A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties

Santiago Escobar [a,c,*], Catherine Meadows [b], José Meseguer [c]

[a] *Universidad Politécnica de Valencia, Valencia, Spain*

[b] *Naval Research Laboratory, Washington, DC, USA*

[c] *University of Illinois at Urbana-Champaign, Urbana, IL, USA*

Abstract

The NRL Protocol Analyzer (NPA) is a tool for the formal specification and analysis of cryptographic protocols that has been used with great effect on a number of complex real-life protocols. One of the most interesting of its features is that it can be used to reason about security in face of attempted attacks on low-level algebraic properties of the functions used in a protocol. Indeed, it has been used successfully to either reproduce or discover a number of such attacks. In this paper we give for the first time a precise formal specification of the main features of the NPA inference system: its grammar-based techniques for invariant generation and its backwards reachability analysis method. This formal specification is given within the well-known rewriting framework so that the inference system is specified as a set of rewrite rules modulo an equational theory describing the behavior of the cryptographic algorithms involved. We then use this formalization to prove some important meta-logical properties about the NPA inference system, including the soundness and completeness of the search algorithm and soundness of the grammar generation algorithm. The formalization and soundness and completeness theorems not only provide also a better understanding of the NPA as it currently operates, but provide a modular basis which can be used as a starting point for increasing the types of equational theories it can handle.

* Corresponding author.

Email addresses: sescobar@dsic.upv.es (Santiago Escobar), meadows@itd.nrl.navy.mil (Catherine Meadows), meseguer@cs.uiuc.edu (José Meseguer).

Preprint submitted to Theoretical Computer Science

1 Introduction

The NRL Protocol Analyzer (NPA) (Meadows, 1996c) is a tool for the formal specification and analysis of cryptographic protocols that has been used with great effect on a number of complex real-life protocols. One of the most interesting of its features is that it can be used, not only to prove or disprove authentication and secrecy properties using the standard Dolev-Yao model (Dolev and Yao, 1983), but also to reason about security in face of attempted attacks on low-level algebraic properties of the functions used in a protocol. Indeed, it has been used successfully to either reproduce or discover a number of such attacks, ranging from the discovery of an authentication attack based on the cancellation properties of encryption and decryption (Meadows, 1992), to the reproduction of Bellovin's attack on a version of the Encapsulating Security Protocol that used cipher block chaining to the discovery of a sophisticated type confusion attack (Stubblebine and Meadows, 2000; Meadows et al., 2004) that resulted in the redesign of a draft for a protocol standard.

NPA's ability to reason well about these low-level functionalities is its combination of symbolic reachability analysis using narrowing, together with its techniques for reducing the size of the search space. On one hand, unification modulo algebraic properties (e.g., encryption and decryption, concatenation and deconcatenation) as narrowing using a finite convergent set of rewrite rules (Baader and Snyder, 2001) allows the tool to represent behavior which is not captured by the usual Dolev-Yao free algebra model. On the other hand, techniques for reducing the size of the search space by using inductively defined co-invariants [1] describing states unreachable to an intruder allows us to start with an infinite search space, and reduce it in many cases to a finite one, thus freeing us from the requirement to put any a priori limits on the number of sessions.

The NPA's use of inductive co-invariants which are defined using grammars similar to tree grammars, has been shown (Meadows, 2000) to be related to structures used in other formalisms, such as strand space ideals (Fabrega et al., 1999) and rank functions (Heather and Schneider, 2005). Indeed, we believe that many of the techniques that have been developed for grammar generation for the NPA could be applied with profit to other systems and some other NPA search space reduction techniques, such as its use of a form of partial order reduction (Peled, 1998), might also prove useful for other systems. Moreover, the NPA's reliance on rewriting and narrowing (TeReSe, 2003; Meseguer, 1992) suggests that it could by profitably compared with other protocol analysis systems that rely on rewriting. However, the adoption

[1] We call here inductively defined sets of unreachable states *co-invariants*, since they are in a sense dual to invariants, which are reachability-closed sets of states.

of NPA language generation techniques has been hampered by the fact that up to now the techniques have lacked an independent formal specification and model, and instead were closely intertwined with other NPA features.

One key contribution of this paper is to rectify this problem by giving for the first time a precise formal specification of the main features of the NPA inference system: its backwards reachability analysis method and its grammar-based techniques for co-invariant generation; both implemented in Maude. We will use the word Maude-NPA to refer to both this formal specification of the original NPA and its Maude implementation. Maude-NPA is given within the well-known rewriting framework so that the inference system is specified as a set of rewrite rules modulo an equational theory describing the behavior of the cryptographic functions involved. A second key contribution of this work is to use the rewriting formalism of the Maude-NPA inference system as a model to prove *meta-logical properties* about such an inference system. Specifically, we prove the *soundness and completeness* of its search algorithm, so that the tool will discover an attack of the type specified by the user if and only if such an attack exists at the level of abstraction supported by the model. We also prove the *unreachability* of the states characterized by grammars, thus showing that the drastic state space reductions afforded by such grammars do not compromise the completeness of the search algorithm. Finally, we have implemented the reachability analysis and language generation techniques in the Maude rewriting logic language, and have used it to generate the formal tree languages used in this paper.

Besides the above-mentioned related work on inductive invariants (Fabrega et al., 1999; Heather and Schneider, 2005), our Maude-NPA rewriting-based formalization facilitates the comparison of NPA with other narrowing-based protocol analysis methods and tools. We do not give here precise comparisons, nor do we try to be exhaustive, but mention some representative approaches. Some of the narrowing-based mechanisms that we discuss have similarities with those used in the OFMC symbolic model checker tool for protocol analysis (Basin et al., 2005). Our work is also related to recent studies and decision procedures to find attacks that may use knowledge of algebraic properties of the underlying cryptographic functions (Comon-Lundh and Shmatikov, 2003; Millen and Shmatikov, 2001; Chevalier et al., 2003a,b). Indeed, as pointed out in (Meseguer and Thati, 2004), narrowing can be viewed as a general inference mechanism unifying many of those analyses. Our rewriting semantics makes clear that the Maude-NPA tool —and its planned extension to support many other analysis of attacks that use knowledge of the algebraic properties of the underlying cryptographic functions— occupies a middle ground between unrestricted (but semi-decidable) narrowing analysis and the decidable, but necessarily restricted, cases for which decision procedures such as those cited above exist.

This work should also be viewed as a first step towards reaching a number of longer-term goals. We are currently working on extending the Maude-NPA's inference system to support the analysis of cryptographic protocols under a wider range of algebraic properties than it currently is capable of, with the ultimate plan of building a next-generation rewriting-based analysis tool that takes into account the algebraic properties of the underlying cryptographic functions. A precise specification of the semantics of the Maude-NPA's inference system in terms of rewrite rules will allow us to cleanly separate out reasoning modulo algebraic properties of the cryptographic algorithms used by the system from the rest of the inference mechanism, paving the way for the insertion of new algebraic properties in a modular fashion. Moreover, we expect that the current formal specification and the proofs of the meta-logical properties will also provide a good basis for specifying the inference system of, and prove meta-logical properties for, this more general next-generation tool.

