp_all
  hence 1: "wfst {} (unlabel A')"
    "fvst (unlabel A') ∩ bvarsst (unlabel A') = {}"
    "wftrms (trmsst (unlabel A'))"
    "Ana_invar_subst (ikst (unlabel A') ∪ assignment_rhsst (unlabel A'))"
  using tr_par_wf[OF assms(3)] Ana_invar_subst' by metis+

  have 2: "tfrst (unlabel A')" by (metis tr_par_tfr assms(2,3) typing_condsst_def)

  show ?thesis by (metis 1 2 typing_cond_def)
qed

lemma tr_par_preserves_par_comp:
  assumes "par_complsst A Sec" "A' ∈ set (trpc A [])"
  shows "par_comp A' Sec"
proof -
  let ?M = "λl. trmssst (proj_unl l A) ∪ pair ' setopssst (proj_unl l A)"
  let ?N = "λl. trmsprojlst l A'"
  have 0: "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (?M l1) (?M l2) Sec"
    using assms(1) unfolding par_complsst_def by simp_all

  { fix l1 l2::'lbl assume *: "l1 ≠ l2"
    hence "GSMP_disjoint (?M l1) (?M l2) Sec" using 0(1) by metis
    moreover have "pair ' snd ' set (proj n []) = {}" for n::'lbl unfolding proj_def by simp
    hence "?N l1 ⊆ ?M l1" "?N l2 ⊆ ?M l2"
      using tr_par_trms_subset[OF tr_par_proj[OF assms(2)]] by (metis Un_empty_right)+
    ultimately have "GSMP_disjoint (?N l1) (?N l2) Sec"
      using GSMP_disjoint_subset by presburger
  }
  hence 1: "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (trmsprojlst l1 A') (trmsprojlst l2 A') Sec"
    using 0(1) by metis

```

```

have 2: "ground Sec" "∀ s ∈ Sec. ∀ s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ Sec"
  using assms(1) unfolding par_complst_def by metis+

show ?thesis using 1 2 unfolding par_comp_def by metis
qed

lemma tr_leaking_prefix_exists:
  assumes "A' ∈ set (trpc A [])" "prefix B A'" "ikst (proj_unl n B) ·set I ⊢ t · I"
  shows "∃ C D. prefix C B ∧ prefix D A ∧ C ∈ set (trpc D []) ∧ (ikst (proj_unl n C) ·set I ⊢ t · I)"
proof -
  let ?P = "λB C C'. B = C@C' ∧ (∀ n t. (n, receive⟨t⟩st) ∉ set C') ∧
    (C = [] ∨ (∃ n t. suffix [(n, receive⟨t⟩st)] C))"
  have "∃ C C'. ?P B C C'"
  proof (induction B)
    case (Cons b B)
    then obtain C C' n s where *: "?P B C C'" "b = (n,s)" by moura
    show ?case
    proof (cases "C = []")
      case True
      note T = True
      show ?thesis
      proof (cases "∃ t. s = receive⟨t⟩st")
        case True
        hence "?P (b#B) [b] C'" using * T by auto
        thus ?thesis by metis
      next
        case False
        hence "?P (b#B) [] (b#C')" using * T by auto
        thus ?thesis by metis
      qed
    next
      case False
      hence "?P (b#B) (b#C) C'" using * unfolding suffix_def by auto
      thus ?thesis by metis
    qed
  qed simp
  then obtain C C' where C:
    "B = C@C'" "∀ n t. (n, receive⟨t⟩st) ∉ set C'"
    "C = [] ∨ (∃ n t. suffix [(n, receive⟨t⟩st)] C)"
    by moura
  hence 1: "prefix C B" by simp
  hence 2: "prefix C A'" using assms(2) by simp

  have "∧m t. (m, receive⟨t⟩st) ∈ set B ⇒ (m, receive⟨t⟩st) ∈ set C" using C by auto
  hence "∧ t. receive⟨t⟩st ∈ set (proj_unl n B) ⇒ receive⟨t⟩st ∈ set (proj_unl n C)"
    unfolding unlabeled_def proj_def by force
  hence "ikst (proj_unl n B) ⊆ ikst (proj_unl n C)" using ikst-is_rcv_set by auto
  hence 3: "ikst (proj_unl n C) ·set I ⊢ t · I" by (metis ideduct_mono[OF assms(3)] subst_all_mono)

  { fix D E m t assume "suffix [(m, receive⟨t⟩st)] E" "prefix E A'" "A' ∈ set (trpc A D)"
    hence "∃ F. prefix F A ∧ E ∈ set (trpc F D)"
    proof (induction A D arbitrary: A' E rule: trpc.induct)
      case (1 D) thus ?case by simp
    next
      case (2 i t' S D)
      note prems = "2.prems"
      note IH = "2.IH"
      obtain A'' where *: "A' = (i, send⟨t'⟩st)#A'" "A'' ∈ set (trpc S D)"
        using prems(3) by auto
      have "E ≠ []" using prems(1) by auto
      then obtain E' where **: "E = (i, send⟨t'⟩st)#E'"
        using *(1) prems(2) by (cases E) auto

```

```

hence "suffix [(m, receive⟨t⟩st)] E'" "prefix E' A'"
  using *(1) prems(1,2) suffix_Cons[of _ _ E'] by auto
then obtain F where "prefix F S" "E' ∈ set (trpc F D)"
  using *(2) ** IH by metis
hence "prefix ((i, Send t')#F) ((i, Send t')#S)" "E ∈ set (trpc ((i, Send t')#F) D)"
  using ** by auto
thus ?case by metis
next
case (3 i t' S D)
note prems = "3.prems"
note IH = "3.IH"
obtain A'' where *: "A' = (i, receive⟨t'⟩st)#A'" "A'' ∈ set (trpc S D)"
  using prems(3) by auto
have "E ≠ []" using prems(1) by auto
then obtain E' where **: "E = (i, receive⟨t'⟩st)#E'"
  using *(1) prems(2) by (cases E) auto
show ?case
proof (cases "(m, receive⟨t⟩st) = (i, receive⟨t'⟩st)")
case True
note T = True
show ?thesis
proof (cases "suffix [(m, receive⟨t⟩st)] E'")
case True
hence "suffix [(m, receive⟨t⟩st)] E'" "prefix E' A'"
  using ** *(1) prems(1,2) by auto
then obtain F where "prefix F S" "E' ∈ set (trpc F D)"
  using *(2) ** IH by metis
hence "prefix ((i, receive⟨t'⟩)#F) ((i, receive⟨t'⟩)#S)"
  "E ∈ set (trpc ((i, receive⟨t'⟩)#F) D)"
  using ** by auto
thus ?thesis by metis
next
case False
hence "E' = []"
  using ***(1) T prems(1)
  suffix_Cons[of "[ (m, receive⟨t⟩st) ]" "(m, receive⟨t⟩st)" E']
  by auto
hence "prefix [(i, receive⟨t'⟩)] ((i, receive⟨t'⟩) # S) ∧ E ∈ set (trpc [(i, receive⟨t'⟩)] D)"
  using * ** prems by auto
thus ?thesis by metis
qed
next
case False
hence "suffix [(m, receive⟨t⟩st)] E'" "prefix E' A'"
  using ** *(1) prems(1,2) suffix_Cons[of _ _ E'] by auto
then obtain F where "prefix F S" "E' ∈ set (trpc F D)" using *(2) ** IH by metis
hence "prefix ((i, receive⟨t'⟩)#F) ((i, receive⟨t'⟩)#S)" "E ∈ set (trpc ((i, receive⟨t'⟩)#F) D)"
  using ** by auto
thus ?thesis by metis
qed
next
case (4 i ac t' t'' S D)
note prems = "4.prems"
note IH = "4.IH"
obtain A'' where *: "A' = (i, ⟨ac: t' ≐ t''⟩st)#A'" "A'' ∈ set (trpc S D)"
  using prems(3) by auto
have "E ≠ []" using prems(1) by auto
then obtain E' where **: "E = (i, ⟨ac: t' ≐ t''⟩st)#E'"
  using *(1) prems(2) by (cases E) auto
hence "suffix [(m, receive⟨t⟩st)] E'" "prefix E' A'"
  using *(1) prems(1,2) suffix_Cons[of _ _ E'] by auto
then obtain F where "prefix F S" "E' ∈ set (trpc F D)"
  using *(2) ** IH by metis

```

```

hence "prefix ((i,Equality ac t' t'')#F) ((i,Equality ac t' t'')#S)"
      "E ∈ set (trpc ((i,Equality ac t' t'')#F) D)"
  using ** by auto
  thus ?case by metis
next
case (5 i t' s S D)
note prems = "5.prems"
note IH = "5.IH"
have *: "A' ∈ set (trpc S (List.insert (i,t',s) D))" using prems(3) by auto
have "E ≠ []" using prems(1) by auto
hence "suffix [(m, receive⟨t⟩st)] E" "prefix E A'"
  using *(1) prems(1,2) suffix_Cons[of _ _ E] by auto
then obtain F where "prefix F S" "E ∈ set (trpc F (List.insert (i,t',s) D))"
  using * IH by metis
hence "prefix ((i,insert⟨t',s⟩)#F) ((i,insert⟨t',s⟩)#S)"
      "E ∈ set (trpc ((i,insert⟨t',s⟩)#F) D)"
  by auto
  thus ?case by metis
next
case (6 i t' s S D)
note prems = "6.prems"
note IH = "6.IH"

define constr where "constr = (λDi.
  (map (λd. (i,⟨check: (pair (t',s)) ≐ (pair (snd d))⟩st)) Di)@
  (map (λd. (i,⟨∀ []⟨≠: [(pair (t',s), pair (snd d))]⟩st)
    (filter (λd. d ∉ set Di) (dbproj i D))))))"

obtain A'' Di where *:
  "A' = constr Di@A'" "A'' ∈ set (trpc S (filter (λd. d ∉ set Di) D))"
  "Di ∈ set (subseqs (dbproj i D))"
  using prems(3) constr_def by auto
have ***: "(m, receive⟨t⟩st) ∉ set (constr Di)" using constr_def by auto
have "E ≠ []" using prems(1) by auto
then obtain E' where *: "E = constr Di@E'"
  using *(1) prems(1,2) ***
  by (metis (mono_tags, lifting) Un_iff list.set_intros(1) prefixI prefix_def
    prefix_same_cases set_append suffix_def)
hence "suffix [(m, receive⟨t⟩st)] E'" "prefix E' A'"
  using *(1) prems(1,2) suffix_append[of "[(m, receive⟨t⟩st)]" "constr Di" E'] ***
  by (metis (no_types, hide_lams) Nil_suffix append_Nil2 in_set_conv_decomp rev_exhaust
    snoc_suffix_snoc suffix_appendD,
    auto)
then obtain F where "prefix F S" "E' ∈ set (trpc F (filter (λd. d ∉ set Di) D))"
  using *(2,3) ** IH by metis
hence "prefix ((i,delete⟨t',s⟩)#F) ((i,delete⟨t',s⟩)#S)"
      "E ∈ set (trpc ((i,delete⟨t',s⟩)#F) D)"
  using *(3) ** constr_def by auto
  thus ?case by metis
next
case (7 i ac t' s S D)
note prems = "7.prems"
note IH = "7.IH"

define constr where "constr = (
  λd: (('lbl strand_label × ('fun,'var) term × ('fun,'var) term)).
  (i,⟨ac: (pair (t',s)) ≐ (pair (snd d))⟩st))"

obtain A'' d where *: "A' = constr d#A'" "A'' ∈ set (trpc S D)" "d ∈ set (dbproj i D)"
  using prems(3) constr_def by auto
have "E ≠ []" using prems(1) by auto
then obtain E' where *: "E = constr d#E'" using *(1) prems(2) by (cases E) auto
hence "suffix [(m, receive⟨t⟩st)] E'" "prefix E' A'"

```

```

    using *(1) prems(1,2) suffix_Cons[of _ _ E'] using constr_def by auto
  then obtain F where "prefix F S" "E' ∈ set (trpc F D)" using *(2) ** IH by metis
  hence "prefix ((i, InSet ac t' s)#F) ((i, InSet ac t' s)#S)"
    "E ∈ set (trpc ((i, InSet ac t' s)#F) D)"
    using *(3) ** unfolding constr_def by auto
  thus ?case by metis
next
case (8 i X G G' S D)
note prems = "8.prems"
note IH = "8.IH"

define constr where
  "constr = map (λH. (i, ∀X(∀≠: (G@H))st)) (trpairs G' (map snd (dbproj i D)))"

obtain A'' where *: "A' = constr@A'" "A'' ∈ set (trpc S D)"
  using prems(3) constr_def by auto
have ***: "(m, receive⟨t⟩st) ∉ set constr" using constr_def by auto
have "E ≠ []" using prems(1) by auto
then obtain E' where **: "E = constr@E'"
  using *(1) prems(1,2) ***
  by (metis (mono_tags, lifting) Un_iff list.set_intros(1) prefixI prefix_def
    prefix_same_cases set_append suffix_def)
hence "suffix [(m, receive⟨t⟩st)] E'" "prefix E' A'"
  using *(1) prems(1,2) suffix_append[of "[m, receive⟨t⟩st]" constr E'] ***
  by (metis (no_types, hide_lams) Nil_suffix append_Nil2 in_set_conv_decomp rev_exhaust
    snoc_suffix_snoc suffix_appendD,
    auto)
then obtain F where "prefix F S" "E' ∈ set (trpc F D)" using *(2) ** IH by metis
hence "prefix ((i, NegChecks X G G')#F) ((i, NegChecks X G G')#S)"
  "E ∈ set (trpc ((i, NegChecks X G G')#F) D)"
  using ** constr_def by auto
thus ?case by metis
qed
}
moreover have "prefix [] A" "[] ∈ set (trpc [] [])" by auto
ultimately have 4: "∃D. prefix D A ∧ C ∈ set (trpc D [])" using C(3) assms(1) 2 by blast

show ?thesis by (metis 1 3 4)
qed

```

### 6.2.5 Theorem: Semantic Equivalence of Translation

context  
begin

An alternative version of the translation that does not perform database-state projections. It is used as an intermediate step in the proof of semantic equivalence.

```

private fun tr'pc::
  "('fun, 'var, 'lbl) labeled_stateful_strand ⇒ ('fun, 'var, 'lbl) labeleddbstatelist
  ⇒ ('fun, 'var, 'lbl) labeled_strand list"
where
  "tr'pc [] D = [[]]"
| "tr'pc ((i, send⟨t⟩)#A) D = map ((#) (i, send⟨t⟩st)) (tr'pc A D)"
| "tr'pc ((i, receive⟨t⟩)#A) D = map ((#) (i, receive⟨t⟩st)) (tr'pc A D)"
| "tr'pc ((i, ⟨ac: t ≐ t'⟩)#A) D = map ((#) (i, ⟨ac: t ≐ t'⟩st)) (tr'pc A D)"
| "tr'pc ((i, insert⟨t, s⟩)#A) D = tr'pc A (List.insert (i, ⟨t, s⟩) D)"
| "tr'pc ((i, delete⟨t, s⟩)#A) D = (
  concat (map (λDi. map (λB. (map (λd. (i, ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st)) Di)@
    (map (λd. (i, ∀ []⟨≠: [(pair (t, s), pair (snd d))⟩st))
      [d ← D. d ∉ set Di])@B)
    (tr'pc A [d ← D. d ∉ set Di])))
  (subseqs D)))"
| "tr'pc ((i, ⟨ac: t ∈ s⟩)#A) D =

```

```

concat (map (λB. map (λd. (i, ⟨ac: (pair (t,s)) ÷ (pair (snd d))⟩st)#B) D) (tr'pc A D))"
| "tr'pc ((i, ∀X(∀≠: F ∨≠: F'))#A) D =
  map ((@) (map (λG. (i, ∀X(∀≠: (F@G))st)) (trpairs F' (map snd D)))) (tr'pc A D)"

```

## Part 1

