Constraint Differentiation: Search-Space Reduction for the Constraint-Based Analysis of Security Protocols

Sebastian Mödersheim[1] Luca Viganò[2] David Basin[3]

[1] IBM Zurich Research Laboratory, Switzerland, smo@zurich.ibm.com

[2] Department of Computer Science, Verona, Italy, luca.vigano@univr.it

[3] Department of Computer Science, ETH Zurich, Switzerland, basin@inf.ethz.ch

December 6, 2008

Abstract

We introduce constraint differentiation, a powerful technique for reducing search when model-checking security protocols using constraint-based methods. Constraint differentiation works by eliminating certain kinds of redundancies that arise in the search space when using constraints to represent and manipulate the messages that may be sent by an active intruder. We define constraint differentiation in a general way, independent of the technical and conceptual details of the underlying constraint-based method and protocol model. Formally, we prove that constraint differentiation terminates and is correct, under the assumption that the original constraint-based approach has these properties. Practically, as a concrete case study, we have integrated this technique into OFMC, a state-of-the-art model-checker, and demonstrated its effectiveness by extensive experimentation. Our results show that constraint differentiation substantially reduces search and considerably improves the performance of OFMC, enabling its application to a wider class of problems.

Keywords. Security protocols. Security protocol verification. Constraints. Model checking. Partial-order reduction.

1 Introduction

Context.

A wide variety of model-checking approaches [9, 11, 16, 18, 19, 21, 29, 30, 32, 35] have been developed to analyze security protocols under the assumption of perfect (black-box) cryptography. The major challenge faced when building such model-checking tools is how to handle the enormous search spaces that arise during protocol analysis. There are two separate problems here that must be addressed.

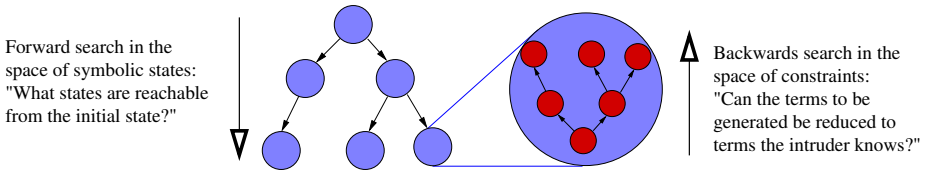

Figure 1: Two layers of search.

The Prolific Intruder Problem: The standard Dolev-Yao intruder model [20] defines an infinite set of messages that the intruder can generate from his "knowledge", i.e., the messages he has previously seen. This gives rise to infinite branch-points in the search space where the intruder may send any of these messages. In some approaches [27, 4], this set may be constrained to be finite by introducing bounds or type restrictions, but even then the branching induced by such a restricted Dolev-Yao intruder is typically still enormous.

The Interleaving Problem: A large number of possible interleavings result from parallel executions of a protocol by the honest agents and the intruder. This gives rise to multiple occurrences of identical (or equivalent) states in the state space, thereby increasing search.

A number of symbolic, constraint-based approaches have been proposed to tackle the problem of the prolific intruder [1, 9, 12, 16, 18, 22, 24, 30]. Although they vary in their details, they have in common a symbolic representation of the state space, where sets of (ground) states are represented by terms with variables and constraints on the variables. These constraints describe what terms an intruder can generate from a given set of known messages according to the Dolev-Yao model. Moreover, all these approaches use similar reduction rules and strategies to reduce the constraints into a solved form providing a finite representation of the infinitely many solutions for the initial constraint set. In particular, reduction is demand-driven, or lazy, in the sense that the variables in the constraints are instantiated during reduction only as needed. Therefore, we will refer to the technique underlying these constraint-based approaches as the *lazy intruder*.

Figure 1 shows the structure of the search spaces that arise when using the lazy intruder technique. The search space is tree-structured and arises in a standard way (see Section 2.5) from a transition system describing the possible interactions between agents executing the protocol and the intruder. The states of the transition system are symbolic, in that they contain variables, whose instantiations are restricted by the constraints. As illustrated, we can in general distinguish between two *layers of search*. The first layer is search in the space of constraints to find all (symbolic) solutions for a given set of constraints under the Dolev-Yao model and the second layer builds upon the first, exploring the symbolic search space to determine if an *attack state* is reachable. As the figure

illustrates, the first kind of search is performed in a backward fashion ("Can the terms to be generated be reduced to terms the intruder knows?"), while the second kind is performed in a forward fashion ("What states are reachable from the initial state?").

Example 1. Consider a node N of the search tree induced by a protocol, as illustrated in Figure 2. N represents a state where several agents are waiting to receive messages. For instance, assume that the agent a expects a message of the form $\{|a, x|\}_k$. This message represents the symmetric encryption of the pair a, x with the key k, where a is the name of the agent and x is a variable that may be instantiated with any ground term (i.e. one without variables). The agent a will therefore accept any message that is encrypted with k and where the encrypted text starts with the name a. The question is now which acceptable messages can the intruder construct from his current knowledge. First, the intruder can send any message that he has previously learned and that is of the required form (even if he does not know k). Second, if he knows k and a, he may choose any arbitrary message m that he can construct and send $\{|a, m|\}_k$. Note that, in general, there are infinitely many possible such messages that the intruder can send unless we introduce some restrictions such as a bound on the depth of the term m.

In a naïve approach, the node N of the search tree has one successor for each acceptable message that the intruder can construct and send to a (plus similar successors for other waiting agents). In contrast, in the lazy intruder approach, N has just one symbolic successor for each waiting agent. In this setting, the agent a receives the symbolic term $\{|a, x|\}_k$ (where x is not instantiated) and the successor symbolic state contains the constraint that the intruder can construct $\{|a, x|\}_k$ from the messages he knew at node N. In the second layer of the search, we now check that the constraints of each reached symbolic state are indeed satisfiable, that is, the variables can be instantiated in at least one way, satisfying the constraints. For instance, if the intruder knows neither k nor any message of the required form, then the example constraint is unsatisfiable. Therefore the respective successor node of N, and the entire subtree below it, is pruned.

Observe that the intruder knowledge in the lazy intruder approach may also contain variables, as the reply of an agent may contain uninstantiated variables from previously received messages. For example, if the intruder knowledge at the node N contains the message $\{|z|\}_k$, for a variable z that appeared in a previous message, then the solutions for the constraint on the generation of $\{|a, x|\}_k$ include the unifiers of z and a, x. The constraint reduction procedure on the second layer then must check whether one of these solutions also satisfies the remaining constraints, like those on the variable z. □

The lazy intruder drastically reduces the size of the resulting search tree generated during protocol analysis, providing an effective solution to the first problem: the prolific Dolev-Yao intruder. However, the second problem is not addressed by the lazy intruder, namely the large number of interleavings possible due to parallel protocol executions. In standard model-checking approaches for

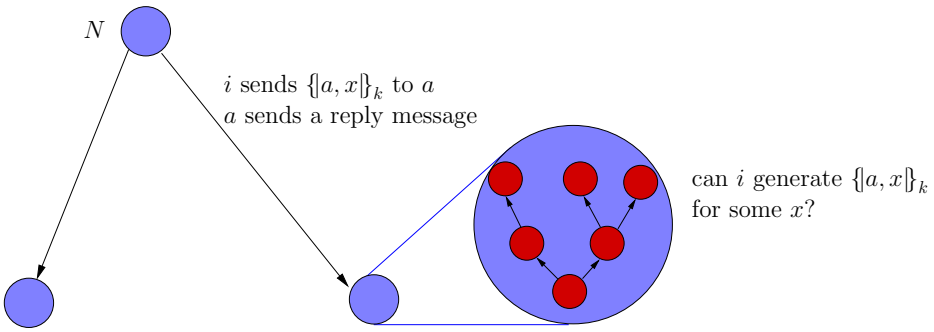

Figure 2: An example of the two layers of search.

concurrent systems, the interleaving problem is often handled using *partial-order reduction* (*POR*), a technique that reduces the number of interleavings that need to be considered by exploiting independencies between the possible transitions [34].[1] One might expect that the direct combination of the lazy intruder with partial-order reduction as part of the second layer of the search (i.e., when searching the symbolic state space) would allow us to simultaneously address both problems. However, as we will explain below, this combination is not effective: the different transitions of the lazy intruder rarely lead to the same (symbolic) successor state and therefore there is practically no independence of transitions that can be exploited by POR.

Contribution.

In this paper, we present a general reduction technique that we call *constraint differentiation*. This technique effectively integrates the lazy intruder and ideas from POR by using independence information from the second layer (the symbolic transition system) when searching the first layer (the constraint reduction).

Figure 3 illustrates the intuition behind constraint differentiation. Consider two symbolic states s' and s'' that can be reached from some state s by different interleavings of the same actions, e.g., message exchanges. In practice, we will see that s' and s'' almost never represent the same set of ground states. However, these sets often have substantial overlap. Constraint differentiation exploits these overlaps by restricting the constraints of s'' to those ground states that are not already covered by s', that is, the shaded part in Figure 3. The situation is symmetric of course and we could restrict s' instead of s''. But either way, the idea is the same: constrain the symbolic states so that the ground states in their intersection are represented by only one of them. In many cases, the restricted symbolic states have unsatisfiable constraints and therefore represent

[1] POR has also been suggested for model-checking approaches for security protocols that use a ground intruder model [17, 23]. However, since they do not address the prolific intruder problem, even with POR, such approaches have substantially poorer performance and scope than approaches based on symbolic intruder models.

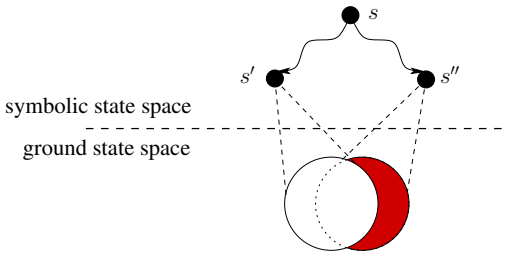

Figure 3: The intuition behind constraint differentiation.

the empty set of ground states. When we recognize this, we can eliminate the respective symbolic state (and all its successors) from the search tree without losing attacks. Also, even when the restricted symbolic state has satisfiable constraints, the restriction may allow other states to be removed later during the search.

Example 2. To illustrate a typical application of constraint differentiation, we return to our example. Assume that when the agent a receives a message of the form $\{|a, x|\}_k$, it replies with $\{|x|\}_{k'}$, for a key k'. Let us call this transition θ_1. Assume further that there is another agent b waiting for a message of the form $\{|b, z|\}_{k'}$ and who replies with $\{|z|\}_{k'}$. Let us call this transition θ_2. These two transitions can be performed in either order, although the resulting symbolic states will differ, as the intruder may use the reply message of one transition as input for the other transition. However, we can restrict one of the two executions to those ground states that are not represented by the other execution.

Consider applying this idea to the execution order where θ_1 precedes θ_2. The intruder must then use the outgoing message $\{|x|\}_{k'}$ of transition θ_1 to generate the incoming message $\{|b, z|\}_{k'}$ of transition θ_2. Assume the intruder does not know k', then the only way to do this is by unifying x and b, z. In particular, this forbids using any other message encrypted with k' that the intruder knew before. (This is not a restriction, because this solution is already covered by the alternative execution order where θ_1 follows θ_2.) Alternatively, if the intruder initially knows the message $\{|a, n|\}_k$ but does not know any other message encrypted with k, or the key k itself, then the constraint resulting from the first transition can only be satisfied with $x = n$. Therefore x and b, z do not unify and the resulting constraints are unsatisfiable. This shows how constraint differentiation can either limit the possible solutions for one execution order or even rule out a particular execution order. □

Constraint differentiation is a general technique that is independent of the technical and conceptual details of the various lazy intruder approaches and underlying protocol models. The basic prerequisite for constraint differentiation is a state space where states are represented symbolically using terms with variables, together with a state-transition function and a goal predicate, formalizing the search objective. Therefore, for the second layer of search, we abstract away

from particular base formalisms and work only with the requirement that the state space is represented symbolically, in the sense that it can be expressed using terms with variables, and attacks are formulated as reachability problems. To this end, we introduce the notion of a *symbolic transition system*, which abstracts from the nonessential differences of the different formalisms. For concreteness, we also show how our own lazy-intruder [9] as well as those developed by other researchers can be recast as symbolic transition systems, and can thus benefit from constraint differentiation.

Our work has both theoretical and practical relevance. Theoretically, our contribution is the formalization of constraint differentiation as a search reduction technique integrating the lazy intruder with ideas from POR. We formally prove that constraint differentiation terminates and is correct, in the sense that it preserves the set of reachable states so that all state-based properties holding before reduction (such as the existence of a state representing an attack) hold after reduction. It follows that constraint differentiation neither excludes attacks nor introduces new ones.

Our theoretical results have immediate practical applications. As a concrete example, we have integrated constraint differentiation into our on-the-fly model-checker OFMC [9], a state-of-the-art tool for security protocol analysis based on the lazy intruder. This integration leads to dramatic reductions in the size of the search spaces that must be searched and usually reduces the search time by several orders of magnitude. This extends the scope of OFMC so that it scales better when falsifying industrial-strength protocols. Moreover, it enables the use of OFMC as an effective verification tool (for a bounded number of sessions) since our reductions make it feasible to exhaustively search the (symbolic) state space resulting from several parallel executions of the protocol being analyzed. We have validated these findings by carrying out extensive testing: we have applied OFMC with and without constraint differentiation to a testsuite of 70 real-world protocols and we provide a detailed report on the results.

Organization.

We proceed as follows. In Section 2, we formalize the lazy intruder and symbolic transition systems. In Section 3, we integrate constraint differentiation into this formal setting. We present experimental results in Section 4 and draw conclusions in Section 5.

Note that this paper supersedes [8], where we originally presented constraint differentiation specialized to the particular lazy intruder of [9].