We start with an overview of the NPA in Section 2 and some preliminaries in Section 3. We introduce the notation used for describing protocols in Section 4. In Section 5, we show how the reachability analysis is performed using grammars to cut down the search space. Then, in Section 6, we first informally explain how the Maude-NPA tool finds grammars defining co-invariants and then formally describe how these grammars are generated. We use the Needham-Schroeder Protocol (Needham and Schroeder, 1978) as the guiding example along the paper. We conclude in Section 7.

2 NRL Protocol Analyzer Overview

In the NPA, protocols are represented in terms of communicating state machines. The state machines communicate by sending messages to an intruder, who has the usual ability to read and redirect traffic, and can also perform operations, e.g., encryption, decryption, concatenation, etc. on messages that it has received. Intruder operations are described in terms of the intruder sending messages to itself. All states and protocol rules are described symbolically, using a mixture of variables and constants, so a single specification can stand for multiple instances. There is no restriction in the number of principals, number of sessions, nonces, time, or local state values, i.e., no data abstraction or approximation is performed. It is also possible to include algebraic properties of the operators (cryptographic and otherwise) that can be expressed in terms of a finite set of rewrite rules and it also allows limited support for commutativity.

Analysis is done in NPA via backwards narrowing search from an (insecure) goal state. Output states of the protocol rules are (E-)unified with subterms of goal states via narrowing using an equational theory E. Specifically, NPA

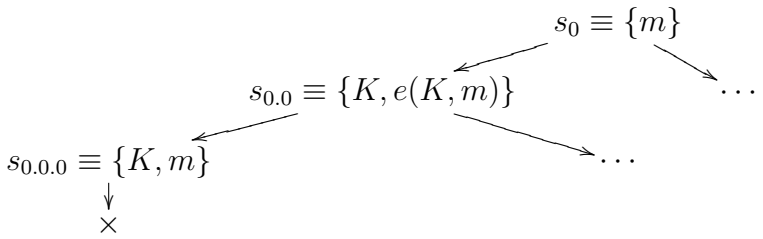

Fig. 1. Backwards protocol state exploration for Example 1

allows only equational theories that can be oriented into a finite convergent set of rewrite rules where the right-hand side of each rule is either a subterm of the left-hand side or a ground term (Kapur and Narendran, 1987; Dershowitz et al., 1992). NPA can either: (i) find an attack, i.e., a protocol run from the insecure state to an initial state, (ii) prove the protocol secure, i.e., the search space is finite and no protocol run was found, or (iii) fail to terminate.

The NPA includes a number of ways for recognizing that a state is unreachable by an intruder, thus reducing the size of the search space. We describe the two most important ones as follows. The first, upon which much of the other NPA state-space reduction functionality is built, is the notion of an intruder learning a term at a concrete moment in a protocol run. The NPA sets a condition that an intruder learns each term only once, and a history of all terms that an intruder will learn in the future is kept by the NPA in its backwards search. Note that this history is just the set of messages found by backwards search from the goal state till the present state. If, as the search proceeds, the NPA encounters a state in which the intruder knows a term which we positively know that the intruder will only learn at some moment in the future (i.e., it is actually in the current history) and thus it cannot be known at the present moment, then the NPA discards that state as unreachable.

Example 1 *Consider a protocol with only two operations, encryption, represented by* $e(K, X)$*, and decryption, represented by* $d(K, X)$*, where* K *is the key and* X *is the message. These operations satisfy the cancellation properties* $d(K, e(K, X)) = X$ *and* $e(K, d(K, X)) = X$*. Suppose also that the intruder has the capability of performing both encryption and decryption. Suppose that a goal state* $s0$ *is given in which the intruder knows a term* m*, and the NPA tells us that one of the states that can immediately precede it (obtained by backwards narrowing) is one,* $s0.0$*, in which the intruder knows* K *and* $e(K, m)$*. Since the intruder uses* K *and* $e(K, m)$ *at the present state* $s0.0$ *to produce* m *in the following state* $s0$*, we say that it actually learns* m *in this concrete protocol transition from state* $s0.0$ *to state* $s0$*, which implies that he cannot know* m *in* $s0.0$ *or any state preceding it (recall that each term is learned only once). Suppose that the NPA finds another state,* $s0.0.0$*, immediately preceding* $s0.0$ *in which the intruder knows* K *and* m*. This state is therefore unreachable, since the intruder will not learn* m *until state* $s0$*, and, as we said, cannot know it in the present state* $s0.0.0$*. Thus* $s0.0.0$ *can be dropped. Figure 1 depicts the*

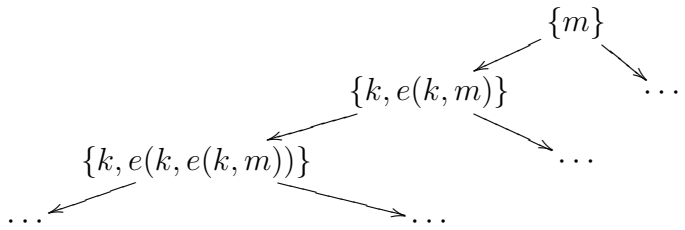

Fig. 2. Backwards protocol state exploration for Example 2

three generated states.

The other technique, one of the most powerful that the NPA uses for state space reduction, is the inductive generation of co-invariants. The idea, described in detail in (Meadows, 1996b), is to inductively generate grammars describing formal tree languages. A user starts by giving the NPA a term (with variables) that he/she believes that the intruder can't learn. This term is called a *seed term* (or *seed grammar*). The NPA then generates a language containing the seed term such that, if the intruder learns a member of the language, then it must know a member of the language in a preceding state of the protocol execution, thus inductively proving the entire language unreachable. The NPA does this by starting out with a language containing a single grammar production, which describes the seed term. It then generates a set of terms that the intruder must know in preceding states in order to be able to produce the seed term in the present state, and uses such a set of terms to construct new grammar productions. Every time the grammar changes, it tests all production rules to see if the co-invariant is still valid, i.e., to see that the knowledge of the terms described by such productions requires previous knowledge of a term described by the same productions. Wherever it fails to do so it generates new productions such that this is therefore the case, and retests again all productions in the same fashion. It continues in this way, generating new production rules and testing them, until it reaches a fixpoint [2].