```

private lemma tr'_par_iff_unlabel_tr:
  assumes "∀ (i,p) ∈ setopslssst A ∪ set D.
    ∀ (j,q) ∈ setopslssst A ∪ set D.
      p = q ⟶ i = j"
  shows "(∃ C ∈ set (tr'pc A D). B = unlabel C) ⟷ B ∈ set (tr (unlabel A) (unlabel D))"
  (is "?A ⟷ ?B")
proof
  { fix C have "C ∈ set (tr'pc A D) ⟹ unlabel C ∈ set (tr (unlabel A) (unlabel D))" using assms
  proof (induction A D arbitrary: C rule: tr'_pc.induct)
    case (5 i t s S D)
    hence "unlabel C ∈ set (tr (unlabel S) (unlabel (List.insert (i, t, s) D)))"
      by (auto simp add: setopslssst_def)
    moreover have
      "insert (i,t,s) (set D) ⊆ setopslssst ((i,insert⟨t,s⟩)#S) ∪ set D"
      by (auto simp add: setopslssst_def)
    hence "∀ (j,p) ∈ insert (i,t,s) (set D). ∀ (k,q) ∈ insert (i,t,s) (set D). p = q ⟶ j = k"
      using "5.prem" (2) by blast
    hence "unlabel (List.insert (i, t, s) D) = (List.insert (t, s) (unlabel D))"
      using map_snd_list_insert_distrib[of "(i,t,s)" D] unfolding unlabel_def by simp
    ultimately show ?case by auto
  next
    case (6 i t s S D)
    let ?f1 = "λd. ⟨check: (pair (t,s)) ÷ (pair d)⟩st"
    let ?g1 = "λd. ∀ [] (∀≠: [(pair (t,s), pair d)]st)"
    let ?f2 = "λd. (i, ?f1 (snd d))"
    let ?g2 = "λd. (i, ?g1 (snd d))"

    define constr1 where "constr1 = (λDi. (map ?f1 Di)@(map ?g1 [d←unlabel D. d ∉ set Di]))"
    define constr2 where "constr2 = (λDi. (map ?f2 Di)@(map ?g2 [d←D. d ∉ set Di]))"

    obtain C' Di where C':
      "Di ∈ set (subseqs D)"
      "C = constr2 Di@C'"
      "C' ∈ set (tr'pc S [d←D. d ∉ set Di])"
      using "6.prem" (1) unfolding constr2_def by mouna

    have 0: "set [d←D. d ∉ set Di] ⊆ set D"
      "setopslssst S ⊆ setopslssst ((i, delete⟨t,s⟩)#S)"
      by (auto simp add: setopslssst_def)
    hence 1:
      "∀ (j, p) ∈ setopslssst S ∪ set [d←D. d ∉ set Di].
        ∀ (k, q) ∈ setopslssst S ∪ set [d←D. d ∉ set Di].
          p = q ⟶ j = k"
      using "6.prem" (2) by blast

    have "∀ (i,p) ∈ set D ∪ set Di. ∀ (j,q) ∈ set D ∪ set Di. p = q ⟶ i = j"
      using "6.prem" (2) subseqs_set_subset(1)[OF C' (1)] by blast
    hence 2: "unlabel [d←D. d ∉ set Di] = [d←unlabel D. d ∉ set (unlabel Di)]"
      using unlabel_filter_eq[of D "set Di"] unfolding unlabel_def by simp

    have 3:
      "λf g::('a × 'a ⇒ 'c). ⋀ A B::(('b × 'a × 'a) list).
        map snd ((map (λd. (i, f (snd d))) A)@(map (λd. (i, g (snd d))) B)) =
        map f (map snd A)@map g (map snd B)"
      by simp
    have "unlabel (constr2 Di) = constr1 (unlabel Di)"
      using 2 3[of ?f1 Di ?g1 "[d←D. d ∉ set Di]"]

```

```

    by (simp add: constr1_def constr2_def unlabel_def)
  hence 4: "unlabel C = constr1 (unlabel Di)@unlabel C'"
    using C'(2) unlabel_append by metis

  have "unlabel Di ∈ set (map unlabel (subseqs D))"
    using C'(1) unfolding unlabel_def by simp
  hence 5: "unlabel Di ∈ set (subseqs (unlabel D))"
    using map_subseqs[of snd D] unfolding unlabel_def by simp

  show ?case using "6.IH"[OF C'(1,3) 1] 2 4 5 unfolding constr1_def by auto
next
case (7 i ac t s S D)
obtain C' d where C':
  "C = (i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#C'"
  "C' ∈ set (tr'pc S D)" "d ∈ set D"
  using "7.prem" (1) by maura

  have "setopslssst S ∪ set D ⊆ setopslssst ((i,InSet ac t s)#S) ∪ set D"
    by (auto simp add: setopslssst_def)
  hence "∀ (j, p) ∈ setopslssst S ∪ set D.
    ∀ (k, q) ∈ setopslssst S ∪ set D.
    p = q ⟶ j = k"
    using "7.prem" (2) by blast
  hence "unlabel C' ∈ set (tr (unlabel S) (unlabel D))" using "7.IH"[OF C'(2)] by auto
  thus ?case using C' unfolding unlabel_def by force
next
case (8 i X F F' S D)
obtain C' where C':
  "C = map (λG. (i,∀ X⟨V≠: (F@G)⟩st)) (trpairs F' (map snd D))@C'"
  "C' ∈ set (tr'pc S D)"
  using "8.prem" (1) by maura

  have "setopslssst S ∪ set D ⊆ setopslssst ((i,NegChecks X F F')#S) ∪ set D"
    by (auto simp add: setopslssst_def)
  hence "∀ (j, p) ∈ setopslssst S ∪ set D.
    ∀ (k, q) ∈ setopslssst S ∪ set D.
    p = q ⟶ j = k"
    using "8.prem" (2) by blast
  hence "unlabel C' ∈ set (tr (unlabel S) (unlabel D))" using "8.IH"[OF C'(2)] by auto
  thus ?case using C' unfolding unlabel_def by auto
qed (auto simp add: setopslssst_def)
} thus "?A ⟹ ?B" by blast

show "?B ⟹ ?A" using assms
proof (induction A arbitrary: B D)
  case (Cons a A)
  obtain ia sa where a: "a = (ia,sa)" by maura

  have "setopslssst A ⊆ setopslssst (a#A)" using a by (cases sa) (auto simp add: setopslssst_def)
  hence 1: "∀ (j, p) ∈ setopslssst A ∪ set D.
    ∀ (k, q) ∈ setopslssst A ∪ set D.
    p = q ⟶ j = k"
    using Cons.prem (2) by blast

  show ?case
  proof (cases sa)
    case (Send t)
    then obtain B' where B':
      "B = send⟨t⟩st#B'"
      "B' ∈ set (tr (unlabel A) (unlabel D))"
      using Cons.prem (1) a by auto
    thus ?thesis using Cons.IH[OF B'(2) 1] a B'(1) Send by auto
  next

```

```

case (Receive t)
then obtain B' where B':
  "B = receive⟨t⟩st#B'"
  "B' ∈ set (tr (unlabel A) (unlabel D))"
  using Cons.premis(1) a by auto
thus ?thesis using Cons.IH[OF B'(2) 1] a B'(1) Receive by auto
next
case (Equality ac t t')
then obtain B' where B':
  "B = ⟨ac: t ≐ t'⟩st#B'"
  "B' ∈ set (tr (unlabel A) (unlabel D))"
  using Cons.premis(1) a by auto
thus ?thesis using Cons.IH[OF B'(2) 1] a B'(1) Equality by auto
next
case (Insert t s)
hence B: "B ∈ set (tr (unlabel A) (List.insert (t,s) (unlabel D)))"
  using Cons.premis(1) a by auto

let ?P = "λi. List.insert (t,s) (unlabel D) = unlabel (List.insert (i,t,s) D)"

{ obtain j where j: "?P j" "j = ia ∨ (j,t,s) ∈ set D"
  using labeled_list_insert_eq_ex_cases[of "(t,s)" D ia] by moura
  hence "j = ia" using Cons.premis(2) a Insert by (auto simp add: setopslsst_def)
  hence "?P ia" using j(1) by metis
} hence j: "?P ia" by metis

have 2: "∀ (k1, p) ∈ setopslsst A ∪ set (List.insert (ia,t,s) D).
  ∀ (k2, q) ∈ setopslsst A ∪ set (List.insert (ia,t,s) D).
  p = q ⟶ k1 = k2"
  using Cons.premis(2) a Insert by (auto simp add: setopslsst_def)

show ?thesis using Cons.IH[OF _ 2] j(1) B Insert a by auto
next
case (Delete t s)
define c where "c ≡ (λ(i::'lbl strand_label) Di.
  map (λd. (i,⟨check: (pair (t,s)) ≐ (pair (snd d))⟩st)) Di @
  map (λd. (i,⟨[]⟨≠: [(pair (t,s), pair (snd d))]⟩st)) [d←D. d ∉ set Di]))"

define d where "d ≡ (λDi.
  map (λd. ⟨check: (pair (t,s)) ≐ (pair d)⟩st) Di @
  map (λd. ∀ []⟨≠: [(pair (t,s), pair d)]⟩st) [d←unlabel D. d ∉ set Di])"

obtain B' Di where B':
  "B = d Di@B'" "Di ∈ set (subseqs (unlabel D))"
  "B' ∈ set (tr (unlabel A) [d←unlabel D. d ∉ set Di])"
  using Cons.premis(1) a Delete unfolding d_def by auto

obtain Di' where Di': "Di' ∈ set (subseqs D)" "unlabel Di' = Di"
  using unlabel_subseqsD[OF B'(2)] by moura

have 2: "∀ (j, p) ∈ setopslsst A ∪ set [d←D. d ∉ set Di'].
  ∀ (k, q) ∈ setopslsst A ∪ set [d←D. d ∉ set Di'].
  p = q ⟶ j = k"
  using 1 subseqs_subset[OF Di'(1)]
  filter_is_subset[of "λd. d ∉ set Di'"]
  by blast

have "set Di' ⊆ set D" by (rule subseqs_subset[OF Di'(1)])
hence "∀ (j, p) ∈ set D ∪ set Di'. ∀ (k, q) ∈ set D ∪ set Di'. p = q ⟶ j = k"
  using Cons.premis(2) by blast
hence 3: "[d←unlabel D. d ∉ set Di] = unlabel [d←D. d ∉ set Di']"
  using Di'(2) unlabel_filter_eq[of D "set Di'"] unfolding unlabel_def by auto

```



```

obtain C where C: "C ∈ set (tr'_{pc} A [d ← D. d ∉ set Di'])" "B' = unlabel C"
  using 3 Cons.IH[OF _ 2] B'(3) by auto
hence 4: "c ia Di' @ C ∈ set (tr'_{pc} (a # A) D)" using Di'(1) a Delete unfolding c_def by auto

have "unlabel (c ia Di') = d Di" using Di' 3 unfolding c_def d_def unlabel_def by auto
hence 5: "B = unlabel (c ia Di' @ C)" using B'(1) C(2) unlabel_append[of "c ia Di'" C] by simp

show ?thesis using 4 5 by blast
next
case (InSet ac t s)
then obtain B' d where B':
  "B = ⟨ac: (pair (t,s)) ≐ (pair d)⟩_{st} # B'"
  "B' ∈ set (tr (unlabel A) (unlabel D))"
  "d ∈ set (unlabel D)"
  using Cons.prem1 a by auto
thus ?thesis using Cons.IH[OF _ 1] a InSet unfolding unlabel_def by auto
next
case (NegChecks X F F')
then obtain B' where B':
  "B = map (λG. ∀X(∀≠: (F@G)_{st}) (tr_{pairs} F' (unlabel D)) @ B'"
  "B' ∈ set (tr (unlabel A) (unlabel D))"
  using Cons.prem1 a by auto
thus ?thesis using Cons.IH[OF _ 1] a NegChecks unfolding unlabel_def by auto
qed
qed simp
qed

```

## Part 2

```

private lemma tr_par_iff_tr'_par:
  assumes "∀(i,p) ∈ setops_{lsst} A ∪ set D. ∀(j,q) ∈ setops_{lsst} A ∪ set D.
    (∃δ. Unifier δ (pair p) (pair q)) ⟶ i = j"
  (is "?R3 A D")
  and "∀(l,t,s) ∈ set D. (fv t ∪ fv s) ∩ bvars_{sst} (unlabel A) = {}" (is "?R4 A D")
  and "fv_{sst} (unlabel A) ∩ bvars_{sst} (unlabel A) = {}" (is "?R5 A D")
  shows "(∃B ∈ set (tr_{pc} A D). ⟦M; unlabel B⟧_d I) ⟷ (∃C ∈ set (tr'_{pc} A D). ⟦M; unlabel C⟧_d I)"
  (is "?P ⟷ ?Q")
proof
  { fix B assume "B ∈ set (tr_{pc} A D)" "⟦M; unlabel B⟧_d I"
    hence ?Q using assms
    proof (induction A D arbitrary: B M rule: tr_{pc}.induct)
      case (1 D) thus ?case by simp
    next
      case (2 i t S D)
      note prems = "2.prem1"
      note IH = "2.IH"

      obtain B' where B': "B = (i, send⟨t⟩_{st}) # B'" "B' ∈ set (tr_{pc} S D)"
        using prems(1) by mouna

      have 1: "⟦M; unlabel B'⟧_d I" using prems(2) B'(1) by simp
      have 4: "?R3 S D" using prems(3) by (auto simp add: setops_{lsst}_def)
      have 5: "?R4 S D" using prems(4) by force
      have 6: "?R5 S D" using prems(5) by force

      have 7: "M ⊢ t · I" using prems(2) B'(1) by simp

      obtain C where C: "C ∈ set (tr'_{pc} S D)" "⟦M; unlabel C⟧_d I"
        using IH[OF B'(2) 1 4 5 6] by mouna
      hence "((i, send⟨t⟩_{st}) # C) ∈ set (tr'_{pc} ((i, Send t) # S) D)" "⟦M; unlabel ((i, send⟨t⟩_{st}) # C)⟧_d I"
        using 7 by auto
      thus ?case by metis
    next

```

```

case (3 i t S D)
note prems = "3.prems"
note IH = "3.IH"

obtain B' where B': "B = (i, receive⟨t⟩st)#B'" "B' ∈ set (trpc S D)" using prems(1) by moura

have 1: "⟦insert (t · I) M; unlabel B'⟧d I" using prems(2) B'(1) by simp
have 4: "?R3 S D" using prems(3) by (auto simp add: setopslssst_def)
have 5: "?R4 S D" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

obtain C where C: "C ∈ set (tr'pc S D)" "⟦insert (t · I) M; unlabel C⟧d I"
using IH[OF B'(2) 1 4 5 6] by moura
hence "((i, receive⟨t⟩st)#C) ∈ set (tr'pc ((i, receive⟨t⟩)#S) D)"
"⟦insert (t · I) M; unlabel ((i, receive⟨t⟩st)#C)⟧d I"
by auto
thus ?case by auto
next
case (4 i ac t t' S D)
note prems = "4.prems"
note IH = "4.IH"

obtain B' where B': "B = (i, ⟨ac: t ≐ t'⟩st)#B'" "B' ∈ set (trpc S D)"
using prems(1) by moura

have 1: "⟦M; unlabel B'⟧d I" using prems(2) B'(1) by simp
have 4: "?R3 S D" using prems(3) by (auto simp add: setopslssst_def)
have 5: "?R4 S D" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

have 7: "t · I = t' · I" using prems(2) B'(1) by simp

obtain C where C: "C ∈ set (tr'pc S D)" "⟦M; unlabel C⟧d I"
using IH[OF B'(2) 1 4 5 6] by moura
hence "((i, ⟨ac: t ≐ t'⟩st)#C) ∈ set (tr'pc ((i, Equality ac t t')#S) D)"
"⟦M; unlabel ((i, ⟨ac: t ≐ t'⟩st)#C)⟧d I"
using 7 by auto
thus ?case by metis
next
case (5 i t s S D)
note prems = "5.prems"
note IH = "5.IH"

have B: "B ∈ set (trpc S (List.insert (i, t, s) D))" using prems(1) by simp

have 1: "⟦M; unlabel B⟧d I" using prems(2) B(1) by simp
have 4: "?R3 S (List.insert (i, t, s) D)" using prems(3) by (auto simp add: setopslssst_def)
have 5: "?R4 S (List.insert (i, t, s) D)" using prems(4,5) by force
have 6: "?R5 S D" using prems(5) by force

show ?case using IH[OF B(1) 1 4 5 6] by simp
next
case (6 i t s S D)
note prems = "6.prems"
note IH = "6.IH"

let ?c11 = "λDi. map (λd. (i, ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st)) Di"
let ?cu1 = "λDi. map (λd. ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st) Di"
let ?c12 = "λDi. map (λd. (i, ∀ [] ⟨≠: [(pair (t, s), pair (snd d))]⟩st)) [d ← dbproj i D. d ∉ set
Di]"
let ?cu2 = "λDi. map (λd. ∀ [] ⟨≠: [(pair (t, s), pair (snd d))]⟩st) [d ← dbproj i D. d ∉ set Di]"
let ?d11 = "λDi. map (λd. (i, ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st)) Di"

```

```

let ?du1 = "λDi. map (λd. ⟨check: (pair (t,s)) ≐ (pair (snd d))⟩st) Di"
let ?dl2 = "λDi. map (λd. (i,∀ [] ⟨≠: [(pair (t,s), pair (snd d))]⟩st)) [d←D. d∉set Di]"
let ?du2 = "λDi. map (λd. ∀ [] ⟨≠: [(pair (t,s), pair (snd d))]⟩st) [d←D. d∉set Di]"

define c where c: "c = (λDi. ?cl1 Di@?cl2 Di)"
define d where d: "d = (λDi. ?dl1 Di@?dl2 Di)"

obtain B' Di where B':
  "Di ∈ set (subseqs (dbproj i D))" "B = c Di@B'" "B' ∈ set (trpc S [d←D. d ∉ set Di])"
  using prems(1) c by moura

have 0: "ikst (unlabel (c Di)) = {}" "ikst (unlabel (d Di)) = {}"
  "unlabel (?cl1 Di) = ?cu1 Di" "unlabel (?cl2 Di) = ?cu2 Di"
  "unlabel (?dl1 Di) = ?du1 Di" "unlabel (?dl2 Di) = ?du2 Di"
  unfolding c d unlabel_def by force+

have 1: "⌊M; unlabel B'⌋d I" using prems(2) B'(2) 0(1) unfolding unlabel_def by auto