2 Constraint-Based Protocol Models

There is a large variety of both protocol models and lazy intruder approaches. Rather than tailoring constraint differentiation to a particular model and associated approach, we proceed more abstractly and formalize the core of constraint differentiation in a general way. This will allow us to focus on the main ideas

behind constraint differentiation and to state and prove properties about it in a way that is independent of minor variations in the underlying formalism. In fact, constraint differentiation can be applied to any lazy intruder approach that satisfies a set of basic assumptions that we will identify shortly. As a concrete example of such approaches, we consider our own lazy intruder from [9] and also describe how other approaches can be seen as instances of our abstract presentation in Appendix A.

We introduce the constraint-based model underlying our approach starting with the first layer of the search, the constraints and their reduction. Afterwards, for the second layer, we introduce symbolic transition systems, which abstract away the details of the formalism used to represent the symbolic state space.

2.1 Terms and Messages

As is standard in formal protocol analysis, messages are represented as *terms*, where $\mathcal{T}_\Sigma(\mathcal{V})$ denotes the set of all terms that can be built using function and constant symbols from a signature Σ and a set $\mathcal{V}$ of variables, disjoint from Σ. Terms without variables are called *ground* whereas terms containing variables are called *non-ground* or *symbolic*.

Constraint differentiation is independent of the choice of Σ, which depends on the particular intruder model considered. As a concrete running example, we will consider Σ_{ex}, which contains a set of constant symbols (such as the name of the intruder, i) and two binary function symbols $\langle\cdot,\cdot\rangle$ and $\{\!|\cdot|\!\}_\cdot$, which respectively denote concatenation and symmetric encryption.

Many protocol analysis approaches interpret terms in the free algebra, whereby syntactically different terms represent different messages. For some protocols, however, it is necessary to consider the algebraic properties of the cryptographic operators employed. For instance, protocols based on the Diffie-Hellman key-exchange require for exponentiation that

$$\exp(\exp(g,x),y) \approx \exp(\exp(g,y),x).$$

Here we have left the modulus implicit, assuming that exponentiation is always performed using the same (publicly known) modulus.

We do not focus on any particular algebraic theory in this paper and our intruder and protocol model will be parameterized by a congruence relation $\approx$ between terms. The free algebra is the special case where $s \approx t$ implies $s = t$. Usually, the algebraic properties are described by a set of equations E and we interpret terms in the *quotient algebra* $\mathcal{T}_{\Sigma/\approx_E}$, i.e., two terms are interpreted as equal iff that is a consequence of E. As constraint differentiation is independent of the particular E, in the following we will simply consider an arbitrary congruence relation $\approx$.

The notions of *atomic* and *composed message* are defined in the usual way as well as substitution, matching, and unification; see for example [7]. We denote the application of a substitution σ to a term t by writing $t\sigma$ and denote the composition of substitutions σ_1 and σ_2 by $\sigma_1\sigma_2$, that is $t(\sigma_1\sigma_2) = (t\sigma_1)\sigma_2$.

The *identity substitution id* is the substitution with the empty domain, that is, $domain(id) = \emptyset$. Also, as is often done in term rewriting [26], we consider only substitutions σ with the property $vars(range(\sigma)) \cap domain(\sigma) = \emptyset$.

2.2 The Dolev-Yao Intruder

We follow Dolev and Yao [20] and consider an intruder who controls the network, but cannot break cryptography. In particular, the intruder can intercept messages and analyze them if he possesses the corresponding keys for decryption, and he can generate messages from his knowledge and send them under any agent's name.

The precise definition of the intruder model depends on different choices such as the set of cryptographic operators that is considered and their algebraic properties. It is now standard to define the core of the intruder model, that is, the intruder's abilities to derive new messages from a given set of messages M, by an inductively defined closure operator, which we denote by $\mathcal{DY}(\cdot)$ (for Dolev-Yao).

Typically, four kinds of rules are used to define $\mathcal{DY}(\cdot)$. The first kind consists of the axiom that states that the closure contains the given set of messages M:

$$\frac{}{m \in \mathcal{DY}(M)} \ (m \in M)\,.$$

Second, there are rules for composing new terms from existing ones by applying an operation such as pairing or symmetric encryption:

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\langle m_1, m_2 \rangle \in \mathcal{DY}(M)} \qquad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{\!|m_1|\!\}_{m_2} \in \mathcal{DY}(M)}\,.$$

We can generalize this by introducing the notion of *intrudable* symbols, which are symbols representing those constants and functions of Σ that the intruder can use when composing messages. For instance, symmetric encryption and pairing should be intrudable, whereas the inverse function $\cdot^{-1}$ that maps public keys to private keys should not be. Moreover, we usually have some globally-known constants like i (the name of the intruder) that are intrudable, while most constants are not intrudable (e.g., those representing nonces and other fresh random numbers created by honest agents). Writing $intrudable(f)$ for an intrudable function f, we can now give just one rule for composing terms:

$$\frac{t_1 \in \mathcal{DY}(M) \quad \ldots \quad t_n \in \mathcal{DY}(M)}{f(t_1, \ldots, t_n) \in \mathcal{DY}(M)} \ (intrudable(f))\,.$$

Third, there may be rules for decomposing messages, whereby the intruder can obtain subterms of a known term. For pairing and symmetric encryption, we have:

$$\frac{\langle m_1, m_2 \rangle \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)} \qquad \frac{\{\!|m_1|\!\}_{m_2} \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{m_1 \in \mathcal{DY}(M)}\,.$$

Fourth, unless terms are interpreted in the free algebra, the intruder deduction must also be closed under $\approx$. It is not possible in general to decouple deduction and algebraic equivalences, so that we cannot simply build the closure under $\approx$ after $\mathcal{DY}(\cdot)$. Thus, we need the following rule:

$$\frac{t \in \mathcal{DY}(M)}{s \in \mathcal{DY}(M)} \ (s \approx t)\,.$$

One can use algebraic properties to model decryption explicitly. For instance,

$$\{|\{|m|\}_k|\}_k \approx m$$

expresses that decryption with the key k of a message m encrypted with k (i.e., performing symmetric encryption twice with the same key) yields the original message m. Using such equations, and given that the decryption operations are intrudable, the two deduction rules given above for decomposing pairs and symmetric encryptions are redundant. However, handling equations is, in practice, more difficult than working in the free algebra and hence rules are preferred to equations, whenever possible.

For constraint differentiation, we allow an arbitrary intruder model $\mathcal{DY}(\cdot)$, provided that the following assumptions are met:

1. The closure contains the initial set of messages, i.e., $M \subseteq \mathcal{DY}(M)$.
2. The name of the intruder i is an intrudable constant symbol of Σ.
3. $\mathcal{DY}(\cdot)$ is closed under the application of intrudable function symbols.
4. $\mathcal{DY}(\cdot)$ is closed under $\approx$.

Note that assumptions 2 and 3 together imply that $\mathcal{DY}(\cdot)$ is never empty.

The $\mathcal{DY}(\cdot)$ closure operator may be applied to sets of terms that contain variables. However, the rules of the Dolev-Yao intruder do not instantiate variables, and thus a derivation of $t \in \mathcal{DY}(IK)$ for a symbolic term t and a set of symbolic terms IK means that the derivation of t is possible under every instantiation of the variables, that is, $t\sigma \in \mathcal{DY}(IK\sigma)$ for all substitutions σ. In contrast, the lazy intruder approach described below tackles the problem of finding instantiations of the variables under which a derivation is possible. Therefore, one may say that variables in $\mathcal{DY}(\cdot)$ derivations are implicitly universally quantified, while they are existentially quantified in the lazy intruder approach.

2.3 The Lazy Intruder

The Dolev-Yao intruder leads to infinite branch-points in the search tree when one naïvely enumerates all messages that the intruder can send. The lazy intruder technique significantly reduces the search tree (without excluding any potential attacks) by exploiting the fact that the actual value of certain parts of a message is often irrelevant for the receiver. So, whenever the receiver will not

further analyze the value of a particular message part, we can postpone during the search the decision about which value the intruder actually chooses for this part by replacing it with a variable and a *constraint* recording which messages the intruder may use to generate the message part. We express this information using constraints of the form $\mathit{from}(T; IK)$, meaning that T is a set of terms generated by the intruder from the set of his known messages IK (standing for "intruder knowledge").

Definition 1. *An* atomic *from* constraint *is an expression of the form*

- $\mathit{from}(T; IK)$ *where* T *and* IK *are finite sets of terms.*

An atomic constraint *is an expression of the form*

- *the constant* TRUE*,*
- *the constant* FALSE*, or*
- *an atomic from constraint.*

A constraint *is either*

- *an atomic constraint, or*
- *the conjunction* $C_1 \wedge C_2$ *of two constraints* C_1 *and* C_2*.*

In a constraint $\mathit{from}(T; IK)$*, we refer to* T *as the* term part *and to* IK *as the* intruder-knowledge part *of the constraint.*

Let C *be a constraint and* σ *be a substitution, where* $\mathit{vars}(C) \cap \mathit{domain}(\sigma) = \emptyset$*. Then we call the pair* (C, σ) *a* constraint store*, which we use to keep track of the substitutions generated during constraint reduction. A constraint* C *is called* simple*, written* $\mathit{simple}(C)$*, if*

- $C = \mathsf{TRUE}$*,*
- $C = \mathit{from}(T; IK)$ *and* $T \subseteq \mathcal{V}$*,*
- $C = C_1 \wedge C_2$ *and* C_1 *and* C_2 *are both simple.*

A constraint store (C, σ) *is simple iff* C *is simple.*

An interpretation $\mathcal{I}$ *is a total function from* $\mathcal{V}$ *to* $\mathcal{T}_\Sigma$*, that is, it assigns a ground term to each variable. For a term* t *and an interpretation* $\mathcal{I}$*, we define*

$$t^{\mathcal{I}} = \begin{cases} \mathcal{I}(t) & \text{if } t \in \mathcal{V}\,, \\ f(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}) & \text{if } t = f(t_1, \ldots, t_n)\,. \end{cases}$$

We now define an (over-loaded) entailment relation between an interpretation on the one side and a constraint, a substitution, or a constraint store on

the other side:

$$\begin{aligned}
\mathcal{I} &\models \mathsf{TRUE} \\
\mathcal{I} &\not\models \mathsf{FALSE} \\
\mathcal{I} \models \mathit{from}(T; IK) \quad &\mathit{iff} \quad T^{\mathcal{I}} \subseteq \mathcal{DY}(IK^{\mathcal{I}}) \\
\mathcal{I} \models \phi \wedge \psi \quad &\mathit{iff} \quad \mathcal{I} \models \phi \text{ and } \mathcal{I} \models \psi \\
\mathcal{I} \models \sigma \quad &\mathit{iff} \quad \text{there is a substitution } \tau \text{ such that } x\sigma\tau \approx x^{\mathcal{I}} \\
&\qquad\quad \text{for all } x \in \mathit{domain}(\sigma) \\
\mathcal{I} \models (C, \sigma) \quad &\mathit{iff} \quad \mathcal{I} \models C \text{ and } \mathcal{I} \models \sigma
\end{aligned}$$

In general, the definition of $\models$ *extends to sets* S *of constraints, substitutions, or constraint stores as follows:*

$$\mathcal{I} \models S \quad \mathit{iff} \quad \mathcal{I} \models e \text{ for some } e \in S$$

A constraint C *is* satisfiable *iff* $\mathcal{I} \models C$ *for some interpretation* $\mathcal{I}$*. We overload* $\models$ *further and also use it for logical entailment and equivalence between constraints.*

$$\begin{aligned}
\phi \models \psi \quad &\mathit{iff} \quad \mathcal{I} \models \phi \text{ implies } \mathcal{I} \models \psi \text{ for every } \mathcal{I} \\
\phi =\!\models \psi \quad &\mathit{iff} \quad \phi \models \psi \text{ and } \psi \models \phi
\end{aligned}$$

These two definitions also extend, straightforwardly, to sets of constraints, substitutions, and constraint stores.

We will silently assume throughout the paper that constraints are always normalized using the following (convergent) set of rewrite rules:

- $C \wedge \mathsf{TRUE}$ is rewritten to C,
- $C \wedge \mathsf{FALSE}$ is rewritten to FALSE.

Observe that every simple constraint is satisfiable since the intruder can always generate some message, such as his own name. Now, the core idea of the lazy intruder is to reduce a given constraint store into an equivalent set of simple constraint stores. In particular, the resulting set of constraint stores is empty iff the given constraint store is unsatisfiable. The reduction rules themselves are of the form

$$\frac{C', \sigma'}{C, \sigma}\, r\ (\Phi)\,,$$

where (C, σ) and (C', σ') are constraint stores and Φ is a side condition on the application of r.

Such a rule expresses that (C', σ') can be *derived* from (C, σ) and therefore constraint reduction rules are applied bottom-up, which we denote by $(C, \sigma) \vdash_r (C', \sigma')$. Note that rules contain rule variables that should not be confused with the (term) variables in constraints. For instance, in the rule G given in Definition 2 below, we have rule variables $t, t_1, \ldots, t_n, T, IK, C, \sigma$, and τ. When

applying rules, we only substitute rule variables, not term variables. Note also that rule matching is performed modulo the properties of conjunction and sets, e.g., $\mathit{from}(\{\{\!|m|\!\}_k\}; \ldots)$ matches $\mathit{from}(\{t\} \cup T; \mathit{IK}) \wedge C$ under the substitution $[t \mapsto \{\!|m|\!\}_k, T \mapsto \emptyset, \mathit{IK} \mapsto \ldots, C \mapsto \mathsf{TRUE}]$.