Example 2 *Consider a protocol that uses the same operators as Example 1, with the same properties. Suppose that each time an honest principal A receives a message X, it outputs $d(k, X)$, where k is a constant standing for a key shared by all honest principals. We can denote this by a protocol rule $X \rightarrow d(k, X)$. In order to keep the example simple, we assume in this case that the intruder does not perform operations itself, but can only intercept and redirect messages that are sent. Suppose now that we want to find out how an intruder can learn a term m that it does not know initially. The NPA uses backwards*

[2] For some seed terms this grammar generation process may *fail* (that is, might never terminate adding always new productions, or fails to add a new production, in which case it terminates but with failure). In such cases no grammar generation fixpoint is reached for the chosen seed term, and no grammar is generated. This means that the chosen seed term cannot be used to cut down the search space, but it does not necessarily mean that the intruder is able to find the seed term.

search, so we ask what rules could produce m, and how. According to the honest principal rule $X \rightarrow d(k, X)$ and the property $d(K, e(K, X)) = X$, we have that the intruder can learn m only if it previously knows $e(k, m)$. That is, we consider the rule application $e(k, m) \rightarrow d(k, e(k, m))$, where $d(k, e(k, m)) =_E m$. We then ask the NPA how the intruder can learn $e(k, m)$, and we find that it can only happen if the intruder previously knows $e(k, e(k, m))$. Figure 2 depicts the three generated states. We see a pattern emerging, which suggests the set of terms belonging to the following formal tree language $\mathcal{L}$:

$$L \mapsto m$$

$$L \mapsto e(k, L)$$

We now want to verify the co-invariant stating that intruder knowledge of any member of $\mathcal{L}$ implies previous knowledge of some member of $\mathcal{L}$. We do this by running the NPA on the term t of each grammar production rule $L \mapsto t$. First we run the NPA on the term m, and verify that it requires intruder knowledge of $e(k, m)$ in a preceding state of the protocol, which is a member of $\mathcal{L}$. We then run the NPA on the term $e(k, L)$, and verify that it requires previous intruder knowledge of $e(k, e(k, L))$, which is also a member of $\mathcal{L}$ provided the fact that L is a member of $\mathcal{L}$.

It is also possible to combine language generation with the *learn-only-once* restriction. For example, suppose that in our search for a term X, we encounter a previous state in which the intruder knows $e(K, X)$. If we try to generate a language using the seed term $e(K, X)$, the results will not be very interesting, since the intruder can always learn $e(K, X)$ if it knows K and X. However, suppose that instead of specifying what the intruder knows we add the restriction that the *intruder does not yet know* X; which will be denoted in the paper by the expression $X \notin \mathcal{I}$. Then we can use the seed term to characterize the terms of the form $e(K, X)$ that the intruder can observe and use to learn X.

The NPA has a number of other features that it uses to limit the search space, including restrictions on the times at which the intruder can learn nonces (not before they are generated), and also a form of partial order reduction (Peled, 1998). However, in this paper we concentrate on the learns-only-once restriction and language generation, since these are the main tools used in reducing from an infinite to a finite search space.

3 Preliminaries

This section defines the terms and concepts from rewriting theory that are used in this paper. Readers already well-versed in these topics might want to

skip this section.

We assume some familiarity with term rewriting and narrowing, see (TeReSe, 2003; Meseguer, 1992) for missing definitions. In this paper, syntactical equality between elements, resp. inequality, is denoted by $e \equiv e'$, resp. $e \not\equiv e'$.

Given a binary relation $\Rightarrow \subseteq T \times T$ on a set T of elements (e.g., $\rightarrow_R \subseteq \mathcal{T}_\Sigma(\mathcal{X}) \times \mathcal{T}_\Sigma(\mathcal{X})$), we say that an element $a \in T$ is $\Rightarrow$-*irreducible* (or is a *normal form* w.r.t. $\Rightarrow$ or is a *fixpoint* of $\Rightarrow$) if there is no element $b \in T$ such that $a \Rightarrow b$. We denote the transitive closure of $\Rightarrow$ by $\Rightarrow^+$, and the transitive and reflexive closure by $\Rightarrow^*$. Also, $a \Rightarrow^! b$ denotes that $a \Rightarrow^* b$ and that b is $\Rightarrow$-irreducible. We say that the relation $\Rightarrow$ is *terminating* if there is no infinite sequence $a_1 \Rightarrow a_2 \Rightarrow \cdots \Rightarrow \cdots$. We say that $\Rightarrow$ is *confluent* if whenever $a \Rightarrow^* b$ and $a \Rightarrow^* c$, there exists an element d such that $b \Rightarrow^* d$ and $c \Rightarrow^* d$.

An *order-sorted signature* Σ is defined by a set of sorts S, a partial order relation of subsort inclusion $\leq$ on S, and an $(\mathsf{S}^* \times \mathsf{S})$-indexed family of $\{\Sigma_{\mathsf{w},\mathsf{s}}\}_{(\mathsf{w},\mathsf{s}) \in \mathsf{S}^* \times \mathsf{S}}$ operations. We denote $f \in \Sigma_{\mathsf{w},\mathsf{s}}$ by $f : \mathsf{w} \rightarrow \mathsf{s}$. In this paper, we use letters in sans-serif font $\mathsf{s}, \mathsf{s}', \ldots$ to denote sorts, and lowercase letters $f, g, h, \ldots$ to denote symbols in Σ. We define a relation $\simeq$ on S as the smallest equivalence relation generated by the subsort inclusion relation $\leq$. We assume that each equivalence class of sorts contains a *top* sort that is a supersort of every other sort in the class. Formally, for each sort s we assume that there is a sort [3] $[\mathsf{s}]$ such that $\mathsf{s} \simeq \mathsf{s}'$ implies $\mathsf{s}' \leq [\mathsf{s}]$. Furthermore, for each $f : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \rightarrow \mathsf{s}$ we assume that there is also an $f : [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n] \rightarrow [\mathsf{s}]$. We require the signature Σ to be *sensible*, i.e., whenever we have $f : \mathsf{w} \rightarrow \mathsf{s}$ and $f : \mathsf{w}' \rightarrow \mathsf{s}'$ with w, w' of equal length, then $\mathsf{w} \simeq \mathsf{w}'$ implies $\mathsf{s} \simeq \mathsf{s}'$.