{ fix j p k q
  assume "(j, p) ∈ setopslsst S ∪ set [d←D. d ∉ set Di]"
    "(k, q) ∈ setopslsst S ∪ set [d←D. d ∉ set Di]"
  hence "(j, p) ∈ setopslsst ((i, delete⟨t,s⟩)#S) ∪ set D"
    "(k, q) ∈ setopslsst ((i, delete⟨t,s⟩)#S) ∪ set D"
    using dbproj_subseq_subset[OF B'(1)] by (auto simp add: setopslsst_def)
  hence "(∃δ. Unifier δ (pair p) (pair q)) ⟹ j = k" using prems(3) by blast
} hence 4: "?R3 S [d←D. d ∉ set Di]" by blast

have 5: "?R4 S (filter (λd. d ∉ set Di) D)" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

obtain C where C: "C ∈ set (tr'pc S [d←D. d ∉ set Di])" "⌊M; unlabel C⌋d I"
  using IH[OF B'(1,3) 1 4 5 6] by moura

have 7: "⌊M; unlabel (c Di)⌋d I" "⌊M; unlabel B'⌋d I"
  using prems(2) B'(2) 0(1) strand_sem_split(3,4)[of M "unlabel (c Di)" "unlabel B'"]
  unfolding c unlabel_def by auto

have "⌊M; unlabel (?cl2 Di)⌋d I" using 7(1) 0(1) unfolding c unlabel_def by auto
hence "⌊M; ?cu2 Di⌋d I" by (metis 0(4))
moreover {
  fix j p k q
  assume "(j, p) ∈ {(i, t, s)} ∪ set D ∪ set Di"
    "(k, q) ∈ {(i, t, s)} ∪ set D ∪ set Di"
  hence "(j, p) ∈ setopslsst ((i, delete⟨t,s⟩)#S) ∪ set D"
    "(k, q) ∈ setopslsst ((i, delete⟨t,s⟩)#S) ∪ set D"
    using dbproj_subseq_subset[OF B'(1)] by (auto simp add: setopslsst_def)
  hence "(∃δ. Unifier δ (pair p) (pair q)) ⟹ j = k" using prems(3) by blast
} hence "∀ (j, p) ∈ {(i, t, s)} ∪ set D ∪ set Di.
  ∀ (k, q) ∈ {(i, t, s)} ∪ set D ∪ set Di.
  (∃δ. Unifier δ (pair p) (pair q)) ⟹ j = k"
  by blast
ultimately have "⌊M; ?du2 Di⌋d I" using labeled_sat_ineq_lift by simp
hence "⌊M; unlabel (?dl2 Di)⌋d I" by (metis 0(6))
moreover have "⌊M; unlabel (?cl1 Di)⌋d I" using 7(1) unfolding c unlabel_def by auto
hence "⌊M; unlabel (?dl1 Di)⌋d I" by (metis 0(3,5))
ultimately have "⌊M; unlabel (d Di)⌋d I" using 0(2) unfolding c d unlabel_def by force
hence 8: "⌊M; unlabel (d Di@C)⌋d I" using 0(2) C(2) unfolding unlabel_def by auto

have 9: "d Di@C ∈ set (tr'pc ((i,delete⟨t,s⟩)#S) D)"
  using C(1) dbproj_subseq_in_subseqs[OF B'(1)]
  unfolding d unlabel_def by auto

show ?case by (metis 8 9)

```

```

next
  case (7 i ac t s S D)
  note prems = "7.prems"
  note IH = "7.IH"

  obtain B' d where B':
    "B = (i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#B'"
    "B' ∈ set (trpc S D)" "d ∈ set (dbproj i D)"
    using prems(1) by moura

  have 1: "⌊M; unlabel B'⌋d I" using prems(2) B'(1) by simp

  { fix j p k q
    assume "(j,p) ∈ setopslssst S ∪ set D"
      "(k,q) ∈ setopslssst S ∪ set D"
    hence "(j,p) ∈ setopslssst ((i, InSet ac t s)#S) ∪ set D"
      "(k,q) ∈ setopslssst ((i, InSet ac t s)#S) ∪ set D"
    by (auto simp add: setopslssst_def)
    hence "(∃δ. Unifier δ (pair p) (pair q)) ⇒ j = k" using prems(3) by blast
  } hence 4: "?R3 S D" by blast

  have 5: "?R4 S D" using prems(4) by force
  have 6: "?R5 S D" using prems(5) by force
  have 7: "pair (t,s) · I = pair (snd d) · I" using prems(2) B'(1) by simp

  obtain C where C: "C ∈ set (tr'pc S D)" "⌊M; unlabel C⌋d I"
    using IH[OF B'(2) 1 4 5 6] by moura
  hence "((i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#C) ∈ set (tr'pc ((i, InSet ac t s)#S) D)"
    "⌊M; unlabel ((i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#C)⌋d I"
    using 7 B'(3) by auto
  thus ?case by metis
next
  case (8 i X F F' S D)
  note prems = "8.prems"
  note IH = "8.IH"

  let ?cl = "map (λG. (i,∀X⟨V≠: (F@G)⟩st)) (trpairs F' (map snd (dbproj i D)))"
  let ?cu = "map (λG. ∀X⟨V≠: (F@G)⟩st) (trpairs F' (map snd (dbproj i D)))"

  let ?dl = "map (λG. (i,∀X⟨V≠: (F@G)⟩st)) (trpairs F' (map snd D))"
  let ?du = "map (λG. ∀X⟨V≠: (F@G)⟩st) (trpairs F' (map snd D))"

  define c where c: "c = ?cl"
  define d where d: "d = ?dl"

  obtain B' where B': "B = c@B'" "B' ∈ set (trpc S D)" using prems(1) c by moura

  have 0: "ikst (unlabel c) = {}" "ikst (unlabel d) = {}"
    "unlabel ?cl = ?cu" "unlabel ?dl = ?du"
    unfolding c d unlabel_def by force+

  have "ikst (unlabel c) = {}" unfolding c unlabel_def by force
  hence 1: "⌊M; unlabel B'⌋d I" using prems(2) B'(1) unfolding unlabel_def by auto

  have "setopslssst S ⊆ setopslssst ((i, NegChecks X F F')#S)" by (auto simp add: setopslssst_def)
  hence 4: "?R3 S D" using prems(3) by blast

  have 5: "?R4 S D" using prems(4) by force
  have 6: "?R5 S D" using prems(5) by force

  obtain C where C: "C ∈ set (tr'pc S D)" "⌊M; unlabel C⌋d I"
    using IH[OF B'(2) 1 4 5 6] by moura

```

```

have 7: "[M; unlabel c]_d I" "[M; unlabel B']_d I"
  using prems(2) B'(1) 0(1) strand_sem_split(3,4)[of M "unlabel c" "unlabel B'"]
  unfolding c unlabel_def by auto

have 8: "d@c ∈ set (tr'_pc ((i, NegChecks X F F')#S) D)"
  using C(1) unfolding d unlabel_def by auto

have "[M; unlabel ?cl]_d I" using 7(1) unfolding c unlabel_def by auto
hence "[M; ?cu]_d I" by (metis 0(3))
moreover {
  fix j p k q
  assume "(j, p) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D"
    "(k, q) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D"
  hence "(j, p) ∈ setopslss ((i, NegChecks X F F')#S) ∪ set D"
    "(k, q) ∈ setopslss ((i, NegChecks X F F')#S) ∪ set D"
  by (auto simp add: setopslss_def)
  hence "(∃δ. Unifier δ (pair p) (pair q)) ⇒ j = k" using prems(3) by blast
} hence "∀(j, p) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D.
  ∀(k, q) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D.
  (∃δ. Unifier δ (pair p) (pair q)) ⇒ j = k"
  by blast
moreover have "fvpairs (map snd D) ∩ set X = {}"
  using prems(4) by fastforce
ultimately have "[M; ?du]_d I" using labeled_sat_ineq_dbproj_sem_equiv[of i] by simp
hence "[M; unlabel ?dl]_d I" by (metis 0(4))
hence "[M; unlabel d]_d I" using 0(2) unfolding c d unlabel_def by force
hence 9: "[M; unlabel (d@c)]_d I" using 0(2) C(2) unfolding unlabel_def by auto

show ?case by (metis 8 9)
qed
} thus "?P ⇒ ?Q" by metis

{ fix C assume "C ∈ set (tr'_pc A D)" "[M; unlabel C]_d I"
  hence ?P using assms
  proof (induction A D arbitrary: C M rule: tr'_pc.induct)
    case (1 D) thus ?case by simp
  next
    case (2 i t S D)
    note prems = "2.prems"
    note IH = "2.IH"

    obtain C' where C': "C = (i, send⟨t⟩st)#C'" "C' ∈ set (tr'_pc S D)"
      using prems(1) by mouna

    have 1: "[M; unlabel C']_d I" using prems(2) C'(1) by simp
    have 4: "?R3 S D" using prems(3) by (auto simp add: setopslss_def)
    have 5: "?R4 S D" using prems(4) by force
    have 6: "?R5 S D" using prems(5) by force

    have 7: "M ⊢ t · I" using prems(2) C'(1) by simp

    obtain B where B: "B ∈ set (trpc S D)" "[M; unlabel B]_d I"
      using IH[OF C'(2) 1 4 5 6] by mouna
    hence "((i, send⟨t⟩st)#B) ∈ set (trpc ((i, Send t)#S) D)"
      "[M; unlabel ((i, send⟨t⟩st)#B)]_d I"
      using 7 by auto
    thus ?case by metis
  next
    case (3 i t S D)
    note prems = "3.prems"
    note IH = "3.IH"

    obtain C' where C': "C = (i, receive⟨t⟩st)#C'" "C' ∈ set (tr'_pc S D)"

```

```

using prems(1) by maura

have 1: "[insert (t · I) M; unlabel C']d I " using prems(2) C'(1) by simp
have 4: "?R3 S D" using prems(3) by (auto simp add: setopslsst_def)
have 5: "?R4 S D" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

obtain B where B: "B ∈ set (trpc S D)" "[insert (t · I) M; unlabel B]d I"
  using IH[OF C'(2) 1 4 5 6] by maura
hence "((i, receive⟨t⟩st)#B) ∈ set (trpc ((i, receive⟨t⟩)#S) D)"
  "[insert (t · I) M; unlabel ((i, receive⟨t⟩st)#B)]d I"
  by auto
thus ?case by auto
next
case (4 i ac t t' S D)
note prems = "4.prems"
note IH = "4.IH"

obtain C' where C': "C = (i, ⟨ac: t ≐ t'⟩st)#C'" "C' ∈ set (tr'pc S D)"
  using prems(1) by maura

have 1: "[M; unlabel C']d I " using prems(2) C'(1) by simp
have 4: "?R3 S D" using prems(3) by (auto simp add: setopslsst_def)
have 5: "?R4 S D" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

have 7: "t · I = t' · I" using prems(2) C'(1) by simp

obtain B where B: "B ∈ set (trpc S D)" "[M; unlabel B]d I"
  using IH[OF C'(2) 1 4 5 6] by maura
hence "((i, ⟨ac: t ≐ t'⟩st)#B) ∈ set (trpc ((i, Equality ac t t')#S) D)"
  "[M; unlabel ((i, ⟨ac: t ≐ t'⟩st)#B)]d I"
  using 7 by auto
thus ?case by metis
next
case (5 i t s S D)
note prems = "5.prems"
note IH = "5.IH"

have C: "C ∈ set (tr'pc S (List.insert (i, t, s) D))" using prems(1) by simp

have 1: "[M; unlabel C]d I " using prems(2) C(1) by simp
have 4: "?R3 S (List.insert (i, t, s) D)" using prems(3) by (auto simp add: setopslsst_def)
have 5: "?R4 S (List.insert (i, t, s) D)" using prems(4,5) by force
have 6: "?R5 S (List.insert (i, t, s) D)" using prems(5) by force

show ?case using IH[OF C(1) 1 4 5 6] by simp
next
case (6 i t s S D)
note prems = "6.prems"
note IH = "6.IH"

let ?d11 = "λDi. map (λd. (i, ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st)) Di"
let ?du1 = "λDi. map (λd. ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st) Di"
let ?d12 = "λDi. map (λd. (i, ∀ [] ⟨≠: [(pair (t, s), pair (snd d))]⟩st)) [d ← dbproj i D. d ∉ set
Di]"
let ?du2 = "λDi. map (λd. ∀ [] ⟨≠: [(pair (t, s), pair (snd d))]⟩st) [d ← dbproj i D. d ∉ set Di]"

let ?c11 = "λDi. map (λd. (i, ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st)) Di"
let ?cu1 = "λDi. map (λd. ⟨check: (pair (t, s)) ≐ (pair (snd d))⟩st) Di"
let ?c12 = "λDi. map (λd. (i, ∀ [] ⟨≠: [(pair (t, s), pair (snd d))]⟩st)) [d ← D. d ∉ set Di]"
let ?cu2 = "λDi. map (λd. ∀ [] ⟨≠: [(pair (t, s), pair (snd d))]⟩st) [d ← D. d ∉ set Di]"

```

```

define c where c: "c = (λDi. ?c11 Di@?c12 Di)"
define d where d: "d = (λDi. ?d11 Di@?d12 Di)"

obtain C' Di where C':
  "Di ∈ set (subseqs D)" "C = c Di@C'" "C' ∈ set (tr'_{pc} S [d←D. d ∉ set Di])"
  using prems(1) c by moura

have 0: "ik_{st} (unlabel (c Di)) = {}" "ik_{st} (unlabel (d Di)) = {}"
  "unlabel (?c11 Di) = ?cu1 Di" "unlabel (?c12 Di) = ?cu2 Di"
  "unlabel (?d11 Di) = ?du1 Di" "unlabel (?d12 Di) = ?du2 Di"
  unfolding c d unlabel_def by force+

have 1: "[M; unlabel C']_d I" using prems(2) C'(2) 0(1) unfolding unlabel_def by auto

{ fix j p k q
  assume "(j, p) ∈ setops_{lssst} S ∪ set [d←D. d ∉ set Di]"
    "(k, q) ∈ setops_{lssst} S ∪ set [d←D. d ∉ set Di]"
  hence "(j, p) ∈ setops_{lssst} ((i, delete⟨t,s⟩)#S) ∪ set D"
    "(k, q) ∈ setops_{lssst} ((i, delete⟨t,s⟩)#S) ∪ set D"
    by (auto simp add: setops_{lssst}_def)
  hence "(∃δ. Unifier δ (pair p) (pair q)) ⟹ j = k" using prems(3) by blast
} hence 4: "?R3 S [d←D. d ∉ set Di]" by blast

have 5: "?R4 S (filter (λd. d ∉ set Di) D)" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

obtain B where B: "B ∈ set (tr_{pc} S [d←D. d ∉ set Di])" "[M; unlabel B]_d I"
  using IH[OF C'(1,3) 1 4 5 6] by moura

have 7: "[M; unlabel (c Di)]_d I" "[M; unlabel C']_d I"
  using prems(2) C'(2) 0(1) strand_sem_split(3,4)[of M "unlabel (c Di)" "unlabel C'"]
  unfolding c unlabel_def by auto

{ fix j p k q
  assume "(j, p) ∈ {(i, t, s)} ∪ set D"
    "(k, q) ∈ {(i, t, s)} ∪ set D"
  hence "(j, p) ∈ setops_{lssst} ((i, delete⟨t,s⟩)#S) ∪ set D"
    "(k, q) ∈ setops_{lssst} ((i, delete⟨t,s⟩)#S) ∪ set D"
    by (auto simp add: setops_{lssst}_def)
  hence "(∃δ. Unifier δ (pair p) (pair q)) ⟹ j = k" using prems(3) by blast
} hence "∀(j, p) ∈ {(i, t, s)} ∪ set D.
  ∀(k, q) ∈ {(i, t, s)} ∪ set D.
  (∃δ. Unifier δ (pair p) (pair q)) ⟹ j = k"
  by blast
moreover have "[M; unlabel (?c11 Di)]_d I" using 7(1) unfolding c unlabel_append by auto
hence "[M; ?cu1 Di]_d I" by (metis 0(3))
ultimately have *: "Di ∈ set (subseqs (dbproj i D))"
  using labeled_sat_eqs_subseqs[OF C'(1)] by simp
hence 8: "d Di@B ∈ set (tr_{pc} ((i, delete⟨t,s⟩)#S) D)"
  using B(1) unfolding d unlabel_def by auto

have "[M; unlabel (?c12 Di)]_d I" using 7(1) 0(1) unfolding c unlabel_def by auto
hence "[M; ?cu2 Di]_d I" by (metis 0(4))
hence "[M; ?du2 Di]_d I" by (metis labeled_sat_ineq_dbproj)
hence "[M; unlabel (?d12 Di)]_d I" by (metis 0(6))
moreover have "[M; unlabel (?c11 Di)]_d I" using 7(1) unfolding c unlabel_def by auto
hence "[M; unlabel (?d11 Di)]_d I" by (metis 0(3,5))
ultimately have "[M; unlabel (d Di)]_d I" using 0(2) unfolding c d unlabel_def by force
hence 9: "[M; unlabel (d Di@B)]_d I" using 0(2) B(2) unfolding unlabel_def by auto

show ?case by (metis 8 9)
next
case (7 i ac t s S D)

```

```

note prems = "7.prems"
note IH = "7.IH"

obtain C' d where C':
  "C = (i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#C'"
  "C' ∈ set (tr'pc S D)" "d ∈ set D"
  using prems(1) by moura

have 1: "⌊M; unlabel C'⌋d I " using prems(2) C'(1) by simp

{ fix j p k q
  assume "(j,p) ∈ setopslssst S ∪ set D"
    "(k,q) ∈ setopslssst S ∪ set D"
  hence "(j,p) ∈ setopslssst ((i, InSet ac t s)#S) ∪ set D"
    "(k,q) ∈ setopslssst ((i, InSet ac t s)#S) ∪ set D"
    by (auto simp add: setopslssst_def)
  hence "(∃δ. Unifier δ (pair p) (pair q)) ⇒ j = k" using prems(3) by blast
} hence 4: "?R3 S D" by blast

have 5: "?R4 S D" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

obtain B where B: "B ∈ set (trpc S D)" "⌊M; unlabel B⌋d I"
  using IH[OF C'(2) 1 4 5 6] by moura

have 7: "pair (t,s) · I = pair (snd d) · I" using prems(2) C'(1) by simp

have "(i,t,s) ∈ setopslssst ((i, InSet ac t s)#S) ∪ set D"
  "(fst d, snd d) ∈ setopslssst ((i, InSet ac t s)#S) ∪ set D"
  using C'(3) by (auto simp add: setopslssst_def)
hence "∃δ. Unifier δ (pair (t,s)) (pair (snd d)) ⇒ i = fst d"
  using prems(3) by blast
hence "fst d = i" using 7 by auto
hence 8: "d ∈ set (dbproj i D)" using C'(3) by auto

have 9: "((i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#B) ∈ set (trpc ((i, InSet ac t s)#S) D)"
  using B 8 by auto
have 10: "⌊M; unlabel ((i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#B)⌋d I"
  using B 7 8 by auto

show ?case by (metis 9 10)
next
case (8 i X F F' S D)
note prems = "8.prems"
note IH = "8.IH"

let ?dl = "map (λG. (i,∀X⟨V≠: (F@G)⟩st)) (trpairs F' (map snd (dbproj i D)))"
let ?du = "map (λG. ∀X⟨V≠: (F@G)⟩st) (trpairs F' (map snd (dbproj i D)))"

let ?cl = "map (λG. (i,∀X⟨V≠: (F@G)⟩st)) (trpairs F' (map snd D))"
let ?cu = "map (λG. ∀X⟨V≠: (F@G)⟩st) (trpairs F' (map snd D))"

define c where c: "c = ?cl"
define d where d: "d = ?dl"

obtain C' where C': "C = c@C'" "C' ∈ set (tr'pc S D)" using prems(1) c by moura

have 0: "ikst (unlabel c) = {}" "ikst (unlabel d) = {}"
  "unlabel ?cl = ?cu" "unlabel ?dl = ?du"
  unfolding c d unlabel_def by force+

have "ikst (unlabel c) = {}" unfolding c unlabel_def by force
hence 1: "⌊M; unlabel C'⌋d I " using prems(2) C'(1) unfolding unlabel_def by auto

```