We need not commit to the formalism used to express and reason about side conditions, and any logic or theory capable of reasoning about sets of terms and unifiers would suffice (e.g., higher-order logic or set theory). We will employ, as standard, the notions of satisfiability and logical equivalence, when speaking of side conditions. As notation, we write $\Phi_1 \equiv \Phi_2$ to denote that two side-conditions Φ_1 and Φ_2 are *logically equivalent* and have the same set of free variables. We also extend this to rules: two rules r_1 and r_2 are *equivalent* iff for all constraint stores $\mathcal{C}$ and $\mathcal{C}'$, it holds that $\mathcal{C} \vdash_{r_1} \mathcal{C}'$ iff $\mathcal{C} \vdash_{r_2} \mathcal{C}'$. We say that the rule r *entails* the rule r' iff $(C, \sigma) \vdash_{r'} (C', \sigma')$ implies $(C, \sigma) \vdash_r (C', \sigma')$ for all C, C', σ, and σ', i.e., r' constitutes a special case of r.

For a set of rules R, $\vdash_R$ is the union of $\vdash_r$ for every $r \in R$. We write $\vdash_R^+$ and $\vdash_R^*$ to denote the transitive and reflexive-transitive closure of $\vdash_R$, respectively.

We will assume that for every rule

$$\frac{\mathit{premise}}{\mathit{conclusion}}\ r\,(\Phi)$$

it is decidable whether a given constraint store matches *conclusion* and that Φ is satisfiable under that match. Moreover, for any such match, we assume that we can compute all instances of $\mathit{vars}(\mathit{premise}) \setminus \mathit{vars}(\mathit{conclusion})$ such that Φ holds. This is necessary in order to compute the set of all constraint stores that can be reached by applying this rule to a given constraint store.

All existing lazy intruder approaches restrict themselves to a particular class of constraints, often called *well-formed* constraints. For instance, one usually requires that variables only originate from the intruder, that is, one can order the constraints such that every variable first occurs in the term-part of a constraint and that the intruder knowledge is monotonically increasing. Constraint differentiation is a general technique that is also independent from the particularities of well-formedness, and we will thus only assume that such a notion is given together with the considered lazy intruder so that

- all reduction rules preserve the well-formedness and
- the lazy intruder is correct and terminating for well-formed constraints.

Rather than committing to a particular variant of the lazy intruder and its concrete reduction strategies, we give below three schemata for lazy intruder reduction rules. As will be shown in Appendix A, instances of these schemata capture the spirit of different lazy intruder approaches.

2.4 Constraint Reduction Rules

We now introduce our abstract formalization of the lazy intruder by defining three categories of constraint reduction rules. Our rules have side conditions,

which constrain the values that can be substituted for the rule's variables. In particular, given two side conditions Φ and Ψ with the same set of variables, we say that Φ *implies* Ψ iff those values satisfying Φ also satisfy Ψ. We say that *a rule r is an instance of a rule r'* iff r and r' have identical premises and conclusions, and the side condition of r implies the side condition of r'. Note that if r is an instance of r' then r is entailed by r', but not vice versa.

Each of the three rule categories is defined by an abstract rule, where the category consists of the set of instances of that rule.

2.4.1 Generation Rules

Generation rules express that the intruder can compose terms if he knows the subterms and the composition function is intrudable.

Definition 2. *A* generation rule *has the form*

$$\frac{(\mathit{from}(\{t_1, \ldots, t_n\} \cup T; \mathit{IK}) \wedge C)\tau, \sigma\tau}{\mathit{from}(\{t\} \cup T; \mathit{IK}) \wedge C, \sigma} \; G\,(\Phi),$$

where the side condition Φ implies

$$t\tau \approx f(t_1, \ldots, t_n)\tau$$

for an intrudable symbol f.

As an example, consider the following rule from [9]:

$$\frac{\mathit{from}(\{t_1, t_2\} \cup T; \mathit{IK}) \wedge C, \sigma}{\mathit{from}(\{\{\!|t_1|\!\}_{t_2}\} \cup T; \mathit{IK}) \wedge C, \sigma} \; G^L_{\text{scrypt}}\,,$$

which expresses that the intruder can generate the symmetric encryption of a message t_2 using a message t_1 as the key, provided that he knows t_1 and t_2. This rule is equivalent to the rule G with

$$\Phi \equiv t = \{\!|t_2|\!\}_{t_1} \wedge n = 2 \wedge \tau = \mathit{id}\,,$$

and is thus an instance of G since $\{\!|\cdot|\!\}_.$ is an intrudable function symbol.[2]

2.4.2 Unification Rules

Unification rules express an alternative way for the intruder to generate a term t of a particular form: he can take any term from his knowledge that can be unified with t.

[2] When considering the free algebra, the substitution τ in the generation rules is always the identity. However, when using equations, the generation steps may require substitutions. For example, when considering the properties of (modular) exponentiation, one way to generate the term B^y is based on the substitution $[B \mapsto G^X]$ (for two fresh variables G and X), and generating the subterms G^y and X.

Definition 3. *A* unification rule *has the form*

$$\frac{(\mathit{from}(T;\{s\}\cup IK)\wedge C)\tau,\sigma\tau}{\mathit{from}(\{t\}\cup T;\{s\}\cup IK)\wedge C,\sigma}\ U\ (\Phi)\,,$$

where the side condition Φ *implies that* $s\tau \approx t\tau$*.*

As an example, consider the following rule from [9]:

$$\frac{(\mathit{from}(T;\{s\}\cup IK)\wedge C)\tau,\sigma\tau}{\mathit{from}(\{t\}\cup T;\{s\}\cup IK)\wedge C,\sigma}\ G^{L}_{\text{unif}}\ (\tau = \mathit{mgu}(s,t)\wedge t\notin\mathcal{V})\,,$$

where $\mathit{mgu}(s,t)$ is the most general unifier (in the free algebra) between s and t, if it exists; note that in the case of algebraic properties, there may not be a single most general unifier, but rather a set of most general unifiers. The rule G^{L}_{unif} is equivalent to U with

$$\Phi \equiv \tau = \mathit{mgu}(s,t)\wedge t\notin\mathcal{V}\ .$$

Here we have the restriction that the unified term in the constraint's term-part is not a variable. It is this restriction that makes the intruder lazy: if the term to be generated is a variable, then the intruder can take any term to be t, and thus the procedure does not need to explore all the possible messages in the intruder knowledge.

2.4.3 Analysis Rules

Analysis rules express that the intruder can obtain subterms of known terms under certain conditions. For example, the intruder can obtain the clear-text of an encrypted message when he knows the decryption key.

Definition 4. *An* analysis rule *has the form*

$$\frac{(\mathit{from}(\{k\};\{m^{?}\}\cup IK)\wedge \mathit{from}(T;\{m,r\}\cup IK)\wedge C)\tau,\sigma\tau}{\mathit{from}(T;\{m\}\cup IK)\}\wedge C,\sigma}\ A\ (\Phi)\,,$$

where the side condition Φ *implies that* $r\tau\in\mathcal{DY}(\{m\tau,k\tau\})$*. That is, the result of the analysis is something that can indeed be derived from the message* m *and the "key"* k*. The expression* $\{m^{?}\}$ *expresses that the rule schema admits rules that include* m *at this position and rules that do not.*

As an example, consider the following rule from [9]:

$$\frac{\mathit{from}(\{k\};IK)\wedge \mathit{from}(T;\{\{\!|r|\!\}_k,r\}\cup IK)\wedge C,\sigma}{\mathit{from}(T;\{\{\!|r|\!\}_k\}\cup IK)\wedge C,\sigma}\ A^{L}_{\text{scrypt}}\,(r\notin IK).$$

This rule describes how the intruder can (attempt to) decrypt a known message $\{\!|r|\!\}_k$: a new constraint is added, namely that the key k can be generated from the knowledge IK and the clear-text r is added to the intruder knowledge. Note

that the new constraint may be unsatisfiable. If the intruder cannot generate k, then this analysis step leads to a dead-end path in the reduction search space. This rule is equivalent to A with

$$\Phi \equiv m = \{|r|\}_k \wedge r \notin IK \wedge \tau = id \ ,$$

and where $\{m^?\} = \{m\}$, i.e., we include m at this position.

The reason that m is optional is that several existing lazy intruder approaches [1, 9, 12, 16, 18, 22, 24, 30] make different choices here depending on their particular reduction strategy (which we do not prescribe here). We will return to the issue of strategies in more detail in Appendix A. The example rule A^L_{scrypt} is an analysis rule, where again the substitution τ of the general rule is trivially the identity in the example, as in the example for the generation rules above. Similarly, the substitution is only used when considering operators with algebraic properties.

We can now define what constitutes a correct and terminating reduction procedure.

Definition 5. *For a constraint store (C, σ) and a set of rules R, let*

$$Red_R(C, \sigma) = \{(C', \sigma') \mid (C, \sigma) \vdash_R^* (C', \sigma') \wedge simple(C')\}$$

be the set of all derivable simple constraints. We also call Red_R the reduction procedure induced by R.

Given a notion of well-formedness, we say that Red_R is terminating *iff $\vdash_R$ is well-founded on well-formed constraint stores(where a constraint store (C, σ) is well-formed iff C is). In other words, there is no infinite chain $(C_1, \sigma_1) \vdash_R (C_2, \sigma_2) \vdash_R \ldots$, where C_1 is well-formed.*

Further, we say that Red_R is sound *(respectively,* complete*) iff $Red_R(C, \sigma) \models (C, \sigma)$ (respectively, $(C, \sigma) \models Red_R(C, \sigma)$) for every well-formed constraint store (C, σ). Red_R is called* correct *iff it is sound and complete. We will write $Red_R(C)$ as shorthand for $Red_R(C, id)$.*

For the rest of this paper, we will assume that we are given a set of lazy intruder rules and a notion of well-formedness such that Red_R is correct and terminating.

2.5 Symbolic Transition Systems

The various implementations of the lazy intruder are based on different formalisms representing the second layer of the search, namely the symbolic state space and its transition relation. For example, [9, 16] use multiset rewriting, [18, 30] use strand spaces, and [1, 12] use process calculi. Constraint differentiation is not specialized to any of these approaches. It only requires a state space where states are represented symbolically using terms with variables, together with a state-transition function and a goal predicate that formalizes the search objective, i.e., characterizes attacks. Therefore, we define the notion of a symbolic transition system that abstracts away from many of the differences of the different formalisms proposed.

The idea behind the symbolic approaches is to use symbolic states, which are terms with variables, to represent sets of ground states, i.e., sets of ground terms. Not all substitutions for variables are allowed and the permissible substitutions are described by some kind of constraints, in our case *from* constraints. The semantics of a particular constraint (i.e., the set of substitutions allowed by the constraint) can then be extended to a symbolic state s: the semantics of s is the set of ground terms represented by s. One can then define a state-transition function as usual, but it must agree with the semantics of each symbolic state in the sense that equivalent symbolic states have equivalent successors and are equivalent with respect to the goal predicate, which in our case describes attacks. Formally:

Definition 6. *A* symbolic transition system *over a countable set Σ of constant and function symbols and a countable set $\mathcal{V}$ of variables is a 5-tuple $(\mathcal{G}, \mathcal{S}, \mathcal{S}_0, \mathcal{T}, \mathcal{P})$, where*

- $\mathcal{G} = T_\Sigma$ *is the* set of ground states*;*
- $\mathcal{S} = T_\Sigma(\mathcal{V}) \times \mathcal{FC}$ *is the* set of symbolic states*, where $\mathcal{FC}$ denotes the* set of all well-formed *from constraints;*
- $\mathcal{S}_0 \in \mathcal{S}$ *is the* initial symbolic state*;*
- $\mathcal{T} : \mathcal{S} \to 2^\mathcal{S}$ *is a* transition function *on symbolic states; and*
- $\mathcal{P}$ *is a* predicate *on symbolic states.*

We also refer to $\mathcal{G}$ and $\mathcal{S}$ as the spaces *of* ground states *and* symbolic states *respectively, and we call $\mathcal{P}$ the* attack predicate *as it will be used to specify symbolic states representing attacks.*

The semantics of a symbolic state $s = (t, C)$ is defined in terms of the semantics of the constraints:

$$[\![(t, C)]\!] = \{t^\mathcal{I} \mid \mathcal{I} \models C\} .$$

We straightforwardly extend $\mathcal{T}$, $\mathcal{P}$, and $[\![\cdot]\!]$ to sets of symbolic states. The symbolic transition system must agree with the semantics of the symbolic states in the following sense: for two sets of symbolic states S_1 and S_2 with $[\![S_1]\!] = [\![S_2]\!]$, we require that $[\![\mathcal{T}(S_1)]\!] = [\![\mathcal{T}(S_2)]\!]$ and $\mathcal{P}(S_1) \iff \mathcal{P}(S_2)$.

The set of reachable symbolic states *is the smallest set that contains the initial symbolic state and is closed under the transition function. A symbolic transition system is* secure *iff no reachable symbolic state satisfies the attack predicate.*

Constraint reduction is extended to symbolic states by

$$Red_R(t, C) = \{(t', C') \mid \exists\sigma.\ (C', \sigma) \in Red_R(C)\ \wedge\ t' = t\sigma\} .$$

Hence, we have that $[\![Red_R(s)]\!] = [\![s]\!]$ for every symbolic state s iff Red_R is correct.

The other lazy intruder approaches can be easily recast as symbolic transition systems. For example, in the concrete case of the model underlying the OFMC tool [9], the ground states in $\mathcal{G}$ are sets of facts that express the local state of agents, the intruder knowledge, and the messages sent on the network that are not yet received. The symbolic states in $\mathcal{S}$ are also sets of facts, but message terms may contain variables. Hence, a symbolic state represents the set of ground states that is obtained by applying ground substitutions to the state's variables. The transition function on symbolic states $\mathcal{T}$ is defined by set rewrite rules that describe the behavior of the honest agents and add new constraints by conjunction to the current constraint when the intruder generates a new message (note that, by the construction of this transition function, the constraint conjunction is well-formed in all reachable symbolic states). Finally, the attack predicate $\mathcal{P}$ is used to express state-based properties of symbolic states. In OFMC, $\mathcal{P}$ formulates (the negation of) standard authentication and secrecy goals. However, in the abstract symbolic transition system above, we neither need commit to any particular kind of attack nor to a particular formalism to specify attacks (any of the other formalisms previously listed, such as strands, could be used). All that is required is that attacks are formalizable as reachability problems.