A Σ-*algebra* is defined by an S-indexed family of sets $\mathsf{A} = \{\mathsf{A}_\mathsf{s}\}_{\mathsf{s} \in \mathsf{S}}$ such that $\mathsf{s} \leq \mathsf{s}'$ implies $\mathsf{A}_\mathsf{s} \subseteq \mathsf{A}_{\mathsf{s}'}$, and for each function $f : \mathsf{w} \rightarrow \mathsf{s}$ with $\mathsf{w} = \mathsf{s}_1 \times \ldots \times \mathsf{s}_n$ a function $f_{\mathsf{A}^{\mathsf{w},\mathsf{s}}} : \mathsf{A}_{\mathsf{s}_1} \times \ldots \times \mathsf{A}_{\mathsf{s}_n} \rightarrow \mathsf{A}_\mathsf{s}$. Further, we require that subsort overloaded operations agree, i.e., for each $f : \mathsf{w} \rightarrow \mathsf{s}$ and $(\mathsf{a}_1, \ldots, \mathsf{a}_n) \in \mathsf{A}^\mathsf{w}$ we require $f_{\mathsf{A}^{\mathsf{w},\mathsf{s}}}(\mathsf{a}_1, \ldots, \mathsf{a}_n) = f_{\mathsf{A}^{[\mathsf{w}],[\mathsf{s}]}}(\mathsf{a}_1, \ldots, \mathsf{a}_n)$, where if $\mathsf{w} = \mathsf{s}_1 \times \ldots \times \mathsf{s}_n$, then $[\mathsf{w}] = [\mathsf{s}_1] \times \ldots \times [\mathsf{s}_n]$. We assume a family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s} \in \mathsf{S}}$ of infinite sets of variables such that $\mathsf{s} \not\equiv \mathsf{s}'$ implies $\mathcal{X}_\mathsf{s} \cap \mathcal{X}_{\mathsf{s}'} = \emptyset$, and the variables in $\mathcal{X}$ are different from constant symbols in Σ. We use uppercase letters $X, Y, W, \ldots$ to denote variables in $\mathcal{X}$. We denote the set of ground Σ-terms and Σ-terms of sort s by $\mathcal{T}_{\Sigma_\mathsf{s}}$ and $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$, respectively. More generally, we write $\mathcal{T}_\Sigma$ for the Σ-algebra of ground terms over Σ, and $\mathcal{T}_\Sigma(\mathcal{X})$ for the Σ-algebra of terms with variables from the set $\mathcal{X}$. In this paper, we use lowercase letters $t, s, u, v, w, \ldots$ to denote terms in $\mathcal{T}_\Sigma(\mathcal{X})$. $\mathcal{V}ar(t)$ denotes the set of variables in $t \in \mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{V}ar_\mathsf{s}(t)$ denotes the set of variables of sort s. A *non-variable* term is simply a

[3] In the order-sorted specifications discussed in this paper we will sometimes leave this top sort and its associated operators implicit, in the sense that an order-sorted signature can always be conservatively completed to one satisfying our requirements.

term that is not a variable.

We use a finite sequence of positive integers, called a *position*, to denote an access path in a term. For $t \in \mathcal{T}_\Sigma(\mathcal{X})$, $\mathcal{P}os(t)$ denotes the set of positions in t, and $\mathcal{P}os_\Sigma(t)$ denotes the set of non-variable positions in t. Given a position p and a set P of positions, we define $p.P = \{p.q \mid q \in P\}$. Given two positions p, q, we denote $p.\{q\}$ also as $p.q$. The root of a term is at the empty position Λ. The subterm of t at position p is denoted by $t|_p$ and $t[s]_p$ is the term t with the subterm at position p replaced by s.

A *substitution* is a mapping $\sigma : \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{X})$ which maps a variable of sort s to a term of sort s' such that $\mathsf{s}' \leq \mathsf{s}$, and which is different from the identity only for a finite subset $\mathcal{D}om(\sigma)$ of $\mathcal{X}$. A substitution σ with $\mathcal{D}om(\sigma) = \{X_1, \ldots, X_n\}$ is usually denoted as $\sigma = [X_1/t_1, \ldots, X_n/t_n]$. The identity substitution is denoted by id, i.e., $\mathcal{D}om(id) = \emptyset$. We denote the homomorphic extension of σ to $\mathcal{T}_\Sigma(\mathcal{X})$ also by σ. The set of variables introduced by σ is $\mathcal{R}an(\sigma) = \cup_{X \in \mathcal{D}om(\sigma)} \mathcal{V}ar(\sigma(X))$. The restriction of a substitution σ to a set of variables V is defined as $\sigma{\downarrow_V}(X) = \sigma(X)$ if $X \in V$; and $\sigma{\downarrow_V}(X) = X$ otherwise. We say that a substitution σ is *away* from a set of variables V if $\mathcal{R}an(\sigma) \cap V = \emptyset$. For substitutions σ, ρ such that $\mathcal{D}om(\sigma) \cap \mathcal{D}om(\rho) = \emptyset$ we define their composition as $(\sigma \circ \rho)(X) = \rho(\sigma(X))$ for each variable $X \in \mathcal{X}$.

Given a binary relation $\Rightarrow \subseteq \mathcal{T}_\Sigma(\mathcal{X}) \times \mathcal{T}_\Sigma(\mathcal{X})$, we say that a term t is *strongly $\Rightarrow$-irreducible* if for any substitution σ such that for each $x \in \mathcal{D}om(\sigma)$, $\sigma(x)$ is $\Rightarrow$-irreducible, then $\sigma(t)$ is $\Rightarrow$-irreducible.

A Σ-*equation* is an expression of the form $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for an appropriate sort s. Order-sorted equational logic has a sound and complete inference system $E \vdash_\Sigma$ (Meseguer, 1998) inducing a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$: $t =_E t'$ if and only if $E \vdash_\Sigma t = t'$; where under the assumption that all sorts S in Σ are non-empty, i.e., $\forall \mathsf{s} \in \mathsf{S} : \mathcal{T}_{\Sigma_\mathsf{s}} \neq \emptyset$, the inference system $E \vdash_\Sigma$ can treat universal quantification in an implicit way.

The *E-subsumption* preorder $\preceq_E$ holds between $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, denoted $t \preceq_E t'$ (meaning that t' is more general than t), if there is a substitution σ such that $t =_E \sigma(t')$; such a substitution σ is said to be an *E-match* from t to t'. We write $t \preceq t'$ when E is empty, i.e., $t \preceq_\emptyset t'$. We extend this to substitutions as follows: $\sigma \preceq_E \sigma'$ iff there is a substitution θ such that $\sigma =_E \sigma' \circ \theta$.

An *E-unifier* for a Σ-equation $t = t'$ is a substitution σ such that $\sigma(t) =_E \sigma(t')$. For $\mathcal{V}ar(t) \cup \mathcal{V}ar(t') \subseteq W$, a set of substitutions $CSU_E(t = t', W)$ is said to be a *complete* set of unifiers of the equation $t =_E t'$ away from W if: (i) each $\sigma \in CSU_E(t = t', W)$ is an E-unifier of $t =_E t'$; (ii) for any E-unifier ρ of $t =_E t'$ there is a $\sigma \in CSU_E(t = t', W)$ such that $\rho{\downarrow_V} \preceq_E \sigma{\downarrow_V}$ and $V = \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$; (iii) for all $\sigma \in CSU_E(t = t', W)$, $\mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$ and $\mathcal{R}an(\sigma) \cap W = \emptyset$. An E-unification algorithm is *complete* if for

any equation $t = t'$ it generates a complete set of E-unifiers. Note that this set needs not be finite. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of solutions. We denote the unification problem $CSU_{\emptyset}(t = t', W)$, which has a unique unifier, simply as the substitution $mgu(t, t')$.