```

have "setopslsst S ⊆ setopslsst ((i, NegChecks X F F')#S)" by (auto simp add: setopslsst_def)
hence 4: "?R3 S D" using prems(3) by blast

have 5: "?R4 S D" using prems(4) by force
have 6: "?R5 S D" using prems(5) by force

obtain B where B: "B ∈ set (trpc S D)" "[M; unlabel B]d I"
  using IH[OF C'(2) 1 4 5 6] by moura

have 7: "[M; unlabel c]d I" "[M; unlabel C']d I"
  using prems(2) C'(1) 0(1) strand_sem_split(3,4)[of M "unlabel c" "unlabel C'"]
  unfolding c_unlabel_def by auto

have 8: "d@B ∈ set (trpc ((i, NegChecks X F F')#S) D)"
  using B(1) unfolding d_unlabel_def by auto

have "[M; unlabel ?c1]d I" using 7(1) unfolding c_unlabel_def by auto
hence "[M; ?cu]d I" by (metis 0(3))
moreover {
  fix j p k q
  assume "(j, p) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D"
    "(k, q) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D"
  hence "(j, p) ∈ setopslsst ((i, NegChecks X F F')#S) ∪ set D"
    "(k, q) ∈ setopslsst ((i, NegChecks X F F')#S) ∪ set D"
    by (auto simp add: setopslsst_def)
  hence "(∃δ. Unifier δ (pair p) (pair q)) ⇒ j = k" using prems(3) by blast
} hence "∀(j, p) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D.
  ∀(k, q) ∈ ((λ(t,s). (i,t,s)) ' set F') ∪ set D.
  (∃δ. Unifier δ (pair p) (pair q)) → j = k"
  by blast
moreover have "fvpairs (map snd D) ∩ set X = {}"
  using prems(4) by fastforce
ultimately have "[M; ?du]d I" using labeled_sat_ineq_dbproj_sem_equiv[of i] by simp
hence "[M; unlabel ?dl]d I" by (metis 0(4))
hence "[M; unlabel d]d I" using 0(2) unfolding c d_unlabel_def by force
hence 9: "[M; unlabel (d@B)]d I" using 0(2) B(2) unfolding unlabel_def by auto

show ?case by (metis 8 9)
qed
} thus "?Q ⇒ ?P" by metis
qed

```

### Part 3

```

private lemma tr'_par_sem_equiv:
  assumes "∀(l,t,s) ∈ set D. (fv t ∪ fv s) ∩ bvarssst (unlabel A) = {}"
  and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}" "ground M"
  and "∀(i,p) ∈ setopslsst A ∪ set D. ∀(j,q) ∈ setopslsst A ∪ set D.
    (∃δ. Unifier δ (pair p) (pair q)) → i = j" (is "?R A D")
  and I: "interpretationsubst I"
  shows "[M; set (unlabel D) ·pset I; unlabel A]s I ↔ (∃B ∈ set (tr'pc A D). [M; unlabel B]d I)"
    (is "?P ↔ ?Q")
proof -
  have 1: "∀(t,s) ∈ set (unlabel D). (fv t ∪ fv s) ∩ bvarssst (unlabel A) = {}"
    using assms(1) unfolding unlabel_def by force

  have 2: "∀(i,p) ∈ setopslsst A ∪ set D. ∀(j,q) ∈ setopslsst A ∪ set D. p = q → i = j"
    using assms(4) subst_apply_term_empty by blast

  show ?thesis by (metis tr_sem_equiv'[OF 1 assms(2,3) I] tr'_par_iff_unlabel_tr[OF 2])
qed

```

## Part 4

```

lemma tr_par_sem_equiv:
  assumes "\(\l, t, s) \in set D. (fv t \cup fv s) \cap bvars_{sst} (unlabel A) = \{\}"
  and "fv_{sst} (unlabel A) \cap bvars_{sst} (unlabel A) = \{\}" "ground M"
  and "\(\i, p) \in setops_{lst} A \cup set D. \(\j, q) \in setops_{lst} A \cup set D.
    (\exists \delta. Unifier \delta (pair p) (pair q)) \longrightarrow i = j"
  and \mathcal{I}: "interpretation_{subst} \mathcal{I}"
  shows "\(\llbracket M; \text{unlabel } D \rrbracket_{pset} \mathcal{I}; \text{unlabel } A \rrbracket_s \mathcal{I} \longleftrightarrow (\exists B \in set (tr_{pc} A D). \llbracket M; \text{unlabel } B \rrbracket_d \mathcal{I})"
    (is "?P \longleftrightarrow ?Q")
using tr_par_iff_tr'_par[OF assms(4,1,2), of M \mathcal{I}] tr'_par_sem_equiv[OF assms] by metis
end

```

## 6.2.6 Theorem: The Stateful Compositionality Result, on the Constraint Level

```

theorem par_comp_constr_stateful:
  assumes \mathcal{A}: "par_comp_{lst} \mathcal{A} Sec" "typing_cond_{sst} (unlabel \mathcal{A})"
  and \mathcal{I}: "\mathcal{I} \models_s \text{unlabel } \mathcal{A}" "interpretation_{subst} \mathcal{I}"
  shows "\(\exists \mathcal{I}_\tau. interpretation_{subst} \mathcal{I}_\tau \wedge wt_{subst} \mathcal{I}_\tau \wedge wf_{trms} (subst\_range \mathcal{I}_\tau) \wedge (\mathcal{I}_\tau \models_s \text{unlabel } \mathcal{A}) \wedge
    ((\forall n. \mathcal{I}_\tau \models_s \text{proj\_unl } n \mathcal{A}) \vee (\exists \mathcal{A}'. \text{prefix } \mathcal{A}' \mathcal{A} \wedge (\mathcal{A}' \text{ leaks Sec under } \mathcal{I}_\tau)))"
proof -
  let ?P = "\(\lambda n A D.
    \(\i, p) \in setops_{lst} (\text{proj } n A) \cup set D.
    \(\j, q) \in setops_{lst} (\text{proj } n A) \cup set D.
    (\exists \delta. Unifier \delta (pair p) (pair q)) \longrightarrow i = j"

  have 1: "\(\forall (\l, t, t') \in set []. (fv t \cup fv t') \cap bvars_{sst} (unlabel \mathcal{A}) = \{\}"
    "fv_{sst} (unlabel \mathcal{A}) \cap bvars_{sst} (unlabel \mathcal{A}) = \{\}" "ground \{\}"
    using \mathcal{A}(2) unfolding typing_cond_{sst}_def by simp_all

  have 2: "\(\bigwedge n. \(\forall (\l, t, t') \in set []. (fv t \cup fv t') \cap bvars_{sst} (\text{proj\_unl } n \mathcal{A}) = \{\}"
    "\(\bigwedge n. fv_{sst} (\text{proj\_unl } n \mathcal{A}) \cap bvars_{sst} (\text{proj\_unl } n \mathcal{A}) = \{\}"
    using 1(1,2) sst_vars_proj_subset[of _ \mathcal{A}] by fast+

  have 3: "\(\bigwedge n. par_comp_{lst} (\text{proj } n \mathcal{A}) Sec"
    using par_comp_{lst}_proj[OF \mathcal{A}(1)] by metis

  have 4:
    "\(\llbracket \{\}; set (\text{unlabel } []) \rrbracket_{pset} \mathcal{I}'; \text{unlabel } \mathcal{A} \rrbracket_s \mathcal{I}' \longleftrightarrow
    (\exists B \in set (tr_{pc} \mathcal{A} []). \llbracket \{\}; \text{unlabel } B \rrbracket_d \mathcal{I}'))"
    when \mathcal{I}': "interpretation_{subst} \mathcal{I}'" for \mathcal{I}'
    using tr_par_sem_equiv[OF 1 _ \mathcal{I}'] \mathcal{A}(1)
    unfolding par_comp_{lst}_def constr_sem_d_def by auto

  obtain \mathcal{A}' where \mathcal{A}': "\mathcal{A}' \in set (tr_{pc} \mathcal{A} [])" "\mathcal{I} \models \langle \text{unlabel } \mathcal{A}' \rangle"
    using 4[OF \mathcal{I}(2)] \mathcal{I}(1) unfolding constr_sem_d_def by maura

  obtain \mathcal{I}_\tau where \mathcal{I}_\tau:
    "interpretation_{subst} \mathcal{I}_\tau" "wt_{subst} \mathcal{I}_\tau" "wf_{trms} (subst\_range \mathcal{I}_\tau)" "\mathcal{I}_\tau \models \langle \text{unlabel } \mathcal{A}' \rangle"
    "(\forall n. (\mathcal{I}_\tau \models \langle \text{proj\_unl } n \mathcal{A}' \rangle)) \vee (\exists \mathcal{A}''. \text{prefix } \mathcal{A}'' \mathcal{A}' \wedge (\text{strand\_leaks}_{lst} \mathcal{A}'' Sec \mathcal{I}_\tau))"
    using par_comp_constr[OF tr_par_preserves_par_comp[OF \mathcal{A}(1) \mathcal{A}'(1)]
      tr_par_preserves_typing_cond[OF \mathcal{A} \mathcal{A}'(1)]
      \mathcal{A}'(2) \mathcal{I}(2)]
    by maura

  have \mathcal{I}_\tau': "\mathcal{I}_\tau \models_s \text{unlabel } \mathcal{A}" using 4[OF \mathcal{I}_\tau(1)] \mathcal{A}'(1) \mathcal{I}_\tau(4) unfolding constr_sem_d_def by auto

  show ?thesis
proof (cases "\(\forall n. (\mathcal{I}_\tau \models \langle \text{proj\_unl } n \mathcal{A}' \rangle)")
  case True
  { fix n assume "\mathcal{I}_\tau \models \langle \text{proj\_unl } n \mathcal{A}' \rangle"
    hence "\(\llbracket \{\}; \{\}; \text{unlabel } (\text{proj } n \mathcal{A}) \rrbracket_s \mathcal{I}_\tau"

```

```

using tr_par_proj[OF A'(1), of n]
tr_par_sem_equiv[OF 2(1,2) 1(3) _ Iτ(1), of n] 3(1)
unfolding par_complssst_def proj_def constr_sem_d_def by force
} thus ?thesis using True Iτ(1,2,3) Iτ' by metis
next
case False
then obtain A''::('fun,'var,'lbl) labeled_strand" where A'':
  "prefix A'' A'' "strand_leakslst A'' Sec Iτ"
  using Iτ by blast
moreover {
  fix t l assume *: "[{}; unlabel (proj l A'')@[send⟨t⟩st]]d Iτ"
  have "Iτ ⊨ ⟨unlabel (proj l A'')⟩" "ikst (unlabel (proj l A'')) ·set Iτ ⊢ t · Iτ"
  using strand_sem_split(3,4)[OF *] unfolding constr_sem_d_def by auto
} ultimately have "∃ t ∈ Sec - declassifiedlst A'' Iτ. ∃ l.
  (Iτ ⊨ ⟨unlabel (proj l A'')⟩) ∧ ikst (unlabel (proj l A'')) ·set Iτ ⊢ t · Iτ"
  unfolding strand_leakslst_def constr_sem_d_def by metis
then obtain s m where sm:
  "s ∈ Sec - declassifiedlst A'' Iτ"
  "Iτ ⊨ ⟨unlabel (proj m A'')⟩"
  "ikst (unlabel (proj m A'')) ·set Iτ ⊢ s · Iτ"
  by maura

```

— We now need to show that there is some prefix  $B$  of  $A''$  that also leaks and where  $B \in \text{set } (\text{tr } C \ D)$  for some prefix  $C$  of  $A$