A symbolic transition system gives rise to a search tree where the root node is the initial state and the children of a node are all states that can be reached in one transition. For every symbolic state, applying the function Red_R yields a set of semantically equivalent symbolic states with simple constraints. This can be exploited for reduction since if the constraint C of a symbolic state is unsatisfiable, then $Red_R(C) = \emptyset$ by the correctness of Red_R. In this case, we can safely prune the subtree of the search tree node containing the unsatisfiable constraint. In the next section, we will see that the integration of constraint differentiation into the symbolic transition system is based on a similar kind of pruning.

Before considering this integration in detail, let us first observe that the lazy intruder can be straightforwardly extended with a technique that we call *step-compression*, which can significantly reduce the size of the search tree without excluding any attacks.[3] Step-compression is based on the idea that, since the intruder completely controls the communication network, we can safely assume that every message from an honest agent is automatically intercepted by the intruder (who can always play it back into the network) and that every message that an honest agent receives comes directly from the intruder. This allows us to restrict the search to transitions where two steps are merged (or "compressed") into one: first, the intruder sends a message to an honest agent and second, the intruder intercepts the agent's reply. The proof that this does not exclude any attacks or introduce new ones can be found in [33].

When step-compression is used, the symbolic transition system has the following property: for every transition from a symbolic state $s_1 = (t_1, C_1)$

[3]Note that step-compression is applied in all symbolic approaches that we know of [1, 9, 12, 18, 22, 24, 30], as well as in some non-symbolic approaches, such as [3].

Σ	Set of symbols such that $\{\mathsf{i}\} \in \Sigma$.
$\mathcal{V}$	Set of variables such that $\Sigma \cap \mathcal{V} = \emptyset$.
$\mathcal{T}_\Sigma(\mathcal{V})$	Set of terms with variables in $\mathcal{V}$.
$\approx$	Congruence relation on $\mathcal{T}_\Sigma(\mathcal{V})$.
intrudable$(\cdot)$	Predicate in Σ, where *intrudable*(i).
$\mathcal{DY}(\cdot)$	Dolev-Yao closure operator. We assume that for any set M of terms, $\mathcal{DY}(M)$ contains M and is closed under $\approx$ and composition with intrudable functions.
well-formed$(\cdot)$	Predicate on both constraints and constraint stores.
R	A finite set of lazy intruder rules (i.e., instances of the G, U, and A rule schemata) such that the following holds: • For all well-formed constraint stores (C, σ), $Red_R(C, \sigma)$ is correct (i.e., $Red_R(C, \sigma) =\!\!\models (C, \sigma)$) and terminating. • For each side condition of rules in R, the set of rule matches are computable (i.e., the set of matches of the rules' meta-variables with a given constraint store). • If (C, σ) is well-formed and $(C, \sigma) \vdash_R^* (C', \sigma')$, then (C', σ') is also well-formed.
$(\mathcal{G}, \mathcal{S}, \mathcal{S}_0, \mathcal{T}, \mathcal{P})$	A symbolic transition system according to Definition 6.

Table 1: Parameters of the abstract lazy intruder approach and our assumptions about their instances.

to a symbolic state $s_2 = (t_2, C_2)$, the constraints will have the form $C_2 = C_1 \wedge \mathit{from}(\{m_1\}; \mathit{IK})$ for some message m_1, representing the message the intruder sends to an honest agent, and a set of messages IK, representing the knowledge the intruder can use to generate m_1. Also, the intruder knowledge in s_2 is augmented by the agent's reply. We will make use of this property in constraint differentiation.

To summarize, we have so far presented an abstract formalization of the lazy intruder and the symbolic, constraint-based approach, independent from the concrete intruder model and the protocol model. Roughly speaking, we assume that we are given an arbitrary instance of this general concept and that this instance is already correct and terminating with respect to a particular intruder model and a given notion of well-formedness. We show in the following section how to apply the constraint differentiation technique to such an instance and prove the correctness and termination of the integration. Table 1 lists all the elements that must be instantiated for a concrete lazy intruder approach, and the properties we assume about this instance.

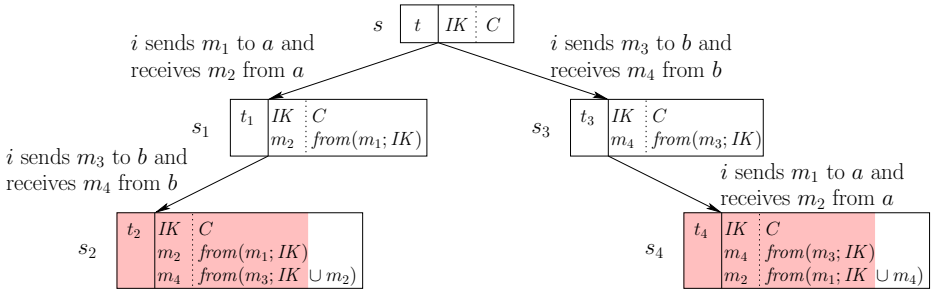

Figure 4: An illustration of constraint differentiation for $t_2 = t_4$. For each symbolic state $s = (t, C)$, we display t in the first column, the intruder knowledge IK in the second column, and the associated constraints C in the third column.

3 Constraint Differentiation

The lazy intruder technique allows one to significantly reduce the size of the search tree generated by the prolific Dolev-Yao intruder without excluding any attacks. In particular, our experiments [9] have shown that the search tree induced by the lazy intruder often has roughly the same size as the tree that would be searched when considering a *passive* intruder, i.e., one that listens to the communication on the network but does not manipulate or generate any messages. This is the maximal reduction possible since the search tree of reachable symbolic states must cover all possible executions of the protocol between honest agents.

However, even when considering only a small number of sessions that can be executed in parallel, searching the tree that contains all the interleavings of these sessions may still be infeasible. In model-checking approaches for concurrent systems, this problem is often handled using *partial-order reduction* (*POR*), a technique that reduces the number of interleavings that need to be considered by exploiting independencies between the possible transitions [34].

One might expect that the direct combination of the lazy intruder with partial-order reduction would counter both the problem of the prolific Dolev-Yao intruder and the large number of interleavings. However, the direct combination of these two techniques is not effective: the different transitions of the lazy intruder rarely lead to the same symbolic successor state and therefore there is practically no independence of transitions that can be exploited by POR. We address this problem by directly incorporating independence information in the constraint reduction. The result is a POR-inspired reduction technique that we call *constraint differentiation*.

To see why the direct combination of partial-order reduction with the lazy intruder is not effective, consider the symbolic transition system (with step-compression) given in Subsection 2.5 which gives rise to a search tree. A direct application of POR would require identifying situations of the form depicted in Figure 4. There are two sequences of transitions. In the left one, the intruder i

first sends a message m_1 to an agent a, receiving the answer m_2, and afterwards he sends a message m_3 to an agent b, receiving the answer m_4. In the right sequence, the intruder first talks to b and then to a. The transitions result in the states $s_2 = (t_2, C_2)$ and $s_4 = (t_4, C_4)$, containing the symbolic terms t_2 and t_4 and the constraints C_2 and C_4. We consider the case $t_2 = t_4$, which holds when the transitions are independent in the sense that on ground states the respective order of operations would lead to the same successor states. In this case, for every substitution σ, the represented ground states $t_2\sigma$ and $t_4\sigma$ are identical. However, the constraints, which determine the set of permissible substitutions, are different due to the fact that the intruder generated the respective messages from different sets of messages, i.e., with different states of intruder knowledge. Hence, the direct combination of partial-order reduction with the lazy intruder is ineffective. Note that this problem is independent of the use of step-compression. It is easy to show that directly applying POR to a symbolic transition system without step-compression can only result in reductions that are also achieved using step-compression.

To see, however, that there are redundancies that we can exploit, observe that there is an overlap in the set of ground states. This is depicted by the two symbolic states s_2 and s_4, as shown by the shaded part in Figure 4. This overlap represents all those ground states in the semantics of the symbolic states that do not exploit the intruder's new knowledge of m_2 or m_4. The key idea behind constraint differentiation is that we can use the independence of transitions by exploiting precisely this overlap. If, for example, we favor the left sequence, then for the state s_4 reached in the right sequence we will only be interested in solutions that are not already subsumed by s_2, that is, those solutions where the intruder actually uses the message m_4 that he learned in the first transition to generate the message m_1 in the second transition.

By exploiting this idea, we can propagate information about independent transitions obtained in the second search layer, the symbolic transition system, to the first layer, the constraint reduction. We exploit the fact that we only need to consider solutions for a given constraint that are obtained by using *new* intruder knowledge. In the example, we could express the fact that the message m_4 needs to be used when creating m_1 by using constraints of the form $\mathit{D\text{-}from}(\{m_1\}; IK; \{m_4\})$, which intuitively has the same meaning as the constraint $\mathit{from}(\{m_1\}; IK \cup \{m_4\})$, except that we exclude all solutions of $\mathit{from}(\{m_1\}; IK)$.

Mirroring the development of Section 2, we proceed as follows. First, we formalize the new kinds of constraints for constraint differentiation, called *D-from* constraints. Second, we define how reduction rules for *from* constraints are translated to rules for *D-from* constraints. Third, we define a constraint reduction function $\mathit{D\text{-}Red}_R$ for *from* and *D-from* constraints, based on the translated reduction rules. Fourth, we show that if Red_R is correct and terminating for well-formed *from* constraints, then so is $\mathit{D\text{-}Red}_R$ for well-formed *from* and *D-from* constraints. We conclude this section by integrating constraint differentiation into the second layer of the search as a transformation of the search tree induced by the symbolic transition system.

3.1 Constraint Reduction with Constraint Differentiation

We now formalize *D-from* constraints and extend related definitions.

Definition 7. *An* atomic *D-from* constraint *is an expression of the form*

$$\textit{D-from}(T; \textit{OIK}; \textit{NIK}),$$

where T, $\textit{OIK}$, and $\textit{NIK}$ are sets of messages. We extend the definition of atomic constraints (and thus also constraints) from Definition 1 with atomic D-from constraints.

As terminology, when we refer to constraints *we mean the combined use of from and D-from constraints, and we use from* constraints *(respectively, D-from* constraints*) to refer to constraints where the atomic constraints contain no D-from (respectively, from) constraints.*

We extend the models relation to constraints of the form $C = \textit{D-from}(T; \textit{OIK}; \textit{NIK})$ as follows:

$$\mathcal{I} \models C \quad \textit{iff} \quad \mathcal{I} \models \lceil C \rceil \textit{ and } \mathcal{I} \not\models \lfloor C \rfloor ,$$

where

$$\begin{aligned} \lceil \textit{D-from}(T; \textit{OIK}; \textit{NIK}) \rceil &= \textit{from}(T; \textit{OIK} \cup \textit{NIK}) \\ \lfloor \textit{D-from}(T; \textit{OIK}; \textit{NIK}) \rfloor &= \textit{from}(T; \textit{OIK}) \end{aligned}$$

are functions mapping D-from constraints to from constraints. We say that $\textit{D-from}(T; \textit{OIK}; \textit{NIK})$ is simple *iff $T \subseteq \mathcal{V}$ and $T \neq \emptyset$.*

Similarly, the definitions of $\lceil \cdot \rceil$, $\lfloor \cdot \rfloor$, and simple are extended straightforwardly to constraint stores. Also, we have assumed that for from constraints, we have a notion of well-formedness. We lift this notion to D-from constraints as follows: the constraint C is well-formed iff $\lceil C \rceil$ is well-formed. Furthermore, when we write symbolic transition system *(Definition 6) we now include systems whose constraints may also be D-from constraints.*

Intuitively, *NIK* represents new messages that are not in *OIK*. The acronyms stand for *new* and *old intruder knowledge*, respectively. As we will use them, these sets will always be disjoint. The constraint $\textit{D-from}(T; \textit{OIK}; \textit{NIK})$ formalizes that the set of terms T must be generated by the intruder using the knowledge in the set $\textit{OIK} \cup \textit{NIK}$, but we are only interested in solutions that employ new information in *NIK* and hence we exclude all solutions of $\textit{from}(T; \textit{OIK})$. The function $\lceil \cdot \rceil$ yields a *from* constraint by removing the requirement on new knowledge; therefore $C \models \lceil C \rceil$. Similarly, $\{\lfloor C \rfloor, C\} \not\models \lceil C \rceil$, as $\lfloor \cdot \rfloor$ returns the solutions removed from $\lceil C \rceil$. Note that for a simple constraint C, both $\lceil C \rceil$ and $\lfloor C \rfloor$ are simple (and hence satisfiable). Unlike for *from* constraints, it does not always hold that every simple *D-from* constraint C is satisfiable, although this is usually the case.

3.2 *D-Red*$_R$ based on *Red*$_R$

We now show how to obtain a reduction procedure for *D-from* constraints, given a reduction procedure for *from* constraints. More precisely, we translate reduction rules for *from* constraints to rules for *D-from* constraints. By taking the union of the original rules and the translated rules, we obtain a new procedure *D-Red*$_R$ for reducing *from* and *D-from* constraints.[4] We then show that *D-Red*$_R$ is correct and terminating, whenever *Red*$_R$ is.

A central feature of our formalization of constraint differentiation is that it is independent of the details and concrete strategy taken by a particular lazy intruder implementation. Intuitively, we can understand this as follows. Given a constraint store (C, σ) with a non-simple C, the procedure *D-Red*$_R$ first checks which reductions the original procedure *Red*$_R$ would choose when applied to $(\lceil C \rceil, \sigma)$. Then, it applies reductions to (C, σ) that are analogous (in a sense to be made precise later) to those chosen by the original procedure, where it eliminates those reductions that are not possible with respect to (C, σ). (Note that $(C, \sigma) \models (\lceil C \rceil, \sigma)$.)