A *rewrite rule* is an expression of the form $l \to r$, where $l, r \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$ for an appropriate sort s. In this paper, we do *not* impose the usual condition $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. An *(unconditional) order-sorted rewrite theory* is a triple $\mathcal{R} = (\Sigma, \phi, E, R)$ with Σ an order-sorted signature, E a set of Σ-equations, R a set of rewrite rules, and where $\phi : \Sigma \to \mathcal{P}(\mathbb{N})$ specifies the *frozen* arguments $\phi(f) \subseteq \{1, \ldots, ar(f)\}$ of each $f \in \Sigma$. We say that a position p in t is *ϕ-frozen* (or *frozen* if ϕ is obvious) if $\exists q < p$ such that $p = q.i.q'$ and $i \in \phi(root(t|_q))$. Note that completeness of the different rewriting and narrowing relations introduced in what follows is always subject to the frozenness requirements imposed by ϕ. A *topmost rewrite theory* is a rewrite theory where rewriting and narrowing steps can only occur at the top of terms. This can occur, for instance, because the sort information and the rules force such rewrites to happen only at the top of terms, or because the frozenness specification ϕ always blocks rewrites at any proper subterm positions.

We define the *one-step rewrite relation* $\to_{\phi,R}$ on $\mathcal{T}_{\Sigma}(\mathcal{X})$ as follows: $t \xrightarrow{p}_{\phi,R} t'$ (or $\xrightarrow{p}_{R}$ if ϕ is not relevant, or $\to_R$ if p is not relevant) if there is a non-ϕ-frozen position $p \in \mathcal{P}os_{\Sigma}(t)$, a (possibly renamed) rule $l \to r$ in R such that $\mathcal{V}ar(t) \cap (\mathcal{V}ar(r) \setminus \mathcal{V}ar(l)) = \emptyset$, and a substitution σ such that $t|_p = \sigma(l)$ and $t' = t[\sigma(r)]_p$. We say that R is *terminating* (*confluent*) if the relation $\to_R$ is terminating (confluent). And we say R is *convergent* if it is terminating and confluent. The *one-step R, E-rewrite relation*[4] on $\mathcal{T}_{\Sigma}(\mathcal{X})$ is defined as follows: $t \xrightarrow{p}_{\phi,R,E} t'$ (or $\xrightarrow{p}_{R,E}$ if ϕ is not relevant, or $\to_{R,E}$ if p is not relevant) if there is a non-ϕ-frozen position $p \in \mathcal{P}os_{\Sigma}(t)$, a (possibly renamed) rule $l \to r$ in R such that $\mathcal{V}ar(t) \cap (\mathcal{V}ar(r) \setminus \mathcal{V}ar(l)) = \emptyset$, and a substitution σ such that $t|_p =_E \sigma(l)$ and $t' = t[\sigma(r)]_p$.

We define the *one-step narrowing relation* on $\mathcal{T}_{\Sigma}(\mathcal{X})$ as follows: $t \overset{p}{\leadsto}_{\sigma,\phi,R} t'$ (or $\overset{p}{\leadsto}_{\sigma,R}$ if ϕ is not relevant, or $\leadsto_{\sigma,R}$ if p is not relevant, or $\leadsto_R$ if σ is not relevant) if there is a non-ϕ-frozen position $p \in \mathcal{P}os_{\Sigma}(t)$, a (possibly renamed) rule $l \to r$ in R such that $\mathcal{V}ar(t) \cap (\mathcal{V}ar(l) \cup \mathcal{V}ar(r)) = \emptyset$, and a unifier $\sigma \equiv mgu(t|_p, l)$ such that $t' = \sigma(t[r]_p)$. The *R, E-narrowing relation* on $\mathcal{T}_{\Sigma}(\mathcal{X})$ is defined as follows, where we assume that there is a complete unification algorithm for E:

[4] A stronger alternative is the $\to_{R/E}$ relation. However, we can safely restrict ourselves to the weaker $\to_{R,E}$ when we have the following properties, satisfied by our inference system: (i) R is a topmost rewrite theory, (ii) the equations in E do not have variables of the top sort of R, and (iii) E has a complete unification algorithm, see (Meseguer and Thati, 2004) for details.

$t \overset{p}{\leadsto}_{\sigma,\phi,R,E} t'$ (or $\overset{p}{\leadsto}_{\sigma,R,E}$ if ϕ is not relevant, or $\leadsto_{\sigma,R,E}$ if p is not relevant, or $\leadsto_{R,E}$ if σ is not relevant) if there is a non-ϕ-frozen position $p \in \mathcal{P}os_\Sigma(t)$, a (possibly renamed) rule $l \to r$ in R such that $\mathcal{V}ar(t) \cap (\mathcal{V}ar(l) \cup \mathcal{V}ar(r)) = \emptyset$, and a E-unifier $\sigma \in CSU_E(t|_p = l, V)$ for $\mathcal{V}ar(t) \cup \mathcal{V}ar(l) \cup \mathcal{V}ar(r) \subseteq V$ such that $t' = \sigma(t[r]_p)$.

An important restriction in narrowing is the *basic narrowing* strategy (Hullot, 1980), which performs narrowing steps only at subterms which have not been introduced by a previous computed substitution, i.e., a subterm that belongs to the original term. Given a narrowing sequence $t_0 \overset{p_1}{\leadsto}_{\theta_1,\phi,R} t_1 \overset{p_2}{\leadsto}_{\theta_2,\phi,R} \cdots \overset{p_n}{\leadsto}_{\theta_n,\phi,R} t_n$, we inductively define the *basic positions* as $B_0 = \mathcal{P}os_\Sigma(t_0)$ and $B_i = (B_{i-1} \setminus p_i.\mathcal{P}os(t_{i-1}|_{p_i})) \cup p_i.\mathcal{P}os_\Sigma(r_i)$ where $l_i \to r_i \in R$ is used at step i. We define a *basic narrowing sequence* $s \leadsto^*_{\mathcal{B},\theta,\phi,R} t$ as $s_0 \overset{p_1}{\leadsto}_{\theta_1,\phi,R} s_1 \cdots s_{n-1} \overset{p_n}{\leadsto}_{\theta_n,\phi,R} s_n$ such that $s \equiv s_0$, $t \equiv s_n$, $\theta \equiv \theta_1 \circ \cdots \circ \theta_n$, and $p_i \in B_{i-1}$ for $1 \leq i \leq n$.