```

obtain B::('fun,'var,'lbl) labeled_strand"
  and C::('fun,'var,'lbl) labeled_stateful_strand"
  where BC:
    "prefix B A'' "prefix C A" "B ∈ set (trpc C [])"
    "ikst (unlabel (proj m B)) ·set Iτ ⊢ s · Iτ"
    "prefix B A''"
  using tr_leaking_prefix_exists[OF A'(1) A''(1) sm(3)] prefix_order.order_trans[OF _ A''(1)]
  by auto
have "[{}; unlabel (proj m B)]d Iτ"
  using sm(2) BC(5) unfolding prefix_def unlabel_def proj_def constr_sem_d_def by auto
hence BC': "Iτ ⊨ ⟨proj_unl m B@[send⟨s⟩st]]"
  using BC(4) unfolding constr_sem_d_def by auto
have BC'': "s ∈ Sec - declassifiedlst B Iτ"
  using BC(5) sm(1) unfolding prefix_def declassifiedlst_def by auto
have 5: "par_complssst (proj n C) Sec" for n
  using A(1) BC(2) par_complssst_split(1)[THEN par_complssst_proj]
  unfolding prefix_def by auto
have "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}"
  "fvsst (unlabel C) ⊆ fvsst (unlabel A)"
  "bvarssst (unlabel C) ⊆ bvarssst (unlabel A)"
  using A(2) BC(2) sst_vars_append_subset(1,2)[of "unlabel C"]
  unfolding typing_condsst_def prefix_def unlabel_def by auto
hence "fvsst (proj_unl n C) ∩ bvarssst (proj_unl n C) = {}" for n
  using sst_vars_proj_subset[of _ C] sst_vars_proj_subset[of _ A]
  by blast
hence 6:
  "∀ (l, t, t') ∈ set []. (fv t ∪ fv t') ∩ bvarssst (proj_unl n C) = {}"
  "fvsst (proj_unl n C) ∩ bvarssst (proj_unl n C) = {}"
  "ground {}"
  for n
  using 2 by auto
have 7: "?P n C []" for n using 5 unfolding par_complssst_def by simp
have "s · Iτ = s" using Iτ(1) BC'' A(1) unfolding par_complssst_def by auto
hence "∃ n. (Iτ ⊨s proj_unl n C) ∧ iksst (proj_unl n C) ·set Iτ ⊢ s · Iτ"
  using tr_par_proj[OF BC(3), of m] BC'(1)
  tr_par_sem_equiv[OF 6 7 Iτ(1), of m]
  tr_par_deduct_iff[OF tr_par_proj(1)[OF BC(3)], of Iτ m s]
  unfolding proj_def constr_sem_d_def by auto
hence "∃ n. Iτ ⊨s (proj_unl n C@[Send s])" using strand_sem_append_stateful by simp

```

moreover have " $s \in \text{Sec} - \text{declassified}_{l_{sst}} C \mathcal{I}_\tau$ " by (metis tr\_par\_declassified\_eq BC(3) BC'')  
 ultimately show ?thesis using  $\mathcal{I}_\tau(1,2,3) \mathcal{I}_\tau' BC(2)$  unfolding strand\_leaks $_{l_{sst}}$ \_def by metis  
 qed  
 qed

### 6.2.7 Theorem: The Stateful Compositionality Result, on the Protocol Level

abbreviation  $wf_{l_{sst}}$  where

" $wf_{l_{sst}} V \mathcal{A} \equiv wf'_{sst} V (\text{unlabel } \mathcal{A})$ "

We state our result on the level of protocol traces (i.e., the constraints reachable in a symbolic execution of the actual protocol). Hence, we do not need to convert protocol strands to intruder constraints in the following well-formedness definitions.

definition  $wf_{l_{sst}}$  :: "(fun, 'var, 'lbl) labeled\_stateful\_strand set  $\Rightarrow$  bool" where

" $wf_{l_{sst}} S \equiv (\forall \mathcal{A} \in S. wf_{l_{sst}} \{\} \mathcal{A}) \wedge (\forall \mathcal{A} \in S. \forall \mathcal{A}' \in S. fv_{l_{sst}} \mathcal{A} \cap bvars_{l_{sst}} \mathcal{A}' = \{\})$ "

definition  $wf_{l_{sst}}'$  ::

"(fun, 'var, 'lbl) labeled\_stateful\_strand set  $\Rightarrow$  (fun, 'var, 'lbl) labeled\_stateful\_strand  $\Rightarrow$  bool"

where

" $wf_{l_{sst}}' S \mathcal{A} \equiv (\forall \mathcal{A}' \in S. wf'_{sst} (wf_{restrictedvars}_{l_{sst}} \mathcal{A}) (\text{unlabel } \mathcal{A}')) \wedge$   
 $(\forall \mathcal{A}' \in S. \forall \mathcal{A}'' \in S. fv_{l_{sst}} \mathcal{A}' \cap bvars_{l_{sst}} \mathcal{A}'' = \{\}) \wedge$   
 $(\forall \mathcal{A}' \in S. fv_{l_{sst}} \mathcal{A}' \cap bvars_{l_{sst}} \mathcal{A} = \{\}) \wedge$   
 $(\forall \mathcal{A}' \in S. fv_{l_{sst}} \mathcal{A} \cap bvars_{l_{sst}} \mathcal{A}' = \{\})$ "

definition typing\_cond\_prot\_stateful where

"typing\_cond\_prot\_stateful  $\mathcal{P} \equiv$   
 $wf_{l_{sst}} \mathcal{P} \wedge$   
 $tfr_{set} (\bigcup (trms_{l_{sst}} ' \mathcal{P}) \cup \text{pair } ' \bigcup (\text{setops}_{sst} ' \text{unlabel } ' \mathcal{P})) \wedge$   
 $wf_{trms} (\bigcup (trms_{l_{sst}} ' \mathcal{P})) \wedge$   
 $(\forall S \in \mathcal{P}. \text{list\_all } tfr_{sstp} (\text{unlabel } S))$ "

definition par\_comp\_prot\_stateful where

"par\_comp\_prot\_stateful  $\mathcal{P} \text{ Sec} \equiv$   
 $(\forall l1 \ l2. l1 \neq l2 \longrightarrow$   
 $\text{GSMP\_disjoint } (\bigcup \mathcal{A} \in \mathcal{P}. trms_{sst} (\text{proj\_unl } l1 \ \mathcal{A}) \cup \text{pair } ' \text{setops}_{sst} (\text{proj\_unl } l1 \ \mathcal{A}))$   
 $(\bigcup \mathcal{A} \in \mathcal{P}. trms_{sst} (\text{proj\_unl } l2 \ \mathcal{A}) \cup \text{pair } ' \text{setops}_{sst} (\text{proj\_unl } l2 \ \mathcal{A})) \text{ Sec}) \wedge$   
 $\text{ground Sec} \wedge (\forall s \in \text{Sec}. \forall s' \in \text{subterms } s. \{\} \vdash_c s' \vee s' \in \text{Sec}) \wedge$   
 $(\forall (i,p) \in \bigcup \mathcal{A} \in \mathcal{P}. \text{setops}_{l_{sst}} \mathcal{A}. \forall (j,q) \in \bigcup \mathcal{A} \in \mathcal{P}. \text{setops}_{l_{sst}} \mathcal{A}. \rightarrow$   
 $(\exists \delta. \text{Unifier } \delta (\text{pair } p) (\text{pair } q)) \longrightarrow i = j) \wedge$   
 $\text{typing\_cond\_prot\_stateful } \mathcal{P}$ "

definition component\_secure\_prot\_stateful where

"component\_secure\_prot\_stateful  $n \ P \ \text{Sec} \ \text{attack} \equiv$   
 $(\forall \mathcal{A} \in P. \text{suffix } [(\text{ln } n, \text{Send } (\text{Fun } \text{attack } []))]) \mathcal{A} \longrightarrow$   
 $(\forall \mathcal{I}_\tau. (\text{interpretation}_{subst} \mathcal{I}_\tau \wedge wt_{subst} \mathcal{I}_\tau \wedge wf_{trms} (\text{subst\_range } \mathcal{I}_\tau)) \longrightarrow$   
 $\neg (\mathcal{I}_\tau \models_s (\text{proj\_unl } n \ \mathcal{A})) \wedge$   
 $(\forall \mathcal{A}'. \text{prefix } \mathcal{A}' \ \mathcal{A} \longrightarrow$   
 $(\forall t \in \text{Sec-declassified}_{l_{sst}} \mathcal{A}' \ \mathcal{I}_\tau. \neg (\mathcal{I}_\tau \models_s (\text{proj\_unl } n \ \mathcal{A}' @ [\text{Send } t])))$ "

definition component\_leaks\_stateful where

"component\_leaks\_stateful  $n \ \mathcal{A} \ \text{Sec} \equiv$   
 $(\exists \mathcal{A}' \ \mathcal{I}_\tau. \text{interpretation}_{subst} \mathcal{I}_\tau \wedge wt_{subst} \mathcal{I}_\tau \wedge wf_{trms} (\text{subst\_range } \mathcal{I}_\tau) \wedge \text{prefix } \mathcal{A}' \ \mathcal{A} \wedge$   
 $(\exists t \in \text{Sec-declassified}_{l_{sst}} \mathcal{A}' \ \mathcal{I}_\tau. (\mathcal{I}_\tau \models_s (\text{proj\_unl } n \ \mathcal{A}' @ [\text{Send } t])))$ "

definition unsat\_stateful where

"unsat\_stateful  $\mathcal{A} \equiv (\forall \mathcal{I}. \text{interpretation}_{subst} \mathcal{I} \longrightarrow \neg (\mathcal{I} \models_s \text{unlabel } \mathcal{A}))$ "

lemma  $wf_{l_{sst}} \text{eqs\_} wf_{l_{sst}}'$  [simp]: " $wf_{l_{sst}} S = wf_{l_{sst}}' S []$ "

unfolding  $wf_{l_{sst}}$ \_def  $wf_{l_{sst}}'$ \_def unlabeled\_def  $wf_{restrictedvars}_{sst}$ \_def by simp

lemma par\_comp\_prot\_impl\_par\_comp\_stateful:

assumes "par\_comp\_prot\_stateful  $\mathcal{P} \ \text{Sec}$ " " $\mathcal{A} \in \mathcal{P}$ "

shows "par\_comp $_{l_{sst}} \mathcal{A} \ \text{Sec}$ "

```

proof -
  have *:
    "∀ l1 l2. l1 ≠ l2 →
      GSMP_disjoint (⋃ A ∈ P. trmssst (proj_unl l1 A) ∪ pair ' setopssst (proj_unl l1 A))
        (⋃ A ∈ P. trmssst (proj_unl l2 A) ∪ pair ' setopssst (proj_unl l2 A)) Sec"
    using assms(1) unfolding par_comp_prot_stateful_def by argo
  { fix l1 l2::'lbl assume **: "l1 ≠ l2"
    hence ***:
      "GSMP_disjoint (⋃ A ∈ P. trmssst (proj_unl l1 A) ∪ pair ' setopssst (proj_unl l1 A))
        (⋃ A ∈ P. trmssst (proj_unl l2 A) ∪ pair ' setopssst (proj_unl l2 A)) Sec"
      using * by auto
    have "GSMP_disjoint (trmssst (proj_unl l1 A) ∪ pair ' setopssst (proj_unl l1 A))
      (trmssst (proj_unl l2 A) ∪ pair ' setopssst (proj_unl l2 A)) Sec"
      using GSMP_disjoint_subset[OF ***] assms(2) by auto
  } hence "∀ l1 l2. l1 ≠ l2 →
    GSMP_disjoint (trmssst (proj_unl l1 A) ∪ pair ' setopssst (proj_unl l1 A))
      (trmssst (proj_unl l2 A) ∪ pair ' setopssst (proj_unl l2 A)) Sec"

    by metis
  moreover have "∀ (i,p) ∈ setopslsst A. ∀ (j,q) ∈ setopslsst A.
    (∃ δ. Unifier δ (pair p) (pair q)) → i = j"
    using assms(1,2) unfolding par_comp_prot_stateful_def by blast
  ultimately show ?thesis
    using assms
    unfolding par_comp_prot_stateful_def par_complsst_def
    by fast
qed

lemma typing_cond_prot_impl_typing_cond_stateful:
  assumes "typing_cond_prot_stateful P" "A ∈ P"
  shows "typing_condsst (unlabel A)"
proof -
  have 1: "wf'sst {} (unlabel A)" "fvlsst A ∩ bvarslsst A = {}"
    using assms unfolding typing_cond_prot_stateful_def wflsst_def by auto

  have "tfrset (⋃ (trmslsst ' P) ∪ pair ' ⋃ (setopssst ' unlabel ' P))"
    "wfttrms (⋃ (trmslsst ' P))"
    "trmslsst A ⊆ ⋃ (trmslsst ' P)"
    "SMP (trmslsst A ∪ pair ' setopssst (unlabel A)) - Var'V ⊆
      SMP (⋃ (trmslsst ' P) ∪ pair ' ⋃ (setopssst ' unlabel ' P)) - Var'V"
    using assms SMP_mono[of "trmslsst A ∪ pair ' setopssst (unlabel A)"
      "⋃ (trmslsst ' P) ∪ pair ' ⋃ (setopssst ' unlabel ' P)"]
    unfolding typing_cond_prot_stateful_def
    by (metis, metis, auto)
  hence 2: "tfrset (trmslsst A ∪ pair ' setopssst (unlabel A))" and 3: "wfttrms (trmslsst A)"
    unfolding tfrset_def by (meson subsetD)+

  have 4: "list_all tfrsstp (unlabel A)" using assms unfolding typing_cond_prot_stateful_def by auto

  show ?thesis using 1 2 3 4 unfolding typing_condsst_def tfrsst_def by blast
qed

theorem par_comp_constr_prot_stateful:
  assumes P: "P = composed_prot Pi" "par_comp_prot_stateful P Sec" "∀ n. component_prot n (Pi n)"
  and left_secure: "component_secure_prot_stateful n (Pi n) Sec attack"
  shows "∀ A ∈ P. suffix [(ln n, Send (Fun attack []))] A →
    unsat_stateful A ∨ (∃ m. n ≠ m ∧ component_leaks_stateful m A Sec)"
proof -
  { fix A A' assume A: "A = A'@[ln n, Send (Fun attack [])]" "A ∈ P"
    let ?P = "∃ A' Iτ. interpretationsubst Iτ ∧ wtsubst Iτ ∧ wfttrms (subst_range Iτ) ∧ prefix A' A"
    ∧
      (∃ t ∈ Sec-declassifiedlsst A' Iτ. ∃ m. n ≠ m ∧ (Iτ ⊨s (proj_unl m A'@[Send
t]))))"
    have tcp: "typing_cond_prot_stateful P" using P(2) unfolding par_comp_prot_stateful_def by simp
  }

```

```

have par_comp: "par_complsst A Sec" "typing_condsst (unlabel A)"
  using par_comp_prot_impl_par_comp_stateful[OF P(2) A(2)]
  typing_cond_prot_impl_typing_cond_stateful[OF tcp A(2)]
  by metis+

have "unlabel (proj n A) = proj_unl n A" "proj_unl n A = proj_unl n (proj n A)"
  "∧ A. A ∈ Pi n ⇒ proj n A = A"
  "proj n A = (proj n A')@[ln n, Send (Fun attack [])]"
  using P(1,3) A by (auto simp add: proj_def unlabel_def component_prot_def composed_prot_def)
moreover have "proj n A ∈ Pi n"
  using P(1) A unfolding composed_prot_def by blast
moreover {
  fix A assume "prefix A A"
  hence *: "prefix (proj n A) (proj n A)" unfolding proj_def prefix_def by force
  hence "proj_unl n A = proj_unl n (proj n A)"
    "∀ I. declassifiedlsst A I = declassifiedlsst (proj n A) I"
    unfolding proj_def declassifiedlsst_def by auto
  hence "∃ B. prefix B (proj n A) ∧ proj_unl n A = proj_unl n B ∧
    (∀ I. declassifiedlsst A I = declassifiedlsst B I)"
    using * by metis
}
ultimately have *:
  "∀ Iτ. interpretationsubst Iτ ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ) →
    ¬(Iτ ⊨s (proj_unl n A)) ∧ (∀ A'. prefix A' A →
      (∀ t ∈ Sec - declassifiedlsst A' Iτ. ¬(Iτ ⊨s (proj_unl n A'@[Send t]))))"
  using left_secure
  unfolding component_secure_prot_stateful_def composed_prot_def suffix_def
  by metis
{ fix I assume I: "interpretationsubst I" "I ⊨s unlabel A"
  obtain Iτ where Iτ:
    "interpretationsubst Iτ" "wtsubst Iτ" "wftrms (subst_range Iτ)"
    "∃ A'. prefix A' A ∧ (A' leaks Sec under Iτ)"
    using par_comp_constr_stateful[OF par_comp I(2,1)] * by moura
  hence "∃ A'. prefix A' A ∧ (∃ t ∈ Sec - declassifiedlsst A' Iτ. ∃ m.
    n ≠ m ∧ (Iτ ⊨s (proj_unl m A'@[Send t])))"
    using Iτ(4) * unfolding strand_leakslsst_def by metis
  hence ?P using Iτ(1,2,3) by auto
} hence "unsat_stateful A ∨ (∃ m. n ≠ m ∧ component_leaks_stateful m A Sec)"
  by (metis unsat_stateful_def component_leaks_stateful_def)
} thus ?thesis unfolding suffix_def by metis
qed

end

```

### 6.2.8 Automated Compositionality Conditions

definition comp\_GSMP\_disjoint where

```

"comp_GSMP_disjoint public arity Ana Γ A' B' A B C ≡
  let Bδ = B ·list var_rename (max_var_set (fvset (set A)))
  in has_all_wt_instances_of Γ (set A') (set A) ∧
    has_all_wt_instances_of Γ (set B') (set Bδ) ∧
    finite_SMP_representation arity Ana Γ A ∧
    finite_SMP_representation arity Ana Γ Bδ ∧
    (∀ t ∈ set A. ∀ s ∈ set Bδ. Γ t = Γ s ∧ mgu t s ≠ None →
      (intruder_synth' public arity {} t ∧ intruder_synth' public arity {} s) ∨
      (∃ u ∈ set C. is_wt_instance_of_cond Γ t u) ∧ (∃ u ∈ set C. is_wt_instance_of_cond Γ s u))"

```

definition comp\_par\_comp<sub>lsst</sub> where

```

"comp_par_complsst public arity Ana Γ pair_fun A M C ≡
  let L = remdups (map (the_LabelN ∘ fst) (filter (Not ∘ is_LabelS) A));
  MPO = λB. remdups (trms_listsst B@map (pair' pair_fun) (setops_listsst B));
  pr = λl. MPO (proj_unl l A)
  in length L > 1 ∧

```

```

list_all (wftrm ' arity) (MPO (unlabel A)) ∧
list_all (wftrm ' arity) C ∧
has_all_wt_instances_of Γ (subtermsset (set C)) (set C) ∧
is_TComp_var_instance_closed Γ C ∧
(∀ i ∈ set L. ∀ j ∈ set L. i ≠ j →
  comp_GSMP_disjoint public arity Ana Γ (pr i) (pr j) (M i) (M j) C) ∧
(∀ (i,p) ∈ setopslsst A. ∀ (j,q) ∈ setopslsst A. i ≠ j →
  (let s = pair' pair_fun p; t = pair' pair_fun q
    in mgu s (t · var_rename (max_var s)) = None))"

locale labeled_stateful_typed_model' =
  stateful_typed_model' arity public Ana Γ Pair
+ labeled_typed_model' arity public Ana Γ label_witness1 label_witness2
  for arity::"'fun ⇒ nat"
  and public::"'fun ⇒ bool"
  and Ana::"'(fun,('fun,'atom::finite) term_type × nat)) term
    ⇒ (('fun,('fun,'atom) term_type × nat)) term list
    × ('fun,('fun,'atom) term_type × nat)) term list)"
  and Γ::"'(fun,('fun,'atom) term_type × nat)) term ⇒ ('fun,'atom) term_type"
  and Pair::"'fun"
  and label_witness1::"'lbl"
  and label_witness2::"'lbl"
begin

sublocale labeled_stateful_typed_model
by unfold_locales

lemma GSMP_disjoint_if_comp_GSMP_disjoint:
  defines "f ≡ λM. {t · δ | t δ. t ∈ M ∧ wtsubst δ ∧ wftrms (subst_range δ) ∧ fv (t · δ) = {}}"
  assumes AB'_wf: "list_all (wftrm ' arity) A'" "list_all (wftrm ' arity) B'"
  and C_wf: "list_all (wftrm ' arity) C"
  and AB'_disj: "comp_GSMP_disjoint public arity Ana Γ A' B' A B C"
  shows "GSMP_disjoint (set A') (set B') ((f (set C)) - {m. {} ⊢c m})"
using GSMP_disjointI[of A' B' A B] AB'_wf AB'_disj C_wf
unfolding comp_GSMP_disjoint_def f_def wftrm_code list_all_iff Let_def by fast

lemma par_complsst_if_comp_par_complsst:
  defines "f ≡ λM. {t · δ | t δ. t ∈ M ∧ wtsubst δ ∧ wftrms (subst_range δ) ∧ fv (t · δ) = {}}"
  assumes A: "comp_par_complsst public arity Ana Γ Pair A M C"
  shows "par_complsst A ((f (set C)) - {m. {} ⊢c m})"
proof (unfold par_complsst_def; intro conjI)
  let ?Sec = "(f (set C)) - {m. {} ⊢c m}"
  let ?L = "remdups (map (the_LabelN ∘ fst) (filter (Not ∘ is_LabelS) A))"
  let ?N1 = "λB. remdups (trms_listsst B @ map (pair' Pair) (setops_listsst B))"
  let ?N2 = "λB. trmssst B ∪ pair ' setopssst B"
  let ?pr = "λl. ?N1 (proj_unl l A)"
  let ?α = "λp. var_rename (max_var (pair p))"

  have 0:
    "length ?L > 1"
    "list_all (wftrm ' arity) (?N1 (unlabel A))"
    "list_all (wftrm ' arity) C"
    "has_all_wt_instances_of Γ (subtermsset (set C)) (set C)"
    "is_TComp_var_instance_closed Γ C"
    "∀ i ∈ set ?L. ∀ j ∈ set ?L. i ≠ j →
      comp_GSMP_disjoint public arity Ana Γ (?pr i) (?pr j) (M i) (M j) C"
    "∀ (i,p) ∈ setopslsst A. ∀ (j,q) ∈ setopslsst A. i ≠ j → mgu (pair p) (pair q · ?α p) = None"
  using A unfolding comp_par_complsst_def pair_code by meson+

  have L_in_iff: "l ∈ set ?L ↔ (∃ a ∈ set A. is_LabelN l a)" for l by force

  have A_wf_trms: "wftrms (trmslsst A ∪ pair ' setopssst (unlabel A))"
  using 0(2)

```