To make our construction independent of the details of the particular lazy intruder approach taken, we require that all approach-specific aspects are formalized within the side conditions of the rules. Hence, when transforming the rules, we must also formalize how their side conditions are translated so that they are indeed used "analogously" in constraint differentiation. To this end, observe that the transformed rules work on *D-from* constraints rather than *from* constraints and in *D-from* constraints the intruder knowledge (previously referred to by the variable *IK*) is now split into two parts, *OIK* and *NIK*. Recall that the side conditions describe those values that the rule variables can take. Therefore the translation of the side conditions affects only those variables that represent constraints or the intruder knowledge of a constraint. These are the variables C and *IK*. We therefore define

$$\langle\!|\Phi|\!\rangle = \Phi[C \mapsto \lceil C \rceil, IK \mapsto OIK \cup NIK] \,.$$

We will show in Lemma 1 that this translation of side-conditions ensures that the translated rules are used (on *D-from* constraints) in a way analogous to the original rules (on *from* constraints).

3.2.1 Generation Rules

Consider a generation rule, which has the form

$$\frac{(\mathit{from}(\{t_1, \ldots, t_n\} \cup T; IK) \wedge C)\tau, \sigma\tau}{\mathit{from}(\{t\} \cup T; IK) \wedge C, \sigma} \; G\,(\Phi)\,,$$

where the side condition Φ implies

$$t\tau \approx f(t_1, \ldots, t_n)\tau \;,$$

[4] Alternatively, it would suffice to only use the new rules and to consider every constraint of the form $\mathit{from}(T; IK)$ simply as syntactic sugar for the constraint $\mathit{D\text{-}from}(T; \emptyset; IK)$.

for an intrudable symbol f.

We translate this rule into the following rule

$$\frac{(\textit{D-from}(\{t_1,\ldots,t_n\}\cup T;\mathit{OIK};\mathit{NIK})\wedge C)\tau,\sigma\tau}{\textit{D-from}(\{t\}\cup T;\mathit{OIK};\mathit{NIK})\wedge C,\sigma}\ \mathit{DG}\ ((n>0\vee T\neq\emptyset)\wedge\langle\!|\Phi|\!\rangle)\ .$$

This translation can be understood as follows. Consider a well-formed constraint store (C,σ) and a generation rule r for *from* constraints that is applicable to $(\lceil C\rceil,\sigma)$. The translation of r yields a generation rule for *D-from* constraints that is applicable and functions analogously to the original generation rule operating over *from* constraints: its application replaces the term t to be generated with its subterms $t_1,\ldots,t_n$. Note that applying the $\lceil\cdot\rceil$ function to all *D-from* constraints in the DG rule yields the original generation rule for *D-from* constraints with a more restricted side condition. Note also that the rule excludes the case when the term t to be generated is a constant and there are no other terms to be generated. By the side condition Φ, this constant must be intrudable (e.g., the intruder name i), so, in this case, the reduction fails, since the intruder cannot use anything from the new intruder knowledge to generate the term. Hence, this translation represents precisely a situation where constraint differentiation removes cases of the reduction.

As a concrete example, recall that the rule for generating symmetric encryptions is a generation rule with

$$\Phi\equiv t=\{\!|t_2|\!\}_{t_1}\wedge n=2\wedge\tau=id\ .$$

Translating this rule for constraint differentiation yields the DG rule with the side condition

$$(n>0\vee T\neq\emptyset)\wedge\langle\!|\Phi|\!\rangle\ .$$

Since Φ does not refer to IK or C, we have $\langle\!|\Phi|\!\rangle=\Phi$, and since $n=2$, we can simplify this entire side condition to just Φ.

3.2.2 Unification Rules

Consider a unification rule, which has the form

$$\frac{(\textit{from}(T;\{s\}\cup \mathit{IK})\wedge C)\tau,\sigma\tau}{\textit{from}(\{t\}\cup T;\{s\}\cup \mathit{IK})\wedge C,\sigma}\ U\ (\Phi)\,,$$

where the side condition Φ implies that $s\tau\approx t\tau$.

This rule is translated into the following two rules for *D-from* constraints:

$$\frac{(\textit{D-from}(T;\{s\}\cup \mathit{OIK};\mathit{NIK})\wedge C)\tau,\sigma\tau}{\textit{D-from}(\{t\}\cup T;\{s\}\cup \mathit{OIK};\mathit{NIK})\wedge C,\sigma}\ \mathit{DU}_{\mathit{OIK}}\ (T\neq\emptyset\wedge\langle\!|\Phi|\!\rangle)\,,$$

$$\frac{(\textit{from}(T;\mathit{OIK}\cup\{s\}\cup \mathit{NIK})\wedge C)\tau,\sigma\tau}{\textit{D-from}(\{t\}\cup T;\mathit{OIK};\{s\}\cup \mathit{NIK})\wedge C,\sigma}\ \mathit{DU}_{\mathit{NIK}}\ (\langle\!|\Phi|\!\rangle)\,.$$

This translation distinguishes two cases. First, DU_{OIK} states that the term t to be generated can be unified with a term s in the old intruder knowledge OIK of the constraint, and there is a non-empty set T of remaining terms to be generated. In this case, we proceed as expected, removing the term t from the terms to be generated. Thus, the new constraint expresses that some new knowledge must be used when generating the other terms T. Note that when $T = \emptyset$, that is, when there are no other terms to be generated, the rule cannot be applied (and thus the reduction fails) since the only term t to be generated can be generated from the old intruder knowledge alone.

The second case is when the term t can be unified with a term s from the new intruder knowledge NIK. Thus, the requirement to use the new intruder knowledge is fulfilled and therefore it is not necessary to use additional terms from the new intruder knowledge when generating the remaining terms T. Hence, the new constraint is a standard *from* constraint for the terms T, where the intruder knowledge contains both the old and the new intruder knowledge.

For instance, the unification rule presented above has the side condition

$$\Phi \equiv t \notin \mathcal{V} \wedge \tau = mgu(t, s) \; .$$

As Φ does not refer to the constraints C or the intruder knowledge IK, $\langle\!|\Phi|\!\rangle = \Phi$ holds in this example.

3.2.3 Analysis Rules

Consider an analysis rule, which has the form

$$\frac{(\mathit{from}(\{k\}; \{m^?\} \cup IK) \wedge \mathit{from}(T; \{m, r\} \cup IK) \wedge C)\tau, \sigma\tau}{\{\mathit{from}(T; \{m\} \cup IK)\} \cup C, \sigma} \; A\,(\Phi)\,,$$

where the side condition Φ implies that $r\tau \in \mathcal{DY}(\{m\tau, k\tau\})$.

We translate this rule into the following two rules:

$$\frac{(\mathit{from}(\{k\}; \{m^?\} \cup OIK \cup NIK) \wedge \mathit{D\text{-}from}(T; \{m\} \cup OIK; \{r\} \cup NIK) \wedge C)\tau, \sigma\tau}{\mathit{D\text{-}from}(T; \{m\} \cup OIK; NIK) \wedge C, \sigma} \; DA_{OIK}\,(\langle\!|\Phi|\!\rangle)\,,$$

$$\frac{(\mathit{from}(\{k\}; \{m^?\} \cup OIK \cup NIK) \wedge \mathit{D\text{-}from}(T; OIK; \{m, r\} \cup NIK) \wedge C)\tau, \sigma\tau}{\mathit{D\text{-}from}(T; OIK; \{m\} \cup NIK) \wedge C, \sigma} \; DA_{NIK}\,(\langle\!|\Phi|\!\rangle)\,.$$

This translation again distinguishes two cases for the *D-from* constraints: the term m to be analyzed is either part of the old or the new intruder knowledge. The two cases are handled in a similar way: the key-term k for decryption can be any term derivable from the old and new intruder knowledge (thus it is a standard *from* constraint) and the result r is always added to the new intruder knowledge.[5]

[5] With the rule DA_{OIK}, we may potentially miss some redundancies that could be exploited: it can happen that a result r is added to the new intruder knowledge that is derivable from the old intruder knowledge alone, namely when we analyze a term m in the old intruder knowledge and the key term k can be generated from OIK alone. As will be discussed in Subsection 3.3, this is not a problem of correctness, but only of efficiency.

For example, recall that the rule for analyzing symmetric encryptions is an analysis rule with

$$\Phi \equiv m = \{\!|r|\!\}_k \wedge \tau = id\ .$$

Again, Φ does not refer to C or IK, thus $\langle\!|\Phi|\!\rangle \equiv \Phi$.

3.2.4 Rule Translation and its Properties

Definition 8. *For a reduction rule r for from constraints, we define the* translation $tr(r)$ of r for *D-from* constraints *as follows:*

$$\begin{aligned} tr(G(\Phi)) &= \{DG((n > 0 \vee T \neq \emptyset) \wedge \langle\!|\Phi|\!\rangle)\} \\ tr(U(\Phi)) &= \{DU_{OIK}(T \neq \emptyset \wedge \langle\!|\Phi|\!\rangle), DU_{NIK}(\langle\!|\Phi|\!\rangle)\} \\ tr(A(\Phi)) &= \{DA_{OIK}(\langle\!|\Phi|\!\rangle), DA_{NIK}(\langle\!|\Phi|\!\rangle)\} \end{aligned}$$

By extension, $tr(R) = \cup_{r \in R}\, tr(r)$ for a set of rules R.

The derivation relation $\vdash_R^D$ is defined as $\vdash_{R \cup tr(R)}$. The set of pairs of simple D-from constraints and substitutions that can be derived from (C, id) is

$$\textit{D-Red}_R(C, \sigma) = \{(C', \sigma') \mid ((C, \sigma) \vdash_R^D (C', \sigma')) \wedge \textit{simple}(C')\}\,.$$

We write $\textit{D-Red}_R(C)$ as a shorthand for $\textit{D-Red}_R(C, id)$.

Some remarks on the properties of the rule translation are in order. First, under $\lceil\cdot\rceil$, the translated rules are entailed by the original rules in the following sense:

Lemma 1. *Let $r = \dfrac{D, \tau}{C, \sigma}\ \langle\!|\Phi|\!\rangle \wedge \Psi$ and $r' = \dfrac{D', \tau'}{C', \sigma'}\ \Phi$ be rules such that $r \in tr(r')$. Then the rule $\lceil r \rceil = \dfrac{\lceil D \rceil, \tau}{\lceil C \rceil, \sigma}\ \langle\!|\Phi|\!\rangle \wedge \Psi$ is entailed by r'.*

Proof. For all rules, we have that $\lceil C \rceil = C'[IK \mapsto OIK \cup NIK$ and $\lceil D \rceil = D'[IK \mapsto OIK \cup NIK]$ and, moreover, the side conditions are more restrictive in the translated version. Thus, in all cases, the translated rules are entailed by the original rules. □

The second property is that every rule application of the translated rules corresponds to a rule application of the original rules:

Lemma 2. $(C, \sigma) \vdash_R^D (C', \sigma')$ *implies* $(\lceil C \rceil, \sigma) \vdash_R (\lceil C' \rceil, \sigma')$.

Proof. Let $(C, \sigma) \vdash_R^D (C', \sigma')$. Thus $(C, \sigma) \vdash_r (C', \sigma')$ for some rule $r \in R \cup tr(R)$. If $r \in R$, then r does not refer to *D-from* constraints, thus r can be applied in the same way to $(\lceil C \rceil, \sigma)$, yielding $(\lceil C' \rceil, \sigma')$. Otherwise, if $r \in tr(r')$ for some $r' \in R$, then r and r' must have the form as in Lemma 1 and $\lceil r \rceil$ is entailed by r' (in the sense of Lemma 1). It follows that $(\lceil C \rceil, \sigma) \vdash_{\lceil r \rceil} (\lceil C' \rceil, \sigma')$ and thus $(\lceil C \rceil, \sigma) \vdash_{r'} (\lceil C' \rceil, \sigma')$. Since $r' \in R$, the claim follows. □

This shows a central property of derivations with the translated rules: the translated rules admit no more derivations than the original rules. Therefore, if $Red_R(\lceil C \rceil, \sigma)$ terminates and is sound, then $\textit{D-Red}_R(C, \sigma)$ also terminates and is sound modulo $\lceil\cdot\rceil$.

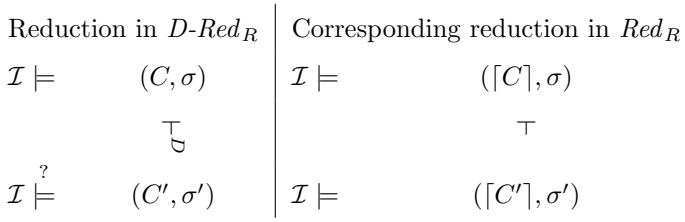

Figure 5: Schema for the completeness proof.

3.3 Properties of *D-Red*$_R$

Roughly speaking, in this section, we show the following. Given a set of rules R such that *Red*$_R$ is correct and terminating for *from* constraints, then *D-Red*$_R$ is also correct and terminating for *D-from* constraints. This is not entirely precise because $C \models\!\!\!=$ *D-Red*$_R(C)$ does *not* hold in general.

Consider for example the *D-from* constraint $C = \textit{D-from}(\{m\}; \mathit{OIK}; \mathit{NIK})$ with $\mathit{OIK} = \{k, \{\!|m|\!\}_k\}$ and $\mathit{NIK} = \{m\}$. Obviously, the intruder can derive the term m from *OIK* alone without using *NIK*, and hence C is unsatisfiable. However, the rule DU_{NIK} is applicable, yielding the simple (and thus satisfiable) constraint $C' = \textit{from}(\emptyset; \mathit{OIK} \cup \mathit{NIK})$; so this derivation is not sound with respect to the semantics of C. However, this derivation is sound with respect to the semantics of $\lceil C \rceil$ and we show below that, in general, *D-Red*$_R$ is sound in this regard, as shown in Lemma 2. This is sufficient, since the entire idea of constraint differentiation is to exploit redundancies in the state space, and interpretations that satisfy $\lceil C \rceil$, but not C, are redundant. Thus *D-Red*$_R$, in general, also returns redundant solutions. However, this is not a problem of the correctness of the approach, only of its efficiency: it may miss some redundancies that could be exploited.