When the equations in the equational theory E can be viewed as a finite convergent set of rewrite rules $\vec{E}$, then basic narrowing with $\hat{E} = \vec{E} \cup \{x \simeq x \to True\}$ provides a sound and complete E-unification algorithm (Baader and Snyder, 2001), i.e., $\sigma{\downarrow_V} \in CSU_E(t = t', W)$ for $V = \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$ and $V \subseteq W$ iff $t \simeq t' \leadsto^!_{\mathcal{B},\sigma,\hat{E}} True$. Note that in general this doesn't provide a finitary E-unification algorithm. However, when the equations in the equational theory E can be viewed as a finite convergent set of rewrite rules $\vec{E}$ where the right-hand side of each rule is either a subterm of the left-hand side or a ground term, then the basic narrowing relation $\leadsto_{\mathcal{B},R}$ provides a sound, complete, and *finitary* E-unification algorithm (Kapur and Narendran, 1987; Dershowitz et al., 1992), i.e., $\sigma{\downarrow_V} \in CSU_E(t = t', W)$ for $V = \mathcal{V}ar(t) \cup \mathcal{V}ar(t')$ and $V \subseteq W$ iff $t \simeq t' \leadsto^!_{\mathcal{B},\sigma,\hat{E}} True$.

In this paper, we only consider equational theories E for protocols that can be converted into a finite convergent set of rewrite rules $\vec{E}$ where the right-hand side of each rule is either a subterm of the left-hand side or a ground term. We will require some protocol messages (i.e., terms) to be strongly $\to_{\vec{E}}$-irreducible and therefore we will make that explicit in the presentation.

4 Protocol Notation

The Maude-NPA's search is based on two parameters: a protocol $\mathcal{P}$, and a grammar sequence $\mathcal{G} = \langle G_1, \ldots, G_n \rangle$. The protocol $\mathcal{P}$ is the one whose security properties we want to check. A grammar G in the sequence $\mathcal{G}$ is used by Maude-NPA to *reduce the search space* explored by backwards narrowing. In this section we introduce the notation, i.e., the signature $\Sigma_\mathcal{P}$, that we use to specify protocols, and in Section 6.3 we introduce the notation, i.e., the signature $\Sigma_\mathcal{G}$, that we use to specify grammars. The concrete grammar notation will not

be relevant until Section 6, although we will introduce brief comments when necessary.

The operators in $\Sigma_{\mathcal{P}}$ and $\Sigma_{\mathcal{G}}$ are generic and apply to many different protocols. They include a special sort Msg of messages. A concrete protocol will add extra symbols involving the sort Msg in a protocol-specific signature Σ, such that $\Sigma \subseteq \Sigma_{\mathcal{G}}$ and $\Sigma \subseteq \Sigma_{\mathcal{P}}$. Special algebraic properties of a protocol may be specified with symbols in Σ and equations in E such that the sort of the terms of the equations must be Msg or smaller, i.e., $t, t' \in \mathcal{T}_{\Sigma}(\mathcal{X})_{\mathsf{s}}$ and $\mathsf{s} \leq \mathsf{Msg}$ for each $t = t' \in E$. The protocol is specified by means of a set $\mathcal{P}$ of *strands* (Fabrega et al., 1999), which are terms of a specific form in the signature $\Sigma_{\mathcal{P}}$. Our inference system will therefore be parametric on the protocol-specific syntax, equations, and strands, i.e., on the triple $(\Sigma, E, \mathcal{P})$.

Definition 1 (Protocol Signature) *Given a protocol $\mathcal{P}$, we use its protocol-specific signature Σ to build the general signature $\Sigma_{\mathcal{P}} = \Sigma \cup \Sigma_{Strand} \cup \Sigma_{\mathcal{I}} \cup \Sigma_{State}$, where signature Σ_{Strand} defines strands, signature $\Sigma_{\mathcal{I}}$ defines the intruder knowledge, and signature Σ_{State} defines a protocol state; both $\Sigma_{\mathcal{I}}$ and Σ_{State} will not be relevant until Section 5.1 below.*

We adopt a notation for specifying a protocol quite close to that of strand spaces (Fabrega et al., 1999). In a *strand*, a local execution of a protocol by a principal is indicated by a sequence of messages as shown below, where nodes representing input messages are assigned a negative sign, and nodes representing output messages are assigned a positive sign

$$[msg_1^-, msg_2^+, msg_3^-, \ldots, msg_{k-1}^-, msg_k^+]$$

We write $msg^{\pm}$ to denote msg^+ or msg^-. We use strands for backwards reachability analysis by narrowing. For this we need a mark, the symbol $|$, to divide past and future, i.e., we consider strands of the form

$$[msg_1^{\pm}, \ldots, msg_{j-1}^{\pm} \mid msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}]$$

where $msg_1^{\pm}, \ldots, msg_{j-1}^{\pm}$ are the past messages, and $msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}$ are the future messages ($msg_j^{\pm}$ is the immediate future message). Note that the mark $|$ can appear in any position in the strand. So, $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}]$ denotes a strand in its initial state, whereas $[msg_1^{\pm}, \ldots, msg_k^{\pm} \mid nil]$ denotes a strand in its final state. For simplicity, $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ will denote the strand $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}]$.

Definition 2 (Strand Signature) *Given a protocol $\mathcal{P}$, we use its protocol-specific signature Σ to build the strand signature*[5]

[5] In the Maude language (Clavel et al., 2002), operators can be defined using a mix-fix syntax where the symbol '_' denotes each argument position, i.e., '_ *op* _' denotes the infix representation of a binary symbol *op*. We use this flexible notation.

$\Sigma_{Strand} = \Sigma \cup \{\ [\ \_\ |\ \_\ ],\ \_^+,\ \_^-,\ \_,\_\ ,\ \_\ \&\ \_\ \}$. *We define sorts* SMsg, SMsgList, Strand, *and* StrandSet *built on top of sort* Msg *where the operators are typed as follows:*

$$[\_|\_] : \mathsf{SMsgList} \times \mathsf{SMsgList} \rightarrow \mathsf{Strand}$$

$$\_^+ : \mathsf{Msg} \rightarrow \mathsf{SMsg}$$

$$\_^- : \mathsf{Msg} \rightarrow \mathsf{SMsg}$$

and where the operator

$$\_,\_ : \mathsf{SMsgList} \times \mathsf{SMsgList} \rightarrow \mathsf{SMsgList}$$

is a list concatenation operator that is associative and has identity[6] *nil, and we assume that there is a subsort relation* SMsg $<$ SMsgList. *Moreover, the operator*

$$\_\ \&\ \_ : \mathsf{StrandSet} \times \mathsf{StrandSet} \rightarrow \mathsf{StrandSet}$$

is a set union operator that is associative, commutative (AC) and has identity $\emptyset$*, and we assume that there is a subsort relation* Strand $<$ StrandSet.

In this paper, we abuse the notation and write $t \in w$ to actually denote that t is a proper subterm of w such that there exist u, v such that $w \equiv u, t, v$ and $\_,\_$ is an appropriate associative (and possibly commutative) operator, i.e., w is a term denoting either a set or a list, such as a term of sort StrandSet or SMsgList. Note that, although we use AC symbols in the protocol signature $\Sigma_{\mathcal{P}}$, these are not allowed in the protocol-specific signature Σ.