```

    unfolding pair_code wf_trm_code list_all_iff trms_list_sst_is_trms_sst setops_list_sst_is_setops_sst
  by auto
hence A_proj_wf_trms: "wf_trms (trms_lsst (proj 1 A)  $\cup$  pair ' setops_sst (proj_unl 1 A))" for 1
  using trms_sst_proj_subset(1)[of 1 A] setops_sst_proj_subset(1)[of 1 A] by blast
hence A_proj_wf_trms': "list_all (wf_trm' arity) (?N1 (proj_unl 1 A))" for 1
  unfolding pair_code wf_trm_code list_all_iff trms_list_sst_is_trms_sst setops_list_sst_is_setops_sst
  by auto

note C_wf_trms = 0(3)[unfolded list_all_iff wf_trm_code[symmetric]]

note 1 = has_all_wt_instances_ofD'[OF wf_trms_subterms[OF C_wf_trms] C_wf_trms 0(4)]

have 2: "GSMP (?N2 (proj_unl 1 A))  $\subseteq$  GSMP (?N2 (proj_unl 1' A))" when "1  $\notin$  set ?L" for 1 1'
  using that L_in_iff GSMP_mono[of "?N2 (proj_unl 1 A)" "?N2 (proj_unl 1' A)"]
    trms_sst_unlabel_subset_if_no_label[of 1 A]
    setops_sst_unlabel_subset_if_no_label[of 1 A]
  unfolding list_ex_iff by fast

have 3: "GSMP_disjoint (?N2 (proj_unl 11 A)) (?N2 (proj_unl 12 A)) ?Sec"
  when "11  $\in$  set ?L" "12  $\in$  set ?L" "11  $\neq$  12" for 11 12
proof -
  have "GSMP_disjoint (set (?N1 (proj_unl 11 A))) (set (?N1 (proj_unl 12 A))) ?Sec"
  using 0(6) that
    GSMP_disjoint_if_comp_GSMP_disjoint[
      OF A_proj_wf_trms'[of 11] A_proj_wf_trms'[of 12] 0(3),
      of "M 11" "M 12"]
  unfolding f_def by blast
  thus ?thesis
  unfolding pair_code trms_list_sst_is_trms_sst setops_list_sst_is_setops_sst
  by simp
qed

obtain a1 a2 where a: "a1  $\in$  set ?L" "a2  $\in$  set ?L" "a1  $\neq$  a2"
  using remdups_ex2[OF 0(1)] by moura

show "ground ?Sec" unfolding f_def by fastforce

{ fix i p j q
  assume p: "(i,p)  $\in$  setops_lsst A" and q: "(j,q)  $\in$  setops_lsst A"
  and pq: " $\exists \delta$ . Unifier  $\delta$  (pair p) (pair q)"

  have " $\exists \delta$ . Unifier  $\delta$  (pair p) (pair q  $\cdot$  ? $\alpha$  p)"
  using pq vars_term_disjoint_imp_unifier[OF var_rename_fv_disjoint[of "pair p"], of _ "pair q"]
  by (metis (no_types, lifting) subst_subst_compose var_rename_inv_comp)
  hence "i = j" using 0(7) mgu_None_is_subst_neq[of "pair p" "pair q  $\cdot$  ? $\alpha$  p"] p q by fast
} thus " $\forall (i,p) \in \text{setops}_{lsst} A. \forall (j,q) \in \text{setops}_{lsst} A. (\exists \delta. \text{Unifier } \delta \text{ (pair p) (pair q)}) \longrightarrow i = j$ "
  by blast

show " $\forall 11\ 12. 11 \neq 12 \longrightarrow \text{GSMP\_disjoint} (?N2 (\text{proj\_unl } 11\ A)) (?N2 (\text{proj\_unl } 12\ A)) ?Sec$ "
  using 2 3 3[OF a] unfolding GSMP_disjoint_def by blast

show " $\forall s \in ?Sec. \forall s' \in \text{subterms } s. \{ \} \vdash_c s' \vee s' \in ?Sec$ "
proof (intro ballI)
  fix s s'
  assume s: "s  $\in$  ?Sec" and s': "s'  $\sqsubseteq$  s"
  then obtain t  $\delta$  where t: "t  $\in$  set C" "s = t  $\cdot$   $\delta$ " "fv s = { }" " $\neg \{ \} \vdash_c s$ "
  and  $\delta$ : "wt_subst  $\delta$ " "wf_trms (subst_range  $\delta$ )"
  unfolding f_def by blast

  obtain m  $\vartheta$  where m: "m  $\in$  set C" "s' = m  $\cdot$   $\vartheta$ " and  $\vartheta$ : "wt_subst  $\vartheta$ " "wf_trms (subst_range  $\vartheta$ )"
  using TComp_var_and_subterm_instance_closed_has_subterms_instances[
    OF 0(5,4) C_wf_trms in_subterms_Union[OF t(1)] s'[unfolded t(2)]  $\delta$ ]
  by blast

```



```

thus "{ } ⊢c s' ∨ s' ∈ ?Sec"
  using ground_subterm[OF t(3) s']
  unfolding f_def by blast
qed
qed

lemma par_complssst_if_comp_par_complssst':
  defines "f ≡ λM. {t · δ | t · δ. t ∈ M ∧ wtsubst δ ∧ wftrms (subst_range δ) ∧ fv (t · δ) = { }}"
  assumes a: "comp_par_complssst public arity Ana Γ Pair A M C"
  and B: "∀b ∈ set B. ∃a ∈ set A. ∃δ. b = a ·lssst δ ∧ wtsubst δ ∧ wftrms (subst_range δ)"
  (is "∀b ∈ set B. ∃a ∈ set A. ∃δ. b = a ·lssst δ ∧ ?D δ")
  shows "par_complssst B ((f (set C)) - {m. { } ⊢c m})"
proof (unfold par_complssst_def; intro conjI)
  define N1 where "N1 ≡ λB::('fun, 'atom) term_type × nat) stateful_strand.
    remdups (trms_listsst B@map (pair' Pair) (setops_listsst B))"

  define N2 where "N2 ≡ λB::('fun, 'atom) term_type × nat) stateful_strand.
    trmssst B ∪ pair ' setopssst B"

  define L where "L ≡ λA::('fun, 'atom) term_type × nat, 'lbl) labeled_stateful_strand.
    remdups (map (the_LabelN ∘ fst) (filter (Not ∘ is_LabelS) A))"

  define α where "α ≡ λp. var_rename (max_var (pair p::('fun, 'atom) term_type × nat) term))
    ::('fun, 'atom) term_type × nat) subst"

  let ?Sec = "(f (set C)) - {m. { } ⊢c m}"

  have 0:
    "length (L A) > 1"
    "list_all (wftrm' arity) (N1 (unlabel A))"
    "list_all (wftrm' arity) C"
    "has_all_wt_instances_of Γ (subtermsset (set C)) (set C)"
    "is_TComp_var_instance_closed Γ C"
    "∀i ∈ set (L A). ∀j ∈ set (L A). i ≠ j →
      comp_GSMP_disjoint public arity Ana Γ (N1 (proj_unl i A)) (N1 (proj_unl j A)) (M i) (M j) C"
    "∀(i,p) ∈ setopslssst A. ∀(j,q) ∈ setopslssst A. i ≠ j → mgu (pair p) (pair q · α p) = None"
    using a unfolding comp_par_complssst_def pair_code L_def N1_def α_def by meson+

  note 1 = trmssst_proj_subset(1) setopssst_proj_subset(1)

  have N1_iff_N2: "set (N1 A) = N2 A" for A
    unfolding pair_code trms_listsst_is_trmssst setops_listsst_is_setopssst N1_def N2_def by simp

  have N2_proj_subset: "N2 (proj_unl 1 A) ⊆ N2 (unlabel A)"
    for 1::'lbl and A:: "('fun, 'atom) term_type × nat, 'lbl) labeled_stateful_strand"
    using 1(1)[of 1 A] image_mono[OF 1(2)[of 1 A], of pair] unfolding N2_def by blast

  have L_in_iff: "1 ∈ set (L A) ↔ (∃a ∈ set A. is_LabelN 1 a)" for 1 A
    unfolding L_def by force

  have L_B_subset_A: "1 ∈ set (L A)" when 1: "1 ∈ set (L B)" for 1
    using L_in_iff[of 1 B] L_in_iff[of 1 A] B 1 by fastforce

  note B_setops = setopslssst_wt_instance_ex[OF B]

  have B_proj: "∀b ∈ set (proj 1 B). ∃a ∈ set (proj 1 A). ∃δ. b = a ·lssst δ ∧ ?D δ" for 1
    using proj_instance_ex[OF B] by fast

  have B': "∀t ∈ N2 (unlabel B). ∃s ∈ N2 (unlabel A). ∃δ. t = s · δ ∧ ?D δ"
    using trmssst_setopssst_wt_instance_ex[OF B] unfolding N2_def by blast

  have B'_proj: "∀t ∈ N2 (proj_unl 1 B). ∃s ∈ N2 (proj_unl 1 A). ∃δ. t = s · δ ∧ ?D δ" for 1
    using trmssst_setopssst_wt_instance_ex[OF B_proj] unfolding N2_def by presburger

```

```

have A_wf_trms: "wftrms (N2 (unlabel A))"
  using N1_iff_N2[of "unlabel A"] 0(2) unfolding wftrms_code list_all_iff by auto
hence A_proj_wf_trms: "wftrms (N2 (proj_unl 1 A))" for 1
  using 1[of 1] unfolding N2_def by blast
hence A_proj_wf_trms': "list_all (wftrms' arity) (N1 (proj_unl 1 A))" for 1
  using N1_iff_N2[of "proj_unl 1 A"] unfolding wftrms_code list_all_iff by presburger

note C_wf_trms = 0(3)[unfolded list_all_iff wftrms_code[symmetric]]

have 2: "GSMP (N2 (proj_unl 1 A))  $\subseteq$  GSMP (N2 (proj_unl 1' A))"
  when "1  $\notin$  set (L A)" for 1 1'
  and A::('fun, ('fun,'atom) term_type  $\times$  nat, 'lbl) labeled_stateful_strand"
  using that L_in_iff[of _ A] GSMP_mono[of "N2 (proj_unl 1 A)" "N2 (proj_unl 1' A)"]
    trmssst_unlabel_subset_if_no_label[of 1 A]
    setopssst_unlabel_subset_if_no_label[of 1 A]
  unfolding list_ex_iff N2_def by fast

have 3: "GSMP (N2 (proj_unl 1 B))  $\subseteq$  GSMP (N2 (proj_unl 1 A))" (is "?X  $\subseteq$  ?Y") for 1
proof
  fix t assume "t  $\in$  ?X"
  hence t: "t  $\in$  SMP (N2 (proj_unl 1 B))" "fv t = {}" unfolding GSMP_def by simp_all
  have "t  $\in$  SMP (N2 (proj_unl 1 A))"
    using t(1) B'_proj[of 1] SMP_wt_instances_subset[of "N2 (proj_unl 1 B)" "N2 (proj_unl 1 A)"]
    by metis
  thus "t  $\in$  ?Y" using t(2) unfolding GSMP_def by fast
qed

have "GSMP_disjoint (N2 (proj_unl 11 A)) (N2 (proj_unl 12 A)) ?Sec"
  when "11  $\in$  set (L A)" "12  $\in$  set (L A)" "11  $\neq$  12" for 11 12
proof -
  have "GSMP_disjoint (set (N1 (proj_unl 11 A))) (set (N1 (proj_unl 12 A))) ?Sec"
    using 0(6) that
    GSMP_disjoint_if_comp_GSMP_disjoint[
      OF A_proj_wf_trms'[of 11] A_proj_wf_trms'[of 12] 0(3),
      of "M 11" "M 12"]
    unfolding f_def by blast
  thus ?thesis using N1_iff_N2 by simp
qed
hence 4: "GSMP_disjoint (N2 (proj_unl 11 B)) (N2 (proj_unl 12 B)) ?Sec"
  when "11  $\in$  set (L A)" "12  $\in$  set (L A)" "11  $\neq$  12" for 11 12
  using that 3 unfolding GSMP_disjoint_def by blast

{ fix i p j q
  assume p: "(i,p)  $\in$  setopslss B" and q: "(j,q)  $\in$  setopslss B"
  and pq: " $\exists \delta$ . Unifier  $\delta$  (pair p) (pair q)"

  obtain p'  $\delta$ p where p': "(i,p')  $\in$  setopslss A" "p = p'  $\cdot_p$   $\delta$ p" "pair p = pair p'  $\cdot$   $\delta$ p"
    using p B_setops unfolding pair_def by auto

  obtain q'  $\delta$ q where q': "(j,q')  $\in$  setopslss A" "q = q'  $\cdot_p$   $\delta$ q" "pair q = pair q'  $\cdot$   $\delta$ q"
    using q B_setops unfolding pair_def by auto

  obtain  $\vartheta$  where "Unifier  $\vartheta$  (pair p) (pair q)" using pq by blast
  hence " $\exists \delta$ . Unifier  $\delta$  (pair p') (pair q'  $\cdot$   $\alpha$  p)"
    using p'(3) q'(3) var_rename_inv_comp[of "pair q'"] subst_subst_compose
    vars_term_disjoint_imp_unifier[
      OF var_rename_fv_disjoint[of "pair p'"],
      of " $\delta$ p  $\circ_s$   $\vartheta$ " "pair q'" "var_rename_inv (max_var_set (fv (pair p'))))  $\circ_s$   $\delta$ q  $\circ_s$   $\vartheta$ "]
    unfolding  $\alpha$ _def by fastforce
  hence "i = j"
    using mgu_None_is_subst_neq[of "pair p'" "pair q'  $\cdot$   $\alpha$  p"] p'(1) q'(1) 0(7)
    unfolding  $\alpha$ _def by fast

```

```

} thus "∀ (i,p) ∈ setopslsst B. ∀ (j,q) ∈ setopslsst B. (∃ δ. Unifier δ (pair p) (pair q)) → i = j"
  by blast

obtain a1 a2 where a: "a1 ∈ set (L A)" "a2 ∈ set (L A)" "a1 ≠ a2"
  using remdups_ex2[OF 0(1)[unfolded L_def]] unfolding L_def by moura

show "∀ l1 l2. l1 ≠ l2 → GSMP_disjoint (N2 (proj_unl l1 B)) (N2 (proj_unl l2 B)) ?Sec"
  using 2[of _ B] 4 4[OF a] L_B_subset_A unfolding GSMP_disjoint_def by blast

show "ground ?Sec" unfolding f_def by fastforce

show "∀ s ∈ ?Sec. ∀ s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ ?Sec"
proof (intro ballI)
  fix s s'
  assume s: "s ∈ ?Sec" and s': "s' ⊆ s"
  then obtain t δ where t: "t ∈ set C" "s = t · δ" "fv s = {}" "¬ {} ⊢c s"
    and δ: "wtsubst δ" "wftrms (subst_range δ)"
    unfolding f_def by blast

  obtain m ∅ where m: "m ∈ set C" "s' = m · ∅" and ∅: "wtsubst ∅" "wftrms (subst_range ∅)"
    using TComp_var_and_subterm_instance_closed_has_subterms_instances[
      OF 0(5,4) C_wf_trms in_subterms_Union[OF t(1)] s'[unfolded t(2)] δ]
    by blast
  thus "{} ⊢c s' ∨ s' ∈ ?Sec"
    using ground_subterm[OF t(3) s']
    unfolding f_def by blast
qed
qed
end
end

```



## 7 Examples

In this chapter, we present two examples illustrating our results: In section 7.1 we show that the TLS example from [2] is type-flaw resistant. In section 7.2 we show that the keyserver examples from [3, 4] are also type-flaw resistant and that the steps of the composed keyserver protocol from [4] satisfy our conditions for protocol composition.

### 7.1 Proving Type-Flaw Resistance of the TLS Handshake Protocol (Example\_TLS)

```
theory Example_TLS
imports "../Typed_Model"
begin
```

```
declare [[code_timing]]
```

#### 7.1.1 TLS example: Datatypes and functions setup

```
datatype ex_atom = PrivKey | SymKey | PubConst | Agent | Nonce | Bot
```

```
datatype ex_fun =
  clientHello | clientKeyExchange | clientFinished
| serverHello | serverCert | serverHelloDone
| finished | changeCipher | x509 | prfun | master | pmsForm
| sign | hash | crypt | pub | concat | privkey nat
| pubconst ex_atom nat
```