Hence, we show soundness only with respect to the original approach. For the converse, completeness, we have that all solutions of C are contained in *D-Red*$_R(C)$.

Theorem 1. *Let R be a set of reduction rules for from constraints such that Red$_R$ is correct and terminating (with respect to the considered intruder model and notion of well-formed constraint), and let C be a well-formed constraint (consisting of from and D-from constraints). Then D-Red$_R(C)$ terminates and*

$$C \models \textit{D-Red}_R(C) \models \lceil C \rceil \ .$$

Proof. From Lemma 2, it follows that, whenever $(C, \sigma) \vdash^D (C', \sigma')$ is a reduction performed by *D-Red*$_R$, then $(\lceil C \rceil, \sigma) \vdash (\lceil C' \rceil, \sigma')$ is a reduction performed by *Red*$_R$. Thus, *Red*$_R$ has at least as many derivations as *D-Red*$_R$, since the only reductions of *Red*$_R$ that have no counterpart in *D-Red*$_R$ are those excluded by the side conditions of the *D-Red*$_R$ rules. Since *Red*$_R$ is correct and terminating by assumption, *D-Red*$_R$ is terminating and sound with respect to $\lceil \cdot \rceil$.

Proving completeness is more involved and the proof has the following shape (see also Figure 5). We consider an arbitrary well-formed constraint store (C, σ), where C is not simple. Moreover, we assume that this constraint store is satisfiable and consider one fixed satisfying interpretation $\mathcal{I} \models (C, \sigma)$. This implies also that $\mathcal{I} \models (\lceil C \rceil, \sigma)$. Since we have assumed the correctness of Red_R, there is a derivation $(\lceil C \rceil, \sigma) \vdash_{r'} (\lceil C' \rceil, \sigma')$, with $\mathcal{I} \models (\lceil C' \rceil, \sigma')$ for some $r' \in R$. We show that in all cases there is a rule $r \in tr(r)$ such that $(C, \sigma) \vdash_r (C', \sigma')$ and $\mathcal{I} \models (C', \sigma')$ (as indicated by the question mark in the figure). Thus, for every solution $\mathcal{I}$, there is a reduction of $\mathit{D\text{-}Red}_R$ that supports $\mathcal{I}$ (unless we have reached a simple constraint that cannot be further reduced). By induction, we then have $(C, \sigma) \models \mathit{D\text{-}Red}_R(C, \sigma)$, i.e., completeness.

We distinguish three cases, corresponding to the three rule schemata G, U, and A, for the reduction from $(\lceil C \rceil, \sigma)$ to $(\lceil C' \rceil, \sigma')$. In all cases, we assume that the side condition Φ of the respective rule (i.e., instance of G, U, or A) holds for the reduction from $(\lceil C \rceil, \sigma)$ to $(\lceil C' \rceil, \sigma')$. Since $\langle\!|\Phi|\!\rangle$ is the adaption of Φ to *D-from* constraints, $\langle\!|\Phi|\!\rangle$ analogously holds for the reduction from $(\lceil C \rceil, \sigma)$ to $(\lceil C' \rceil, \sigma')$. We therefore need not refer to the concrete instances of Φ and $\langle\!|\Phi|\!\rangle$ in the following, and rather consider only the general condition required by the definition of the rule schemata, e.g., $s\tau \approx t\tau$ for the U rule schema.

$\boxed{G}$ For a reduction using a G rule, we must show that DG is applicable:

$\mathcal{I} \models \mathit{D\text{-}from}(\{t\} \cup T; \mathit{OIK}; \mathit{NIK}) \wedge C, \sigma$ and
$\mathcal{I} \models \mathit{from}(\{t\} \cup T; \mathit{OIK} \cup \mathit{NIK}) \wedge C, \sigma$ and
$\mathcal{I} \models (\mathit{from}(\{t_1, \ldots, t_n\} \cup T; \mathit{OIK} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau$ and
$t\tau \approx f(t_1, \ldots, t_n)\tau$ and $\mathit{intrudable}(f)$
implies
$(T \neq \emptyset \vee n > 0)$ and $\mathcal{I} \models (\mathit{D\text{-}from}(\{t_1, \ldots, t_n\} \cup T; \mathit{OIK}; \mathit{NIK}) \wedge C)\tau, \sigma\tau$

We can exclude the case that both $T = \emptyset$ and $n = 0$, as otherwise $t\tau$ is an intrudable constant and thus $\mathcal{I} \not\models \mathit{D\text{-}from}(\{t\} \cup T; \mathit{OIK}; \mathit{NIK})$, contradicting the first assumption.

Suppose the second conjunct of the conclusion is false, i.e.

$$\mathcal{I} \not\models (\mathit{D\text{-}from}(\{t_1, \ldots, t_n\} \cup T; \mathit{OIK}; \mathit{NIK}) \wedge C)\tau, \sigma\tau \ .$$

Then either t_i and T cannot be generated from $\mathit{OIK} \cup \mathit{NIK}$ or they can be generated from OIK alone. The first possibility contradicts the third assumption and thus we have

$$\mathcal{I} \models \mathit{from}(\{t_1, \ldots, t_n\} \cup T; \mathit{OIK})\tau \ .$$

But since $t\tau \approx f(t_1, \ldots, t_n)\tau$ and $\mathit{intrudable}(f)$, it holds that $\mathcal{I} \models \tau$ and thus $t\tau^{\mathcal{I}} \in \mathcal{DY}(\{t_1, \ldots, t_n\}^{\mathcal{I}})$. We then also have

$$\mathcal{I} \models \mathit{from}(\{t\} \cup T; \mathit{OIK})\tau$$

and thus

$$\mathcal{I} \not\models \mathit{D\text{-}from}(\{t\} \cup T; \mathit{OIK}; \mathit{NIK})\tau \ ,$$

contradicting the first assumption.

$\boxed{U}$ For a reduction using a U rule, we must show that DU_{OIK} or DU_{NIK} is applicable. We thus distinguish two cases depending on whether a term in the OIK or the NIK part is unified.

The $\underline{NIK \text{ case}}$ is trivial as the conclusion is part of the assumptions:

$$\begin{array}{l}\mathcal{I} \models \textit{D-from}(\{t\} \cup T; \mathit{OIK}; \{s\} \cup \mathit{NIK}) \wedge C, \sigma \text{ and} \\ \mathcal{I} \models \textit{from}(\{t\} \cup T; \mathit{OIK} \cup \{s\} \cup \mathit{NIK}) \wedge C, \sigma \text{ and} \\ \mathcal{I} \models (\textit{from}(T; \mathit{OIK} \cup \{s\} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau \text{ and} \\ t\tau \approx s\tau \\ \text{implies} \\ \mathcal{I} \models (\textit{from}(T; \mathit{OIK} \cup \{s\} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau\end{array}$$

For the $\underline{OIK \text{ case}}$ we show:

$$\begin{array}{l}\mathcal{I} \models \textit{D-from}(\{t\} \cup T; \{s\} \cup \mathit{OIK}; \mathit{NIK}) \wedge C, \sigma \text{ and} \\ \mathcal{I} \models \textit{from}(\{t\} \cup T; \{s\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge C, \sigma \text{ and} \\ \mathcal{I} \models (\textit{from}(T; \{s\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau \text{ and} \\ t\tau \approx s\tau \wedge T \neq \emptyset \\ \text{implies} \\ T \neq \emptyset \wedge \mathcal{I} \models (\textit{D-from}(T; \{s\} \cup \mathit{OIK}; \mathit{NIK}) \wedge C)\tau, \sigma\tau\,.\end{array}$$

Suppose that $T = \emptyset$. Then $\{t\} \cup T$ can be generated from $\{s\} \cup \mathit{OIK}$ alone, contradicting the first assumption.

Suppose $\mathcal{I} \not\models (\textit{D-from}(T; \{s\} \cup \mathit{OIK}; \mathit{NIK}) \wedge C)\tau, \sigma\tau$. Then, by the assumptions, it can have failed only due to the new *D-from* constraint, i.e.,

$$\mathcal{I} \models \textit{from}(T; \{s\} \cup \mathit{OIK})\,.$$

Since $t\tau \approx s\tau$ and $\mathcal{I} \models \tau$, we have also

$$\mathcal{I} \models \textit{from}(\{t\} \cup T; \{s\} \cup \mathit{OIK})$$

and therefore

$$\mathcal{I} \not\models \textit{D-from}(\{t\} \cup T; \{s\} \cup \mathit{OIK}; \mathit{NIK})\,,$$

which contradicts the assumption.

$\boxed{A}$ For a reduction using an A rule, we must show that DA_{OIK} or DA_{NIK} is applicable. We thus distinguish whether a term in OIK or NIK is analyzed.

In the case that a term in OIK is analyzed, we show the following:

$$\begin{array}{l}\mathcal{I} \models \textit{D-from}(T; \{m\} \cup \mathit{OIK}; \mathit{NIK}) \wedge C, \sigma \text{ and} \\ \mathcal{I} \models \textit{from}(T; \{m\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge C, \sigma \text{ and} \\ \mathcal{I} \models (\textit{from}(\{k\}; \{m^?\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge \textit{from}(T; \{m, r\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau \text{ and} \\ r\tau \in \mathcal{DY}(\{m\tau, k\tau\}) \\ \text{implies} \\ \mathcal{I} \models (\textit{from}(\{k\}; \{m^?\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge \textit{D-from}(T; \{m\} \cup \mathit{OIK}; \{r\} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau\end{array}$$

Suppose the conclusion is false. Then it follows that either

- $\mathcal{I} \not\models \mathit{from}(\{k\}; \{m^?\} \cup \mathit{OIK} \cup \mathit{NIK})\tau$, contradicting the second assumption, or
- $\mathcal{I} \not\models \mathit{D\text{-}from}(T; \{m\} \cup \mathit{OIK}; \{r\} \cup \mathit{NIK})\tau$.

Moreover, since $\mathcal{I} \models \mathit{from}(T; \{m, r\} \cup \mathit{OIK} \cup \mathit{NIK})\tau$ by the second assumption, it follows that

$$\mathcal{I} \models \mathit{from}(T; \{m\} \cup \mathit{OIK})\tau \ ,$$

which implies

$$\mathcal{I} \not\models \mathit{D\text{-}from}(T; \{m\} \cup \mathit{OIK}; \mathit{NIK})\tau$$

and thus, since $\mathcal{I} \models \tau$,

$$\mathcal{I} \not\models \mathit{D\text{-}from}(T; \{m\} \cup \mathit{OIK}; \mathit{NIK}) \ ,$$

which contradicts the first assumption.

In the case that a term in *NIK* is analyzed, we show the following:

$\mathcal{I} \models \mathit{D\text{-}from}(T; \mathit{OIK}; \{m\} \cup \mathit{NIK}) \wedge C, \sigma$ and
$\mathcal{I} \models \mathit{from}(T; \{m\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge C, \sigma$ and
$\mathcal{I} \models (\mathit{from}(\{k\}; \{m^?\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge \mathit{from}(T; \{m, r\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau$ and
$r\tau \in \mathcal{DY}(\{m\tau, k\tau\})$
implies
$\mathcal{I} \models (\mathit{from}(\{k\}; \{m^?\} \cup \mathit{OIK} \cup \mathit{NIK}) \wedge \mathit{D\text{-}from}(T; \mathit{OIK}; \{m, r\} \cup \mathit{NIK}) \wedge C)\tau, \sigma\tau$

Again, assuming the conclusion does not hold gives us

$$\mathcal{I} \models \mathit{from}(T; \mathit{OIK})\tau$$

and thus

$$\mathcal{I} \not\models \mathit{D\text{-}from}(T; \mathit{OIK}; \{m\} \cup \mathit{NIK})\tau \ .$$

As $\mathcal{I} \models \tau$, this again contradicts the first assumption.

This concludes the completeness proof: for every *D-from* constraint store and substitution (C, σ) that has a solution $\mathcal{I}$, we can find a reduction (using $\mathit{Red}_R(\lceil\cdot\rceil)$) that still supports $\mathcal{I}$. Thus, if Red_R is complete for *from* constraints, then so is $\mathit{D\text{-}Red}_R$ for *D-from* constraints. □

This theorem tell us that $\mathit{D\text{-}Red}_R$ is correct and terminating (under the assumption that this already holds for Red_R on well-formed *from* constraints). Thus, we can apply $\mathit{D\text{-}Red}_R$ to reduce the search space without excluding attacks or introducing new ones. To formalize this reduction of the search space, we integrate constraint differentiation into the search tree induced by the symbolic transition system.

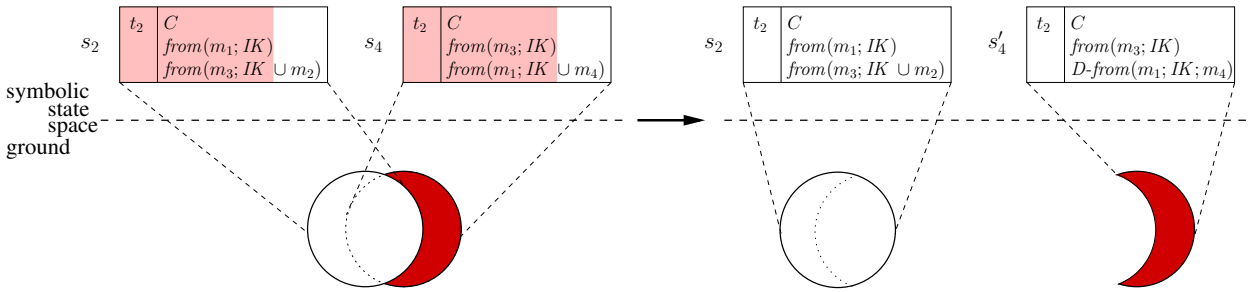

Figure 6: Constraint differentiation at work.