In security analyses it is often necessary to use fresh unguessable values. For this we use a special sort, called Fresh, that can be used in the protocol-specific signature Σ. The meaning of a variable of sort Fresh is that it will never be instantiated by a computed E-unifier, i.e., for $t \overset{p}{\leadsto}_{\sigma,R,E} t'$ with $l \rightarrow r \in R$, we have that $\mathcal{V}ar_{\mathsf{Fresh}}(t) \cup \mathcal{V}ar_{\mathsf{Fresh}}(l) \cup \mathcal{V}ar_{\mathsf{Fresh}}(r) \not\subseteq \mathcal{D}om(\sigma)$; see Appendix A for details. For instance, one can use variables of sort Fresh for nonces in the strands. This restriction ensures that if nonces are represented using variables of sort Fresh, they will never be merged and no approximation for nonces is performed. Note that the introduction of new nonces during protocol execution is ensured by the renaming of rules performed by narrowing, and therefore two strands running different sessions of the same protocol will have different nonces. For instance, a strand of the form $[\ e(k, N)^+\ ]$ where k is a key shared by all honest principals and N is a variable of sort Fresh will produce different messages $e(k, N_1), \ldots, e(k, N_k)$. Moreover, since variables of sort Fresh can never be bound, a principal can

[6] In this paper, when we say that operator $op : \mathsf{s} \times \mathsf{s} \rightarrow \mathsf{s}$ has an identity operator op', we implicitly assume an operator declaration $op' : \rightarrow \mathsf{s}$, declaring op' as a nullary operator of sort s appearing in each signature where the operator declaration $op : \mathsf{s} \times \mathsf{s} \rightarrow \mathsf{s}$ appears.

never detect that he/she received data of sort Fresh that he/she never created. For instance, in the strand $[\ e(k,N)^-,\ e(k,N)^-\ ]$, the variable N cannot be of sort Fresh and it has to be of sort Msg. Note that the framework is very flexible and the user can specify some constant symbols of sort Fresh to play with nonces that can indeed be merged. Since variables of sort Fresh are treated in a special way, we will make them explicit in the strands by writing $(r_1,\ldots,r_k : \mathsf{Fresh})\ [msg_1^{\pm},\ldots,msg_n^{\pm}]$, where $r_1,\ldots,r_k$ are all the variables of sort Fresh appearing in $msg_1^{\pm},\ldots,msg_n^{\pm}$.

Another important aspect of our inference system is that everything the intruder can learn must be learned through strands, i.e., the intruder knows nothing in an initial state. However, this is not a limitation, since we can write strands $[\ m^+\ ]$ for any message m the intruder is able to know at an initial state.

Example 3 *The Needham-Schroeder protocol (Needham and Schroeder, 1978) uses public keys to achieve authentication between two parties, A and B. The protocol involves an initiator A, a responder B, and a key server S. We use the common notation $A \hookrightarrow B : M$ to stand for "A sends message M to B". Encryption/decryption keys are represented by K_M, denoting the key of M. Nonces, i.e., random numbers used as fresh unguessable values in messages, are represented by N_X, denoting a nonce created by X. The informal protocol description proceeds as follows.*

(1) $A \hookrightarrow S : B$
A requests B's public key from S.

(2) $S \hookrightarrow A : \{K_B, B\}_{K_S}$
S sends B's public key and name to A, signed with its key.

(3) $A \hookrightarrow B : \{A, N_A\}_{K_B}$
A sends a nonce N_A, together with its name to B, encrypted with B's key. B decrypts the message to get A's name and nonce.

(4) $B \hookrightarrow S : A$
B requests A's public key from S.

(5) $S \hookrightarrow B : \{K_A, A\}_{K_S}$
S sends A's public key and name to B, signed with its key.

(6) $B \hookrightarrow A : \{N_A, N_B\}_{K_A}$
B sends a nonce N_B and A's previous nonce N_A to A, encrypted with A's key. A decrypts the message and checks whether its previous nonce N_A is present or not. If it finds N_A, it assumes that a connection with B has been established.

(7) $A \hookrightarrow B : \{N_B\}_{K_B}$
A sends N_B to B encrypted with B's key. B decrypts the message and checks whether the decrypted result is N_B. If it is indeed N_B, it assumes that a connection with A has been established.

For the formal description of the protocol, we first discard all the steps interacting with the server, since the intruder can ask the server for any public key, and we just assume that the intruder knows all the public keys. That is, we consider the simpler version of the protocol[7] *:*

(1) $A \hookrightarrow B : \{A, N_A\}_{K_B}$
(2) $B \hookrightarrow A : \{N_A, N_B\}_{K_A}$
(3) $A \hookrightarrow B : \{N_B\}_{K_B}$

Then, we make explicit the signature Σ *describing messages, nonces, etc. A nonce* N_A *is denoted by* $n(A, r)$*, where* r *is a variable of sort* Fresh*. Concatenation of two messages, e.g.,* N_A *and* N_B*, is denoted by the operator* $\_;\_$*, e.g.,* $n(A, r)$; $n(B, r')$*. Encryption of a message* M *with the public key* K_A *of principal* A *is denoted by* $pk(A, M)$*, e.g.,* $\{N_B\}_{K_B}$ *is denoted by* $pk(B, n(B, r'))$*. We assume that all public keys are known by the intruder, so that the intruder can perform* $pk(A, m)$ *for any* A *and a known message* m*. Encryption with a secret key is denoted by* $sk(A, M)$*. The secret key of the intruder is fixed and is denoted by the constant* c*, so that the only secret key operation the intruder can perform is* $sk(c, m)$ *for a known message* m*. Note that this corresponds to a naïve implementation of RSA; other encryption systems can be encoded with a different signature, rewrite rules, and equations. The protocol-specific signature* Σ *is as follows:*

$$pk : \mathsf{Name} \times \mathsf{Msg} \to \mathsf{Enc} \qquad sk : \mathsf{Name} \times \mathsf{Msg} \to \mathsf{Enc}$$

$$c : \to \mathsf{Name}$$

$$n : \mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Nonce}$$

$$\_;\_ : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$$

together with the following subsort relations

$$\mathsf{Name}\ \mathsf{Nonce}\ \mathsf{Enc} < \mathsf{Msg}.$$

In the following we will use letters A, B *for variables of sort* Name*, letters* r, r' *for variables of sort* Fresh*, letters* M, M_1, M_2, Z *for variables of sort* Msg*, letters* L, L_1, L_2 *for variables of sort* SMsgList*, and letters* SS, SS' *for variables of sort* StrandSet*; whereas letters* X, Y *will also represent variables, but their sort will depend on the concrete position in a term. The encryption/decryption cancellation properties are described using the following equations* E*:*

$$pk(X, sk(X, Z)) = Z$$
$$sk(X, pk(X, Z)) = Z$$

[7] The entire Needham-Schroeder protocol has been analyzed in the NPA tool (Meadows, 1996a).