```
type_synonym ex_type = "(ex_fun, ex_atom) term_type"
type_synonym ex_var = "ex_type  $\times$  nat"
```

```
instance ex_atom::finite
proof
  let ?S = "UNIV::ex_atom set"
  have "?S = {PrivKey, SymKey, PubConst, Agent, Nonce, Bot}" by (auto intro: ex_atom.exhaust)
  thus "finite ?S" by (metis finite.emptyI finite.insertI)
qed
```

```
type_synonym ex_term = "(ex_fun, ex_var) term"
type_synonym ex_terms = "(ex_fun, ex_var) terms"
```

```
primrec arity::"ex_fun  $\Rightarrow$  nat" where
  "arity changeCipher = 0"
| "arity clientFinished = 4"
| "arity clientHello = 5"
| "arity clientKeyExchange = 1"
| "arity concat = 5"
| "arity crypt = 2"
| "arity finished = 1"
| "arity hash = 1"
| "arity master = 3"
| "arity pmsForm = 1"
| "arity prfun = 1"
| "arity (privkey _) = 0"
| "arity pub = 1"
| "arity (pubconst _ _) = 0"
| "arity serverCert = 1"
```

## 7 Examples

```

/ "arity serverHello = 5"
/ "arity serverHelloDone = 0"
/ "arity sign = 2"
/ "arity x509 = 2"

fun public::"ex_fun  $\Rightarrow$  bool" where
  "public (privkey _) = False"
/ "public _ = True"

fun Anacrypt::"ex_term list  $\Rightarrow$  (ex_term list  $\times$  ex_term list)" where
  "Anacrypt [Fun pub [k],m] = ([k], [m])"
/ "Anacrypt _ = ([], [])"

fun Anasign::"ex_term list  $\Rightarrow$  (ex_term list  $\times$  ex_term list)" where
  "Anasign [k,m] = ([], [m])"
/ "Anasign _ = ([], [])"

fun Ana::"ex_term  $\Rightarrow$  (ex_term list  $\times$  ex_term list)" where
  "Ana (Fun crypt T) = Anacrypt T"
/ "Ana (Fun finished T) = ([], T)"
/ "Ana (Fun master T) = ([], T)"
/ "Ana (Fun pmsForm T) = ([], T)"
/ "Ana (Fun serverCert T) = ([], T)"
/ "Ana (Fun serverHello T) = ([], T)"
/ "Ana (Fun sign T) = Anasign T"
/ "Ana (Fun x509 T) = ([], T)"
/ "Ana _ = ([], [])"

```

### 7.1.2 TLS example: Locale interpretation

```

lemma assm1:
  "Ana t = (K,M)  $\implies$  fvset (set K)  $\subseteq$  fv t"
  "Ana t = (K,M)  $\implies$  ( $\bigwedge$  g S'. Fun g S'  $\sqsubseteq$  t  $\implies$  length S' = arity g)
     $\implies$  k  $\in$  set K  $\implies$  Fun f T'  $\sqsubseteq$  k  $\implies$  length T' = arity f"
  "Ana t = (K,M)  $\implies$  K  $\neq$  []  $\vee$  M  $\neq$  []  $\implies$  Ana (t  $\cdot$   $\delta$ ) = (K  $\cdot_{list}$   $\delta$ , M  $\cdot_{list}$   $\delta$ )"
by (rule Ana.cases[of "t"], auto elim!: Anacrypt.elims Anasign.elims)+

lemma assm2: "Ana (Fun f T) = (K, M)  $\implies$  set M  $\subseteq$  set T"
by (rule Ana.cases[of "Fun f T"]) (auto elim!: Anacrypt.elims Anasign.elims)

lemma assm6: "0 < arity f  $\implies$  public f" by (cases f) simp_all

global interpretation im: intruder_model arity public Ana
  defines wftrm = "im.wftrm"
  and wftrms = "im.wftrms"
by unfold_locales (metis assm1(1), metis assm1(2), rule Ana.simps, metis assm2, metis assm1(3))

```

### 7.1.3 TLS Example: Typing function

```

definition  $\Gamma_v$ ::"ex_var  $\Rightarrow$  ex_type" where
  " $\Gamma_v$  v = (if ( $\forall$  t  $\in$  subterms (fst v). case t of
    (TComp f T)  $\Rightarrow$  arity f > 0  $\wedge$  arity f = length T
    | _  $\Rightarrow$  True)
    then fst v else TAtom Bot)"

fun  $\Gamma$ ::"ex_term  $\Rightarrow$  ex_type" where
  " $\Gamma$  (Var v) =  $\Gamma_v$  v"
/ " $\Gamma$  (Fun (privkey _) _) = TAtom PrivKey"
/ " $\Gamma$  (Fun changeCipher _) = TAtom PubConst"
/ " $\Gamma$  (Fun serverHelloDone _) = TAtom PubConst"
/ " $\Gamma$  (Fun (pubconst  $\tau$  _) _) = TAtom  $\tau$ "
/ " $\Gamma$  (Fun f T) = TComp f (map  $\Gamma$  T)"

```

## 7.1.4 TLS Example: Locale interpretation (typed model)

```

lemma assm7: "arity c = 0  $\implies$   $\exists$  a.  $\forall$  X.  $\Gamma$  (Fun c X) = TAtom a" by (cases c) simp_all

lemma assm8: "0 < arity f  $\implies$   $\Gamma$  (Fun f X) = TComp f (map  $\Gamma$  X)" by (cases f) simp_all

lemma assm9: "infinite {c.  $\Gamma$  (Fun c []) = TAtom a  $\wedge$  public c}"
proof -
  let ?T = "(range (pubconst a))::ex_fun set"
  have *:
    " $\bigwedge$  x y::nat. x  $\in$  UNIV  $\implies$  y  $\in$  UNIV  $\implies$  (pubconst a x = pubconst a y) = (x = y)"
    " $\bigwedge$  x::nat. x  $\in$  UNIV  $\implies$  pubconst a x  $\in$  ?T"
    " $\bigwedge$  y::ex_fun. y  $\in$  ?T  $\implies$   $\exists$  x  $\in$  UNIV. y = pubconst a x"
  by auto
  have "?T  $\subseteq$  {c.  $\Gamma$  (Fun c []) = TAtom a  $\wedge$  public c}" by auto
  moreover have " $\exists$  f::nat  $\Rightarrow$  ex_fun. bij_betw f UNIV ?T"
    using bij_betwI'[OF *] by blast
  hence "infinite ?T" by (metis nat_not_finite bij_betw_finite)
  ultimately show ?thesis using infinite_super by blast
qed

lemma assm10: "TComp f T  $\sqsubseteq$   $\Gamma$  t  $\implies$  arity f > 0"
proof (induction rule:  $\Gamma$ .induct)
  case (1 x)
  hence *: "TComp f T  $\sqsubseteq$   $\Gamma_v$  x" by simp
  hence " $\Gamma_v$  x  $\neq$  TAtom Bot" unfolding  $\Gamma_v$ _def by force
  hence " $\forall$  t  $\in$  subterms (fst x). case t of
    (TComp f T)  $\Rightarrow$  arity f > 0  $\wedge$  arity f = length T
    | _  $\Rightarrow$  True"
    unfolding  $\Gamma_v$ _def by argo
  thus ?case using * unfolding  $\Gamma_v$ _def by fastforce
qed auto

lemma assm11: "im.wftrm ( $\Gamma$  (Var x))"
proof -
  have "im.wftrm ( $\Gamma_v$  x)" unfolding  $\Gamma_v$ _def im.wftrm_def by auto
  thus ?thesis by simp
qed

lemma assm12: " $\Gamma$  (Var ( $\tau$ , n)) =  $\Gamma$  (Var ( $\tau$ , m))"
  apply (cases " $\forall$  t  $\in$  subterms  $\tau$ . case t of
    (TComp f T)  $\Rightarrow$  arity f > 0  $\wedge$  arity f = length T
    | _  $\Rightarrow$  True")
  by (auto simp add:  $\Gamma_v$ _def)

lemma Ana_const: "arity c = 0  $\implies$  Ana (Fun c T) = ([], [])"
by (cases c) simp_all

lemma Ana_keys_subterm: "Ana t = (K, T)  $\implies$  k  $\in$  set K  $\implies$  k  $\sqsubseteq$  t"
proof (induct t rule: Ana.induct)
  case (1 U)
  then obtain m where "U = [Fun pub [k], m]" "K = [k]" "T = [m]"
    by (auto elim!: Anacrypt.elims Anasign.elims)
  thus ?case using Fun_subterm_inside_params[of k crypt U] by auto
qed (auto elim!: Anacrypt.elims Anasign.elims)

global_interpretation tm: typed_model' arity public Ana  $\Gamma$ 
by (unfold_locales, unfold wftrm_def[symmetric],
  metis assm7, metis assm8, metis assm9, metis assm10, metis assm11, metis assm6,
  metis assm12, metis Ana_const, metis Ana_keys_subterm)

```

### 7.1.5 TLS example: Proving type-flaw resistance

abbreviation  $\Gamma_{v\_clientHello}$  where

```
" $\Gamma_{v\_clientHello}$   $\equiv$ 
  TComp clientHello [TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce]"
```

abbreviation  $\Gamma_{v\_serverHello}$  where

```
" $\Gamma_{v\_serverHello}$   $\equiv$ 
  TComp serverHello [TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce]"
```

abbreviation  $\Gamma_{v\_pub}$  where

```
" $\Gamma_{v\_pub}$   $\equiv$  TComp pub [TAtom PrivKey]"
```

abbreviation  $\Gamma_{v\_x509}$  where

```
" $\Gamma_{v\_x509}$   $\equiv$  TComp x509 [TAtom Agent,  $\Gamma_{v\_pub}$ ]"
```

abbreviation  $\Gamma_{v\_sign}$  where

```
" $\Gamma_{v\_sign}$   $\equiv$  TComp sign [TAtom PrivKey,  $\Gamma_{v\_x509}$ ]"
```

abbreviation  $\Gamma_{v\_serverCert}$  where

```
" $\Gamma_{v\_serverCert}$   $\equiv$  TComp serverCert [ $\Gamma_{v\_sign}$ ]"
```

abbreviation  $\Gamma_{v\_pmsForm}$  where

```
" $\Gamma_{v\_pmsForm}$   $\equiv$  TComp pmsForm [TAtom SymKey]"
```

abbreviation  $\Gamma_{v\_crypt}$  where

```
" $\Gamma_{v\_crypt}$   $\equiv$  TComp crypt [ $\Gamma_{v\_pub}$ ,  $\Gamma_{v\_pmsForm}$ ]"
```

abbreviation  $\Gamma_{v\_clientKeyExchange}$  where

```
" $\Gamma_{v\_clientKeyExchange}$   $\equiv$ 
  TComp clientKeyExchange [ $\Gamma_{v\_crypt}$ ]"
```

abbreviation  $\Gamma_{v\_HSMsigs}$  where

```
" $\Gamma_{v\_HSMsigs}$   $\equiv$  TComp concat [
   $\Gamma_{v\_clientHello}$ ,
   $\Gamma_{v\_serverHello}$ ,
   $\Gamma_{v\_serverCert}$ ,
  TAtom PubConst,
   $\Gamma_{v\_clientKeyExchange}$ ]"
```

abbreviation " $T_1$   $n \equiv \text{Var } (TAtom \text{ Nonce}, n)$ "

abbreviation " $T_2$   $n \equiv \text{Var } (TAtom \text{ Nonce}, n)$ "

abbreviation " $R_A$   $n \equiv \text{Var } (TAtom \text{ Nonce}, n)$ "

abbreviation " $R_B$   $n \equiv \text{Var } (TAtom \text{ Nonce}, n)$ "

abbreviation " $S$   $n \equiv \text{Var } (TAtom \text{ Nonce}, n)$ "

abbreviation " $Cipher$   $n \equiv \text{Var } (TAtom \text{ Nonce}, n)$ "

abbreviation " $Comp$   $n \equiv \text{Var } (TAtom \text{ Nonce}, n)$ "

abbreviation " $B$   $n \equiv \text{Var } (TAtom \text{ Agent}, n)$ "

abbreviation " $Pr_{ca}$   $n \equiv \text{Var } (TAtom \text{ PrivKey}, n)$ "

abbreviation " $PMS$   $n \equiv \text{Var } (TAtom \text{ SymKey}, n)$ "

abbreviation " $P_B$   $n \equiv \text{Var } (TComp \text{ pub } [TAtom \text{ PrivKey}], n)$ "

abbreviation " $HSMsigs$   $n \equiv \text{Var } (\Gamma_{v\_HSMsigs}, n)$ "

#### Defining the over-approximation set

abbreviation  $clientHello_{trm}$  where

```
" $clientHello_{trm}$   $\equiv$  Fun clientHello [ $T_1$  0,  $R_A$  1,  $S$  2,  $Cipher$  3,  $Comp$  4]"
```

abbreviation  $serverHello_{trm}$  where

```
" $serverHello_{trm}$   $\equiv$  Fun serverHello [ $T_2$  0,  $R_B$  1,  $S$  2,  $Cipher$  3,  $Comp$  4]"
```

abbreviation  $serverCert_{trm}$  where



```

"serverCerttrm ≡ Fun serverCert [Fun sign [Prca 0, Fun x509 [B 1, PB 2]]]"

abbreviation serverHelloDonetrm where
  "serverHelloDonetrm ≡ Fun serverHelloDone []"

abbreviation clientKeyExchangetrm where
  "clientKeyExchangetrm ≡ Fun clientKeyExchange [Fun crypt [PB 0, Fun pmsForm [PMS 1]]]"

abbreviation changeCiphertrm where
  "changeCiphertrm ≡ Fun changeCipher []"

abbreviation finishedtrm where
  "finishedtrm ≡ Fun finished [Fun prfun [
    Fun clientFinished [
      Fun prfun [Fun master [PMS 0, RA 1, RB 2]],
      RA 3, RB 4, Fun hash [HSMsgs 5]
    ]
  ]]"

definition MTLS::"ex_term list" where
  "MTLS ≡ [
    clientHellotrm,
    serverHellotrm,
    serverCerttrm,
    serverHelloDonetrm,
    clientKeyExchangetrm,
    changeCiphertrm,
    finishedtrm
  ]"

```

### 7.1.6 Theorem: The TLS handshake protocol is type-flaw resistant

```

theorem "tm.tfrset (set MTLS)"
by (rule tm.tfrset_if_comp_tfrset) eval

end

```

## 7.2 The Keyserver Example (Example\_Keyserver)

```

theory Example_Keyserver
imports "../Stateful_Compositionality"
begin

declare [[code_timing]]

```

### 7.2.1 Setup

#### Datatypes and functions setup

```

datatype ex_lbl = Label1 ("1") | Label2 ("2")

datatype ex_atom =
  Agent | Value | Attack | PrivFunSec
| Bot

datatype ex_fun =
  ring | valid | revoked | events | beginauth nat | endauth nat | pubkeys | seen
| invkey | tuple | tuple' | attack nat
| sign | crypt | update | pw
| encodingsecret | pubkey nat
| pubconst ex_atom nat

type_synonym ex_type = "(ex_fun, ex_atom) term_type"

```

## 7 Examples

```

type_synonym ex_var = "ex_type × nat"

lemma ex_atom_UNIV:
  "(UNIV::ex_atom set) = {Agent, Value, Attack, PrivFunSec, Bot}"
by (auto intro: ex_atom.exhaust)

instance ex_atom::finite
by intro_classes (metis ex_atom_UNIV finite.emptyI finite.insertI)

lemma ex_lbl_UNIV:
  "(UNIV::ex_lbl set) = {Label1, Label2}"
by (auto intro: ex_lbl.exhaust)

type_synonym ex_term = "(ex_fun, ex_var) term"
type_synonym ex_terms = "(ex_fun, ex_var) terms"

primrec arity::"ex_fun ⇒ nat" where
  "arity ring = 2"
| "arity valid = 3"
| "arity revoked = 3"
| "arity events = 1"
| "arity (beginauth _) = 3"
| "arity (endauth _) = 3"
| "arity pubkeys = 2"
| "arity seen = 2"
| "arity invkey = 2"
| "arity tuple = 2"
| "arity tuple' = 2"
| "arity (attack _) = 0"
| "arity sign = 2"
| "arity crypt = 2"
| "arity update = 4"
| "arity pw = 2"
| "arity (pubkey _) = 0"
| "arity encodingsecret = 0"
| "arity (pubconst _ _) = 0"

fun public::"ex_fun ⇒ bool" where
  "public (pubkey _) = False"
| "public encodingsecret = False"
| "public _ = True"

fun Ana_crypt::"ex_term list ⇒ (ex_term list × ex_term list)" where
  "Ana_crypt [k,m] = ([Fun invkey [Fun encodingsecret [], k]], [m])"
| "Ana_crypt _ = ([], [])"

fun Ana_sign::"ex_term list ⇒ (ex_term list × ex_term list)" where
  "Ana_sign [k,m] = ([], [m])"
| "Ana_sign _ = ([], [])"

fun Ana::"ex_term ⇒ (ex_term list × ex_term list)" where
  "Ana (Fun tuple T) = ([], T)"
| "Ana (Fun tuple' T) = ([], T)"
| "Ana (Fun sign T) = Ana_sign T"
| "Ana (Fun crypt T) = Ana_crypt T"
| "Ana _ = ([], [])"

```

### Keyserver example: Locale interpretation

```

lemma assm1:
  "Ana t = (K,M) ⇒ fv_set (set K) ⊆ fv t"
  "Ana t = (K,M) ⇒ (⋀g S'. Fun g S' ⊆ t ⇒ length S' = arity g)
    ⇒ k ∈ set K ⇒ Fun f T' ⊆ k ⇒ length T' = arity f"

```