3.4 Integrating Constraint Differentiation with Symbolic Transition Systems

Consider again the tree in Figure 4, which characterizes when constraint differentiation is applicable: we exploit the fact that the two symbolic states s_2 and s_4 of Figure 4 represent overlapping sets of ground states as shown by the shaded parts in s_2 and s_4.

Figure 6 merges parts of Figure 3 and Figure 4 to illustrate how constraint differentiation works: we pick one, say s_4, of the overlapping states s_2 and s_4 in Figure 4 (where $t_2 = t_4$) and replace the *from* constraint that does not appear in the other constraint with a *D-from* constraint. This yields the transformed state s'_4. That is, we use constraint differentiation to restrict the extension of one of the two symbolic states to those ground states that are not covered by the other (as illustrated by the shaded part in the set of ground states). The following theorem shows that s_2 and s'_4 represent the same ground states as s_2 and s_4.

Theorem 2. *Consider two symbolic states $s_2 = (t_2, C_2)$ and $s_4 = (t_2, C_4)$ with constraints of the form $C_2 = C \wedge from(\{m_1\}; IK) \wedge from(\{m_3\}; IK \cup \{m_2\})$ and $C_4 = C \wedge from(\{m_3\}; IK) \wedge from(\{m_1\}; IK \cup \{m_4\})$, for m_1, m_2, m_3, and m_4 messages and C a constraint. Moreover, let $C'_4 = C \wedge from(\{m_3\}; IK) \wedge D\text{-}from(\{m_1\}; IK; \{m_4\})$ and $s'_4 = (t_2, C'_4)$. Then $[\![s_2]\!] \cup [\![s_4]\!] = [\![s_2]\!] \cup [\![s'_4]\!]$.*

Proof. It suffices to show that $\{C_2, C_4\} =\!\|\!\models \{C_2, C'_4\}$. Since $\lceil C'_4 \rceil = C_4$ and $\lceil C'_4 \rceil =\!| C'_4$, the $=\!|$-direction is trivial. To show $\{C_2, C_4\} \models \{C_2, C'_4\}$, we show that for every interpretation $\mathcal{I}$ with $\mathcal{I} \models C_4$ and $\mathcal{I} \not\models C_2$ it holds that $\mathcal{I} \models C'_4$. So assume $\mathcal{I} \not\models C'_4$. Then, since $\mathcal{I} \models C_4$, it must hold that $\mathcal{I} \models \lfloor C_4 \rfloor$. Hence, $\mathcal{I} \models from(\{m_1\}; IK)$ and, since $\mathcal{I} \models from(\{m_3\}; IK)$, we have $\mathcal{I} \models C_2$, which contradicts the assumption. □

This theorem allows us to transform a search tree by replacing *from* constraints with more restrictive *D-from* constraints without changing the set of represented ground states. If under the more restrictive constraint C'_4 the intruder could not use any new message from his knowledge, then even if C_4 is satisfiable, C'_4 is unsatisfiable (so that the shaded part of the set of ground

states in Figure 6 is also empty), which we can check using *D-Red*$_R$. This is the maximal reduction that can be achieved by constraint differentiation: the node of the state s_4 and its subtree can be completely pruned from the search tree as the intruder could not generate anything "interesting", i.e., nothing that he could not have generated before. Note that we can always consider the symmetric situation: if performing the restriction on s_2 rather than s_4 leads to an unsatisfiable constraint, then we can remove the respective subtree.

When we apply *D-Red*$_R$ to a state that results by replacing *from* constraints with *D-from* constraints, in the best case the constraint conjunction turns out to be unsatisfiable, so the state (and the respective subtree) can be pruned. However, it is also possible that after applying *D-Red*$_R$ there still remain simple *D-from* constraints (i.e., where the T-part is a set of variables). This means that it is not yet determined what the intruder will use here and it is possible that it is some message from *NIK*. Such a *D-from* constraint is nonetheless useful for the reduction, as it constrains the child nodes by excluding certain solutions: the *D-from* constraint prevents all later instantiations of the variable in the T-part if these instantiations do not use some message of the *NIK*-part.

As an example, consider the differentiated constraint

$$C_1 = \textit{D-from}(\{\{|M|\}_k\}; \{\{|m_1|\}_k\}; \{\{|m_2|\}_k\}) \ .$$

There is only one solution for C_1, namely $M = m_2$, while $\lceil C_1 \rceil$ has additionally the solution $M = m_1$. Thus, a symbolic state that contains only the constraint C_1 cannot yet be pruned from the search space. Assume now that during further search (i.e., in a child node) we get the additional constraint

$$C_2 = \textit{from}(h(M); \{\{|m_1|\}_k, \{|m_2|\}_k, h(m_1)\}) \ .$$

Now C_2 has only the solution $M = m_1$. So both C_1 and C_2 are satisfiable individually, and $\lceil C_1 \rceil \wedge C_2$ is also satisfiable with $M = m_1$, but $C_1 \wedge C_2$ is unsatisfiable. Thus we can prune child nodes that contain this constraint, even though the new constraint C_2 is not differentiated. Hence, constraint differentiation may also constrain the successors of differentiated nodes.

3.5 Implementing Constraint Differentiation in OFMC

Theorem 2 describes a sufficient condition for applying constraint differentiation in a symbolic search tree. Namely, it suffices to identify during search a subtree that matches the pattern of states $s_1, \ldots, s_4$ described in the theorem. The question then is how to detect such patterns efficiently during the search for reachable symbolic states. Detecting all pattern occurrences can be time consuming and thereby mitigate the benefits of constraint differentiation in the first place. There is thus a trade-off between the time spent in detecting occurrences of this pattern and the actual reduction achieved.

We now briefly describe how we have dealt with this trade-off when integrating constraint differentiation into our on-the-fly protocol model-checker OFMC [9]. We employ a heuristic that may miss some potential applications of

Theorem 2 in exchange for better run-time performance. In particular, rather than checking every pair of states for such an application, our heuristic performs a simple local check on reachable states. Although the heuristic may fail to prune all those states that can safely be pruned, the correctness of the entire approach is always maintained.

In OFMC, the transition system is given by a set of transition rules. The most common kind of rule, called a *standard rule*, describes how an honest agent in a particular local state can receive a message, send a reply, update its local state, and introduce additional facts into the successor state (these additional facts are, for instance, used to express goal-related information, e.g. that a certain message is supposed to be a secret between a certain group of agents). Every transition induced by a standard rule is thus related to a particular honest agent in one protocol session.

Consider now a symbolic state that allows for two or more transitions related to different executions of the protocol. (These transitions may relate to the same honest agent and may be induced by the same transition rule.) Then, neither transition can disable the other and also the order of the two transitions can only make a difference in the intruder's constraints, exactly as in the pattern described by Theorem 2. It is thus safe to apply the theorem in one of the two resulting states of the pair of transitions.

Note that OFMC also supports a more general form of transition rules than the standard rules described above. One can, for example, specify rules that formalize the relationship between several protocol runs of the same agent. We do not apply constraint differentiation to a pair of transitions where at least one is induced by a non-standard rule because, in this case, the rule form alone does not guarantee that both transitions can be taken in either order and lead to the same resulting states (modulo the intruder constraints). This restriction does not reduce the class of protocols that we can consider but only the amount of redundancies that constraint differentiation can exploit.

4 Experimental Results

Even without constraint differentiation, OFMC is a state-of-the-art tool for finding protocol flaws, as is documented in [9]. It requires, for example, only a few seconds to find attacks against all flawed protocols of the Clark-Jacob library.

OFMC is one of the back-ends of the AVISPA Tool for protocol analysis [6, 2]. Before considering in detail the experiments that we have carried out, note that the specification language of AVISPA (and thus also OFMC's) requires one to specify a concrete scenario under which a protocol should be checked. This scenario is a set of instantiations of the protocol's roles with concrete agent names. OFMC additionally allows for *symbolic sessions* [9], which is where the scenario is described using variables (of type agent), representing all possible concrete scenarios for a given number of sessions.

To illustrate the effects of constraint differentiation, in Figure 7 we show

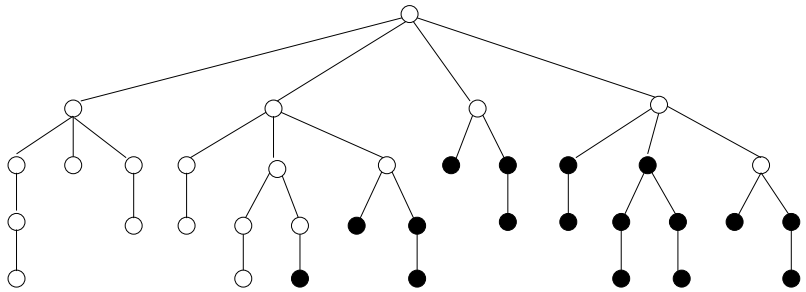

Figure 7: The effect of constraint differentiation on the concrete search tree of NSL with one symbolic session.

the search tree of OFMC for the Needham-Schroeder-Lowe Public Key protocol NSL [27] in the case of one symbolic session. Even in such a simple case, constraint differentiation has a significant effect on the search tree: of the 36 symbolic nodes, the 17 displayed in black can be immediately pruned after constraint differentiation.

In general, the integration of constraint differentiation into OFMC has substantially improved its performance and scope. With constraint differentiation, OFMC scales much better when used to find flaws (falsification) in industrial-strength protocols. Moreover, constraint differentiation substantially improves the effectiveness of OFMC as a verification tool. Note, in this regard, that verifying correct protocols is generally much more complex than falsifying flawed protocols since flaw detection terminates as soon as an error is detected, while verification requires that the entire search space is examined.

As concrete examples, we have applied OFMC to a large testsuite of 70 real-world protocols, which are listed in Figure 8. The first 66 are the protocols of the *AVISPA protocol library* [5] (we follow the naming conventions used in the AVISPA library, where further information on the protocols can be found). The additional 4 examples come from OFMC's own testsuite and formalize protocols with algebraically-defined operators. In particular, Crypto API is a partial model of a hardware security module [36]; Diffie-Hellman secure channels uses device pairing to secure the Diffie-Hellman exchange [13]; Minimal Diffie-Hellman uses an algebraic theory that is simplified, as far as possible, while retaining protocol executability; SRP (with implicit decryption) is a more realistic model of the Secure Remote Passwords protocol [37] than the one contained in AVISPA library.

Figure 8 compares the performance of OFMC on the testsuite with and without constraint differentiation.[6] In 8 cases, the analysis without constraint differentiation exceeds 1 hour CPU time, which we have set as a time-out. Moreover, without constraint differentiation, the total analysis time for all examples requires over 10 hours (due to time-out we do not know the precise value), and

[6]The experiments described here are performed on an IBM Laptop with an Intel T2600 Dual Core Processor (2×2.16 GHz) and 2GB RAM.

Protocol	*Time/s With CD*	*Time/s Without CD*
ISO1	0.01	0.01
ISO2	0.03	0.12
ISO3	0.01	0.01
ISO4	0.28	1.37
CHAPv2	0.14	34.23
EKE	0.03	0.06
EKE2	0.03	0.15
SPEKE	1.54	104.17
AAAMobileIP	0.10	3.65
IKEv2-CHILD	0.39	25.31
IKEv2-DS	0.07	0.12
IKEv2-DSx	11.87	910.93
IKEv2-MAC	2.25	34.67
IKEv2-MACx	11.03	829.40
TLS	0.18	1.34
Kerberos-Cross-Realm	4.21	25.32
Kerberos-Forwardable	7.76	117.87
Kerberos-PKINIT	2.17	2.54
Kerberos-Ticket-Cache	0.67	2.09
Kerberos-basic	0.71	1.51
Kerberos-preauth	1.96	2.18
LPD-IMSR	0.01	0.12
LPD-MSR	0.01	0.01
CRAM-MD5	0.14	19.14
PBK	0.18	0.85
PBK-fix	0.07	0.73
PBK-fix-weak-auth	1.76	651.71
DHCP-delayed-auth	0.03	0.18
H.530	0.35	0.40
H.530-fix	6.82	341.32
Lipkey-Spkm-known-initiator	0.15	21.17
Lipkey-Spkm-unknown-initiator	4.20	163.68
ASW	0.43	0.71
ASW-abort	2.14	4.53
CTP-non-predictive-fix	0.03	0.04
FairZG	5.10	*TO*

Protocol	*Time/s With CD*	*Time/s Without CD*
QoS-NSLP	222.31	*TO*
SET-purchase	0.51	0.98
SET-purchase-honest-PG	1.34	16.37
TSIG	0.12	3.85
Geopriv	0.17	29.21
Geopriv selfsignatures	0.04	0.07
Geopriv 2 pseudonyms	0.21	127.79
Geopriv password	33.51	*TO*
Geopriv pervasive	30.50	*TO*
HIP	0.21	3.23
UMTS AKA	0.01	0.01
2pRSA	0.28	70.01
8021x Radius	0.09	5.43
APOP	1.35	2260.53
EAP-AKA	0.17	18.93
EAP-Archie	0.28	104.43
EAP-IKEv2	0.92	238.87
EAP-SIM	1.28	1204.79
EAP-TLS	0.60	32.76
EAP-TTLS-CHAP	0.54	25.15
IKEv2-EAP-Archie	6.78	237.96
PEAP	6.31	*TO*
RADIUS	0.25	2.54
SHARE	0.03	0.04
SIMPLE	385.73	*TO*
SIP	0.81	*TO*
S/KEY	0.37	*TO*
TESLA	0.04	0.17
TSP	0.09	0.92
SSH Transport	15.17	1610.67
Crypto API	0.31	0.31
Diffie-Hellman secure ch.	1.53	13.92
Minimal-Diffie-Hellman	0.01	0.03
SRP (implicit decrypt.)	28.10	269.32
SUM	806.83	> 38379.93
Without TimeOuts	122.19	9579.93

Figure 8: Comparison of OFMC's performance with and without constraint differentiation on the AVISPA protocol library and four additional protocols.

	With CD		*Without CD*	
Depth	*Time*	*Nodes*	*Time*	*Nodes*
1	0.01	4	0.01	4
2	0.01	14	0.01	18
3	0.03	48	0.04	82
4	0.09	162	0.15	370
5	0.31	517	0.68	1612
6	1.12	1532	3.48	6648
7	3.56	4212	16.73	25396
8	10.82	10296	69.68	87052
9	28.48	21797	256.20	257208
10	63.92	36731	800.76	612696
11	106.81	40048	1989.79	1031184
12	127.35	19031	3417.40	879648

Figure 9: Comparison of the search trees with and without constraint differentiation (CD) for the protocol IKEv2 with Digital Signatures with two symbolic sessions.

about 2.5 hours when excluding the examples that time-out. In contrast, with constraint differentiation, the maximum time required for an example is 386 seconds, and the total CPU time for the entire library is below 14 minutes. Thus the time required for all examples with constraint differentiation is several orders of magnitude smaller than the time without constraint differentiation on many individual examples, and more than 40 times smaller for the whole testsuite.

The improvement achieved varies for different protocols. One common reason for this stems from early versus late authentication. Late authentication means that the first messages of the protocol are exchanged without any means of authentication. Since anyone can send such messages, there are usually fewer nodes in upper parts of the search tree that can be removed using constraint differentiation.

Let us now take a detailed look at one of the above problems: the protocol IKEv2 with digital signatures [25]. We specify two symbolic sessions, which covers all concrete scenarios with two sessions (hence, the analysis times differ from those given in Figure 8, where we considered the specification of the protocol without symbolic sessions as given in the AVISPA library). Figure 9 compares both the size of the plies in the search tree and the running time for search up to (and including) the ply when running with and without constraint differentiation. These results can be explained by the fact that constraint differentiation is most effective when the original search space contains many interleavings of parallel sessions. The savings are most dramatic on the deeper plies of the search tree as the number of interleavings grows exponentially in the original

model; since many interleavings are redundant and constraint differentiation can exploit this redundancy, the number of nodes does not necessarily grow exponentially with the depth of the tree. Hence, the difference between an exponential growth without constraint differentiation and an often sub-exponential growth with constraint differentiation leads to more dramatic savings the deeper the tree is searched.

In summary, constraint differentiation is an effective technique for substantially reducing the size of the search space that must be considered. By employing constraint differentiation, OFMC scales significantly better with the complexity of the verification problem. This extends the scope of OFMC and in some cases enables the analysis of problems that were previously out of the scope of OFMC and other tools.

5 Conclusions

Constraint differentiation effectively integrates the lazy intruder with ideas from partial-order reduction. We have proved that this integration does not change the set of represented ground states and hence is correct in an appropriate technical sense. We have validated our approach experimentally, using the model-checker OFMC. Our experiments show that constraint differentiation leads to dramatic reductions in the size of the state space that must be searched and thereby substantially improves OFMC's performance and scope.

Although our practical experience with constraint differentiation is only with OFMC, constraint differentiation is a general technique and its implementation is straightforward. Thus it should also be possible to integrate it into other tools to obtain similar speed ups. This remains, however, as future work.

Constraint differentiation, as we have presented it, manipulates constraints formalizing terms that must be generated by the Dolev-Yao intruder. Hence, the technique as developed is specialized to security protocols. Note though that the general idea of reducing the number of interleavings searched by manipulating constraints is found in other model-checking approaches involving constraints, such as for timed or hybrid systems. For example, [31, 10] present a method, inspired by partial-order reduction for timed systems. System states consist of control states plus clock constraints and the passage of time makes it difficult to directly exploit partial-order reduction. They show how different interleavings can be identified by disjoining their constraints. The strategy here is different than ours (merging transitions together by unioning constraints, instead of keeping them separate and differentiating constraints) but the ideas are similarly motivated. Similar ideas are found, for example, in the work of [28]. Investigating the application of such reduction approaches to security protocol analysis, and symmetrically the application of constraint differentiation beyond the domain of security protocols, also remains as future work.

Acknowledgments

The work presented in this paper was partially supported by the FP7-ICT-2007-1 Project no. 2164 71, "AVANTSSAR: Automated Validation of Trust and Security of Service-oriented Architectures" (`www.avantssar.eu`). We thank Benedikt Schmidt, Matthias Schmalz, and Cas Cremers for useful comments on a draft of this paper.

References

[1] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In C. Palamidessi, editor, *Proceedings of Concur'00*, LNCS 1877, pages 380–394. Springer, 2002.

[2] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proceedings of CAV'05*. Springer, 2005.

[3] A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Proceedings of FORTE 2002*, LNCS 2529, pages 210–225. Springer, 2002.

[4] A. Armando and L. Compagna. SAT-based Model-Checking for Security Protocols Analysis. *International Journal of Information Security*, 6(1):3–32, 2007.

[5] The AVISPA library. `http://www.avispa-project.org/library`.

[6] The AVISPA tool for security protocol analysis. `http://www.avispa-project.org`.

[7] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[8] D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. In V. Atluri and P. Liu, editors, *Proceedings of CCS'03*, pages 335–344. ACM Press, 2003.

[9] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[10] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In D. Sangiorgi and R. de Simone, editors, *Proceedings of CONCUR'98*, LNCS 1466, pages 485–500. Springer, 1998.

[11] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.

[12] M. Boreale and M. G. Buscemi. A framework for the analysis of security protocols. In *Proceedings of CONCUR 2002*, LNCS 2421, pages 483–498. Springer, 2002.

[13] M. Cagalj, S. Capkun, and J.-P. Hubaux. Key agreement in peer-to-peer wireless networks. *Proceedings of the IEEE (Special Issue on Cryptography and Security)*, 94(2), 2006.

[14] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In *Proceedings of LICS'03*, pages 261–270. IEEE Computer Society Press, 2003.

[15] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *Proceedings of FST TCS'03*, LNCS 2914, pages 124–135. Springer, 2003.

[16] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer, 2002.

[17] S. Clarke, E. Jha and W. Marrero. Partial order reductions for security protocol verification. In *Proceedings of TACAS'00*, LNCS 1785, pages 503–518, 2000.

[18] R. Corin and S. Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proceedings of SAS 2002*, LNCS 2477, pages 326–341. Springer, 2002.

[19] C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proceedings of CAV'08*, LNCS 5123. Springer, 2008.

[20] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[21] B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[22] M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proceedings of CSFW'01*. IEEE Computer Society Press, 2001.

[23] W. Fokkink, M. T. Dashti, and A. Wijs. Partial Order Reduction for Branching Security Protocols. In *Proceedings of WITS'07*, 2007.

[24] A. Huima. Efficient infinite-state analysis of security protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[25] C. Kaufman. RFC 4306: Internet Key Exchange (IKEv2) Protocol, Dec. 2005.

[26] C. Kirchner and H. Kirchner. Rewriting, solving, proving. A preliminary version of a book available at `http://www.loria.fr/~ckirchne/rewriting.html`.

[27] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[28] D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theoretical Computer Science*, 345(1):27–59, 2005.

[29] C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[30] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.

[31] M. Minea. Partial order reduction for model checking of timed automata. In J. C. M. Baeten and S. Mauw, editors, *Proceedings of CONCUR'99*, LNCS 1664, pages 431–446. Springer, 1999.

[32] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 141–153, 1997.

[33] S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols*. PhD thesis, ETH Zurich, Switzerland, 2007.

[34] D. Peled. Ten Years of Partial Order Reduction. In *Proceedings of CAV 1998*, LNCS 1427, pages 17–28. Springer, 1998.

[35] D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

[36] G. Steel. Deduction with XOR Constraints in Security API Modelling. In *Proceedings of CADE 20*, LNAI 3632. Springer, 2005.

[37] T. Wu. The Secure Remote Password Protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.

A Concrete Lazy Intruders

To illustrate the generality of our abstract formalization, we show how several representative examples of concrete lazy intruder approaches can be recast in our formalization, i.e., as instances of the reduction rule schemata that we have introduced in Subsection 2.4. Our intent here is not to define formal translations, but rather to show the main ideas behind such translation. As shown in Theorem 1, given a set of reduction rules implementing a correct and terminating approach, the integration of constraint differentiation preserves these properties. We begin with our own previous work.

A.1 Basin, Mödersheim, and Viganò [9]

As this has been our running example, we have discussed many of the rules already. Here we show how to represent the analysis of pairs in our formalism, as this is the only encoding example that is substantially different from the previous ones.

The analysis rule for pairing from [9] is

$$\frac{\mathit{from}(T; \{m_1, m_2, \langle m_1, m_2\rangle\} \cup IK) \wedge C, \sigma}{\mathit{from}(T; \{\langle m_1, m_2\rangle\} \cup IK) \wedge C, \sigma} \; A^L_{\text{pair}}\left(\{m_1, m_2\} \setminus IK \neq \emptyset\right).$$

In this case, we map A^L_{pair} to two instances of $A\,(\Phi)$, where the side conditions Φ are:

$$\Phi \equiv m = \langle r, r'\rangle \wedge k = i \wedge r \notin IK$$

and

$$\Phi \equiv m = \langle r', r\rangle \wedge k = i \wedge r \notin IK\ .$$

Here, m, r, and k are variables from the rule A and r' is a place holder for the other part of the pair being analyzed. We have also used the name of the intruder i as the "decryption key" that the intruder needs to generate. Since $\mathit{intrudable}(i)$, the intruder can always decompose pairs into their components. We have two rules, one for each of the projections.

A.2 Chevalier and Vigneron [16]

Here we give a representative example of how one of the lazy intruder rules of [16] can be cast in our setting. In particular, they have a rule of the form (we have renamed variables for clarity)

$$T, \textsc{comp}(t) \text{ from } \textsc{know}(s \cup IK); C \rightarrow T\tau \text{ from } \textsc{know}((s \cup IK)\tau); C\tau\ ,$$

where $\tau = \mathit{mgu}(t, s)$ and $t \notin \mathcal{V}$. This differs from our unification rule schema only syntactically (and in the lack of explicitly recording the substitution). We can translate this rule to our setting as

$$\frac{\mathit{from}(T; \{s\} \cup IK) \wedge C; \sigma\tau}{\mathit{from}(\{t\} \cup T; \{s\} \cup IK) \wedge C; \sigma}\ \tau = \mathit{mgu}(s, t), t \notin \mathcal{V}\,,$$

which is an instance of the U rule.

With a similar syntactic transformation, we can also recast the other rules as instances of the G and A schemata, where we have to apply the same transformations for the analysis of a pair as in the case of our lazy intruder approach.

A.3 Extensions to Algebraic Properties

[15, 14] have presented extensions of the lazy intruder approach of [16] to handle algebraic properties. They show that it is sufficient for constraint satisfiability to non-deterministically choose substitutions of a bounded size and check constraint satisfaction under these substitutions, modulo the algebraic properties. We can directly express such a non-deterministic algorithm with our rules. The main rule in our translation is

$$\frac{(\mathit{from}(\{t_1,\ldots,t_n\}\cup T; IK)\wedge C)\tau;\sigma\tau}{\mathit{from}(\{t\}\cup T; IK)\wedge C;\sigma}\ \Phi\,,$$

where

$$\Phi \equiv t\notin\mathcal{V}\wedge\tau\in \mathit{Subst}(k)\wedge t\tau\approx f(t_1,\ldots,t_n)\tau\wedge \mathit{intrudable}(f)\,.$$

Here $k\in\mathbb{N}$ is an additional parameter of the constraint reduction procedure. It is a function of the size of the given constraint, the number of sessions considered, and the size of the protocol. $\mathit{Subst}(k)$ is the set of all substitutions where terms are bounded to size k (and the given set of variables used in the constraint store). As before, $\approx$ is the congruence relation of the algebraic theory under consideration. Note that this approach does not find an equivalent constraint but is focused on satisfiability. Thus the constraint reduction will lead to at least one simple solution iff there exists a solution.

A.4 Millen and Shmatikov [30]

In [30], Millen and Shmatikov present their lazy intruder as a collection of rules, which differ from our schemata only in two main points. First, they employ an ordering on the constraints and only allow for reduction along this order. This requires some coding in the translation: we must label constraints with ordering information, and then check that the reduction is applied to the first non-simple constraint according to that order.

For instance, their rule (which is read top-down):

$$\frac{C_<, s:IK, C_>;\sigma}{\tau C_<, \tau C_>;\tau\cup\sigma}\ (un)_{\text{where } \tau=mgu(s,t), t\in IK}$$

is then translated into our approach as follows:

$$\frac{(\mathit{from}^l(T\,;\,IK)\wedge C)\tau;\sigma\tau}{\mathit{from}^l(\{t\}\cup T\,;\,\{s\}\cup IK)\wedge C;\sigma}\ (\Phi)$$

where

$$\begin{aligned}\Phi \equiv \quad & T = \emptyset \wedge t \notin \mathcal{V} \wedge \tau = mgu(s,t) \\ & \wedge \forall l', T', IK', C'.\, (C = (\mathit{from}^{l'}(T'\,;\, IK') \wedge C') \ \wedge l' < l) \implies T' \subseteq \mathcal{V})\,.\end{aligned}$$

Note that [30] use constraints with only a single term in the term-part and thus $T = \emptyset$ in our rule.

A second obstacle is the variable elimination rule of [30] that we cannot map to our rule schemata. The variable elimination rule removes variables from the intruder knowledge of constraints to ensure that we do not attempt to decrypt the respective term. In our translation, we simply leave the variables in the intruder knowledge and add as a side-condition of every analysis rule that the term to be analyzed is not a variable.

With these two points in mind, the translation of the other rules is straightforward.