```

"Ana t = (K,M)  $\implies$  K  $\neq$  []  $\vee$  M  $\neq$  []  $\implies$  Ana (t .  $\delta$ ) = (K .list  $\delta$ , M .list  $\delta$ )"
by (rule Ana.cases[of "t"], auto elim!: Anacrypt.elims Anasign.elims)+

lemma assm2: "Ana (Fun f T) = (K, M)  $\implies$  set M  $\subseteq$  set T"
by (rule Ana.cases[of "Fun f T"]) (auto elim!: Anacrypt.elims Anasign.elims)

lemma assm6: "0 < arity f  $\implies$  public f" by (cases f) simp_all

global interpretation im: intruder_model arity public Ana
  defines wftrm = "im.wftrm"
by unfold_locales (metis assm1(1), metis assm1(2), rule Ana.simps, metis assm2, metis assm1(3))

type_synonym ex_strand_step = "(ex_fun, ex_var) strand_step"
type_synonym ex_strand = "(ex_fun, ex_var) strand"

```

### Typing function

```

definition  $\Gamma_v :: "ex\_var \Rightarrow ex\_type"$  where
  " $\Gamma_v v =$  (if ( $\forall t \in \text{subterms } (\text{fst } v).$  case t of
    (TComp f T)  $\Rightarrow$  arity f > 0  $\wedge$  arity f = length T
    | _  $\Rightarrow$  True)
    then fst v else TAtom Bot)"

fun  $\Gamma :: "ex\_term \Rightarrow ex\_type"$  where
  " $\Gamma$  (Var v) =  $\Gamma_v v$ "
| " $\Gamma$  (Fun (attack _) _) = TAtom Attack"
| " $\Gamma$  (Fun (pubkey _) _) = TAtom Value"
| " $\Gamma$  (Fun encodingsecret _) = TAtom PrivFunSec"
| " $\Gamma$  (Fun (pubconst  $\tau$  _) _) = TAtom  $\tau$ "
| " $\Gamma$  (Fun f T) = TComp f (map  $\Gamma$  T)"

```

### Locale interpretation: typed model

```

lemma assm7: "arity c = 0  $\implies$   $\exists a. \forall X. \Gamma$  (Fun c X) = TAtom a" by (cases c) simp_all

lemma assm8: "0 < arity f  $\implies$   $\Gamma$  (Fun f X) = TComp f (map  $\Gamma$  X)" by (cases f) simp_all

lemma assm9: "infinite {c.  $\Gamma$  (Fun c []) = TAtom a  $\wedge$  public c}"
proof -
  let ?T = "(range (pubconst a))::ex_fun set"
  have *:
    " $\bigwedge x y :: \text{nat}. x \in \text{UNIV} \implies y \in \text{UNIV} \implies (\text{pubconst } a \ x = \text{pubconst } a \ y) = (x = y)$ "
    " $\bigwedge x :: \text{nat}. x \in \text{UNIV} \implies \text{pubconst } a \ x \in ?T$ "
    " $\bigwedge y :: \text{ex\_fun}. y \in ?T \implies \exists x \in \text{UNIV}. y = \text{pubconst } a \ x$ "
  by auto
  have "?T  $\subseteq$  {c.  $\Gamma$  (Fun c []) = TAtom a  $\wedge$  public c}" by auto
  moreover have " $\exists f :: \text{nat} \Rightarrow \text{ex\_fun}. \text{bij\_betw } f \ \text{UNIV } ?T$ "
    using bij_betwI'[OF *] by blast
  hence "infinite ?T" by (metis nat_not_finite bij_betw_finite)
  ultimately show ?thesis using infinite_super by blast
qed

```

```

lemma assm10: "TComp f T  $\sqsubseteq$   $\Gamma$  t  $\implies$  arity f > 0"
proof (induction rule:  $\Gamma$ .induct)
  case (1 x)
  hence *: "TComp f T  $\sqsubseteq$   $\Gamma_v x$ " by simp
  hence " $\Gamma_v x \neq \text{TAtom Bot}$ " unfolding  $\Gamma_v$ _def by force
  hence " $\forall t \in \text{subterms } (\text{fst } x).$  case t of
    (TComp f T)  $\Rightarrow$  arity f > 0  $\wedge$  arity f = length T
    | _  $\Rightarrow$  True"
  unfolding  $\Gamma_v$ _def by argo
  thus ?case using * unfolding  $\Gamma_v$ _def by fastforce
qed auto

```

```

lemma assm11: "im.wftrm (Γ (Var x))"
proof -
  have "im.wftrm (Γv x)" unfolding Γv_def im.wftrm_def by auto
  thus ?thesis by simp
qed

lemma assm12: "Γ (Var (τ, n)) = Γ (Var (τ, m))"
apply (cases "∀ t ∈ subterms τ. case t of
  (TComp f T) ⇒ arity f > 0 ∧ arity f = length T
  | _ ⇒ True")
by (auto simp add: Γv_def)

lemma Ana_const: "arity c = 0 ⇒ Ana (Fun c T) = ([], [])"
by (cases c) simp_all

lemma Ana_subst': "Ana (Fun f T) = (K,M) ⇒ Ana (Fun f T · δ) = (K ·list δ, M ·list δ)"
by (cases f) (auto elim!: Anacrypt.elims Anasign.elims)

global_interpretation tm: typed_model' arity public Ana Γ
by (unfold locales, unfold wftrm_def[symmetric])
  (metis assm7, metis assm8, metis assm9, metis assm10, metis assm11, metis assm6,
   metis assm12, metis Ana_const, metis Ana_subst')

```

### Locale interpretation: labeled stateful typed model

```

global_interpretation stm: labeled_stateful_typed_model' arity public Ana Γ tuple 1 2
by standard (rule arity.simps, metis Ana_subst', metis assm12, metis Ana_const, simp)

type_synonym ex_stateful_strand_step = "(ex_fun,ex_var) stateful_strand_step"
type_synonym ex_stateful_strand = "(ex_fun,ex_var) stateful_strand"

type_synonym ex_labeled_stateful_strand_step =
  "(ex_fun,ex_var,ex_lbl) labeled_stateful_strand_step"

type_synonym ex_labeled_stateful_strand =
  "(ex_fun,ex_var,ex_lbl) labeled_stateful_strand"

```

## 7.2.2 Theorem: Type-flaw resistance of the keyserver example from the CSF18 paper

```

abbreviation "PK n ≡ Var (TAtom Value,n)"
abbreviation "A n ≡ Var (TAtom Agent,n)"
abbreviation "X n ≡ (TAtom Agent,n)"

abbreviation "ringset t ≡ Fun ring [Fun encodingsecret [], t]"
abbreviation "validset t t' ≡ Fun valid [Fun encodingsecret [], t, t']"
abbreviation "revokedset t t' ≡ Fun revoked [Fun encodingsecret [], t, t']"
abbreviation "eventsset ≡ Fun events [Fun encodingsecret []]"

abbreviation Sks :: "(ex_fun,ex_var) stateful_strand_step list" where
  "Sks ≡ [
    insert⟨Fun (attack 0) [], eventsset⟩,
    delete⟨PK 0, validset (A 0) (A 0)⟩,
    ∀ (TAtom Agent,0)⟨PK 0 not in revokedset (A 0) (A 0)⟩,
    ∀ (TAtom Agent,0)⟨PK 0 not in validset (A 0) (A 0)⟩,
    insert⟨PK 0, validset (A 0) (A 0)⟩,
    insert⟨PK 0, ringset (A 0)⟩,
    insert⟨PK 0, revokedset (A 0) (A 0)⟩,
    select⟨PK 0, validset (A 0) (A 0)⟩,
    select⟨PK 0, ringset (A 0)⟩,
    receive⟨Fun invkey [Fun encodingsecret [], PK 0]⟩,
    receive⟨Fun sign [Fun invkey [Fun encodingsecret [], PK 0], Fun tuple' [A 0, PK 0]]⟩,
  ]"

```

```

send⟨Fun invkey [Fun encodingsecret [], PK 0]⟩,
send⟨Fun sign [Fun invkey [Fun encodingsecret [], PK 0], Fun tuple' [A 0, PK 0]]⟩
]"

theorem "stm.tfrsst Sks"
proof -
  let ?M = "concat (map subterms_list (trms_listsst Sks@map (pair' tuple) (setops_listsst Sks)))"
  have "comp_tfrsst arity Ana  $\Gamma$  tuple ?M Sks" by eval
  thus ?thesis by (rule stm.tfrsst_if_comp_tfrsst)
qed

```

### 7.2.3 Theorem: Type-flaw resistance of the keyserver examples from the ESORICS18 paper

```

abbreviation "signmsg t t'  $\equiv$  Fun sign [t, t']"
abbreviation "cryptmsg t t'  $\equiv$  Fun crypt [t, t']"
abbreviation "invkeymsg t  $\equiv$  Fun invkey [Fun encodingsecret [], t]"
abbreviation "updatemsg a b c d  $\equiv$  Fun update [a,b,c,d]"
abbreviation "pwmsg t t'  $\equiv$  Fun pw [t, t']"

abbreviation "beginauthset n t t'  $\equiv$  Fun (beginauth n) [Fun encodingsecret [], t, t']"
abbreviation "endauthset n t t'  $\equiv$  Fun (endauth n) [Fun encodingsecret [], t, t']"
abbreviation "pubkeyssset t  $\equiv$  Fun pubkeys [Fun encodingsecret [], t]"
abbreviation "seensset t  $\equiv$  Fun seen [Fun encodingsecret [], t]"

declare [[coercion "Var::ex_var  $\Rightarrow$  ex_term"]]
declare [[coercion_enabled]]

```

definition S'<sub>ks</sub>::"ex\_labeled\_stateful\_strand\_step list" where

```

"S'ks  $\equiv$  [
  concat (map subterms_list (trms_listsst Sks@map (pair' tuple) (setops_listsst Sks)))

  #1#W1#1#
  ⟨1, send⟨invkeymsg (PK 0)⟩⟩,
  ⟨*, ⟨PK 0 in validset (A 0) (A 1)⟩⟩,
  ⟨1, receive⟨Fun (attack 0) []⟩⟩,

  #1#W2#1#
  ⟨1, send⟨signmsg (invkeymsg (PK 0)) (Fun tuple' [A 0, PK 0])⟩⟩,
  ⟨*, ⟨PK 0 in validset (A 0) (A 1)⟩⟩,
  ⟨*,  $\forall X 0, X 1 \langle \text{PK } 0 \text{ not in validset (Var (X 0)) (Var (X 1))} \rangle \rangle$ ,
  ⟨1,  $\forall X 0, X 1 \langle \text{PK } 0 \text{ not in revokedset (Var (X 0)) (Var (X 1))} \rangle \rangle$ ,
  ⟨*, ⟨PK 0 not in beginauthset 0 (A 0) (A 1)⟩⟩,

  #1#W3#1#
  ⟨*, ⟨PK 0 in beginauthset 0 (A 0) (A 1)⟩⟩,
  ⟨*, ⟨PK 0 in endauthset 0 (A 0) (A 1)⟩⟩,

  #1#W4#1#
  ⟨*, receive⟨PK 0⟩⟩,
  ⟨*, receive⟨invkeymsg (PK 0)⟩⟩,

  #1#W5#1#
  ⟨1, insert⟨PK 0, ringset (A 0)⟩⟩,
  ⟨*, insert⟨PK 0, validset (A 0) (A 1)⟩⟩,
  ⟨*, insert⟨PK 0, beginauthset 0 (A 0) (A 1)⟩⟩,
  ⟨*, insert⟨PK 0, endauthset 0 (A 0) (A 1)⟩⟩,

  #1#W6#1#
  ⟨1, select⟨PK 0, ringset (A 0)⟩⟩,
  ⟨1, delete⟨PK 0, ringset (A 0)⟩⟩,

```

```

H016/W7/L1
⟨*, ⟨PK 0 not in endauthset 0 (A 0) (A 1)⟩⟩,
⟨*, delete⟨PK 0, validset (A 0) (A 1)⟩⟩,
⟨1, insert⟨PK 0, revokedset (A 0) (A 1)⟩⟩,

H016/W7/B/L1
nothing/h016

H016/W7/B/L1
⟨1, send⟨PK 0⟩⟩,

H016/W7/D/L1
⟨1, send⟨Fun (attack 0) []⟩⟩,

H016/W7/D/L1
H016/W7/L1
⟨2, send⟨invkeymsg (PK 0)⟩⟩,
⟨*, ⟨PK 0 in validset (A 0) (A 1)⟩⟩,
⟨2, receive⟨Fun (attack 1) []⟩⟩,

H016/W7/L2
⟨2, send⟨cryptmsg (PK 0) (updatemsg (A 0) (A 1) (PK 1) (pwmsg (A 0) (A 1)))⟩⟩,
⟨2, select⟨PK 0, pubkeysset (A 0)⟩⟩,
⟨2, ∀X 0⟨PK 0 not in pubkeysset (Var (X 0))⟩⟩,
⟨2, ∀X 0⟨PK 0 not in seenset (Var (X 0))⟩⟩,

H016/W7/B/L2
⟨*, ⟨PK 0 in beginauthset 1 (A 0) (A 1)⟩⟩,
⟨*, ⟨PK 0 in endauthset 1 (A 0) (A 1)⟩⟩,

H016/W7/D/L2
⟨*, receive⟨PK 0⟩⟩,
⟨*, receive⟨invkeymsg (PK 0)⟩⟩,

H016/W7/B/L2
⟨2, select⟨PK 0, pubkeysset (A 0)⟩⟩,
⟨*, insert⟨PK 0, beginauthset 1 (A 0) (A 1)⟩⟩,
⟨2, receive⟨cryptmsg (PK 0) (updatemsg (A 0) (A 1) (PK 1) (pwmsg (A 0) (A 1)))⟩⟩,

H016/W7/B/L2
⟨*, ⟨PK 0 not in endauthset 1 (A 0) (A 1)⟩⟩,
⟨*, insert⟨PK 0, validset (A 0) (A 1)⟩⟩,
⟨*, insert⟨PK 0, endauthset 1 (A 0) (A 1)⟩⟩,
⟨2, insert⟨PK 0, seenset (A 0)⟩⟩,

H016/W7/L2
⟨2, receive⟨pwmsg (A 0) (A 1)⟩⟩,

H016/W7/B/L2
nothing/h016

H016/W7/B/L2
⟨2, insert⟨PK 0, pubkeysset (A 0)⟩⟩,

H016/W7/D/L2
⟨2, send⟨Fun (attack 1) []⟩⟩
]"

theorem "stm.tfrsst (unlabel S'_{ks})"
proof -
  let ?S = "unlabel S'_{ks}"
  let ?M = "concat (map subterms_list (trms_listsst ?S@map (pair' tuple) (setops_listsst ?S)))"
  have "comp_tfrsst arity Ana Γ tuple ?M ?S" by eval

```

```

  thus ?thesis by (rule stm.tfrsst_if_comp_tfrsst)
qed

```

### 7.2.4 Theorem: The steps of the keyserver protocols from the ESORICS18 paper satisfy the conditions for parallel composition

theorem

fixes  $S$  f

```

defines "S ≡ [PK 0, invkeymsg (PK 0), Fun encodingsecret []]@concat (
  map (λs. [s, Fun tuple [PK 0, s]])
    [validset (A 0) (A 1), beginauthset 0 (A 0) (A 1), endauthset 0 (A 0) (A 1),
     beginauthset 1 (A 0) (A 1), endauthset 1 (A 0) (A 1)]]@
  [A 0]"

```

and " $f ≡ λM. \{t \cdot \delta \mid t \delta. t \in M \wedge tm.wt_{subst} \delta \wedge im.wf_{trms} (subst\_range \delta) \wedge fv (t \cdot \delta) = \{\}\}$ "

and " $Sec ≡ (f (set S)) - \{m. im.intruder\_synth \{\} m\}$ "

shows "stm.par\_comp<sub>l<sub>sst</sub></sub> S' <sub>ks</sub> Sec"

proof -

```

let ?N = "λP. concat (map subterms_list (trms_listsst P@map (pair' tuple) (setops_listsst P)))"

```

```

let ?M = "λl. ?N (proj_unl l S' ks)"

```

```

have "comp_par_complsst public arity Ana Γ tuple S' ks ?M S"

```

```

  unfolding S_def by eval

```

```

thus ?thesis

```

```

  using stm.par_complsst_if_comp_par_complsst[of S' ks ?M S]

```

```

  unfolding Sec_def f_def wftrms_def[symmetric] by blast

```

qed

end





# Bibliography

- [1] A. V. Hess. *Typing and Compositionality for Stateful Security Protocols*. PhD thesis, 2019. URL <https://orbit.dtu.dk/en/publications/typing-and-compositionality-for-stateful-security-protocols>.
- [2] A. V. Hess and S. Mödersheim. Formalizing and Proving a Typing Result for Security Protocols in Isabelle/HOL. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 451–463. IEEE Computer Society, 2017. doi: 10.1109/CSF.2017.27.
- [3] A. V. Hess and S. Mödersheim. A Typing Result for Stateful Protocols. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 374–388. IEEE Computer Society, 2018. doi: 10.1109/CSF.2018.00034.
- [4] A. V. Hess, S. Mödersheim, and A. D. Brucker. Stateful Protocol Composition. In J. López, J. Zhou, and M. Soriano, editors, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *Lecture Notes in Computer Science*, pages 427–446. Springer, 2018. doi: 10.1007/978-3-319-99073-6\_21.