The two strands $\mathcal{P}$ associated to the three protocol steps shown above are:

(s1) $(r : \mathsf{Fresh})\ [pk(B, A; n(A, r))^+, pk(A, n(A, r); Z)^-, pk(B, Z)^+]$
This strand represents principal A initiating the protocol by sending his/her name and a nonce, both encrypted with B's public key, to B in the first message. Then, A receives B's response and sends a final message consisting of the rest of the message received from B.

(s2) $(r' : \mathsf{Fresh})\ [pk(B, A; W)^-, pk(A, W; n(B, r'))^+, pk(B, n(B, r'))^-]$
This strand represents principal B receiving A's first message, checking that it is the public key encryption of A's name concatenated with some value W, and then sending to A the concatenation of that value W with B's own nonce, encrypted with A's public key. Then, B receives the final message from A and verifies that the final message that it receives has B's nonce encrypted with B's public key.

The following strands describe the intruder ability to concatenate, deconcatenate, encrypt and decrypt messages according to the Dolev-Yao attacker's capabilities (Dolev and Yao, 1983):

- *Concatenation of two messages into a message.*

(s3) $[M_1^-, M_2^-, (M_1; M_2)^+]$

- *Extraction of two concatenated messages.*

(s4) $[(M_1; M_2)^-, M_1^+, M_2^+]$

- *Encryption of a message with a public key.*

(s5) $[M^-, pk(Y, M)^+]$

- *Encryption of a message with the intruder secret key.*

(s6) $[M^-, sk(c, M)^+]$

Note that we have simplified the intruder rules w.r.t. (Dolev and Yao, 1983). For strand s3, we do not need the extra strand $[M_2^-, M_1^-, (M_1; M_2)^+]$, since our asynchronous model[8] *allows M_1 to be generated independently of M_2. Similarly, we do not need $[(M_1; M_2)^-, M_2^+, M_1^+]$ for strand s4. For strands s5 and s6, we could use the standard Dolev-Yao rules and write $[Y^-, M^-, pk(Y, M)^+]$ and $[Y^-, M^-, sk(Y, M)^+]$ instead. However, since the intruder knows all public keys but only his secret key, we must then add two new sorts* PKey *and* SKey*, add two new strands $[PA^+]$ and $[c^+]$ where PA is a variable of sort* PKey*,*

[8] According to the model of Section 5.3 below, the acceptance of message M_1^- will add a constraint $M_1 \in \mathcal{I}$ to the intruder knowledge and the acceptance of message M_2^- will add another constraint $M_2 \in \mathcal{I}$ to the intruder knowledge, and both will be synchronized with different messages M^+ and thus converted into constraints $M_1 \notin \mathcal{I}$ and $M_2 \notin \mathcal{I}$ independently of the order in which $M_1 \in \mathcal{I}$ and $M_2 \in \mathcal{I}$ were introduced into the intruder knowledge.

and update the operators pk, sk and c as follows: pk : $\mathsf{PKey} \times \mathsf{Msg} \rightarrow \mathsf{Enc}$, *sk* : $\mathsf{SKey} \times \mathsf{Msg} \rightarrow \mathsf{Enc}$, *and c* : $\rightarrow \mathsf{SKey}$. *But this is equivalent to writing strand s5, which says that the intruder only has to know M, since the key Y is already known, and strand s6, which propagates symbol c into the first argument of sk.*

5 Reachability Analysis

The reachability analysis is based on the notion of a protocol state. In Section 5.1, we introduce the notation for specifying a protocol state and the intruder knowledge associated to each protocol state. In Section 5.2, we present the reachability analysis from a general point of view and explain which are the possible results of analyzing a protocol. In Section 5.3, we describe (and justify) the concrete rewrite rules used in the backwards reachability analysis; these rules are automatically obtained from the strands $\mathcal{P}$. In Section 5.4, we formally define the reachability analysis as a backwards narrowing search integrating the grammars as a test to cut-down the search space. We prove soundness and completeness of the reachability analysis in Section 5.5.

As explained in the Introduction, the reachability analysis makes use of grammars to cut down many undesirable and useless search paths. We postpone until Section 6 the discussion of how grammars are generated and focus, instead, on how they are used for reachability purposes in this section. For this section, we only need to know that grammars are produced from a set of seed terms $\{sd_1, \ldots, sd_n\}$. For each given seed term sd_i, we either succeed in generating a grammar $G^!_{sd_i}$ or fail to do so; $G^!$ denotes the fixpoint of the grammar generation process when applied to the grammar G, and G_{sd_i} denotes the grammar associated to the seed term sd_i. The fixpoint of all the seed terms for which we have obtained a fixpoint are kept in a grammar sequence $\mathcal{G} = \langle G^!_{sd_{i_1}}, \ldots, G^!_{sd_{i_m}} \rangle$, where $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$. We also assume that we can check whether a term t of sort Msg is in the language associated with some grammar G in the sequence $\mathcal{G}$ under the assumption that some constraints $\mathcal{C}$ hold. This check is denoted by $\langle \mathcal{G}, \mathcal{C} \rangle \vdash (t \in \mathcal{L})$.

5.1 State Notation

An important aspect of the reachability analysis is the notion of the *intruder knowledge*. Since we are using backwards search, we will not have a precise picture of the intruder knowledge set at each protocol state, but we use the convention that the intruder *learns a term only once*, i.e., if the intruder does learn a term in the future, then it cannot know it in the present. Thus, in our

backwards search, we keep track of the set of terms that the intruder positively knows or doesn't know at some point. In order to represent the knowledge of the intruder, we use a signature $\Sigma_{\mathcal{I}}$ allowing us to specify positive and negative constraints $t{\in}\mathcal{I}$ and $t{\notin}\mathcal{I}$ stating what the intruder knows or doesn't know. The exact point in a protocol run where a term t that was not known, i.e., $t{\notin}\mathcal{I}$, becomes known, i.e., $t{\in}\mathcal{I}$, indicates the moment when the intruder learned t.

Definition 3 (Intruder Signature) *We define the intruder signature* $\Sigma_{\mathcal{I}} = \Sigma \cup \{\ \_{\in}\mathcal{I},\ \_{\notin}\mathcal{I},\ \_{\not\approx}\_,\ \_,\_\ \}$*, the sorts* IntruderCtr *and* IntruderSet*, declare the subsort relation* IntruderCtr $<$ IntruderSet*, and introduce the operators*

$$\_{\in}\mathcal{I} : \mathsf{Msg} \rightarrow \mathsf{IntruderCtr} \qquad \_{\notin}\mathcal{I} : \mathsf{Msg} \rightarrow \mathsf{IntruderCtr}$$

$$\_{\not\approx}\_ : \mathsf{Msg} \times \mathsf{Msg} \rightarrow \mathsf{IntruderCtr}$$

and the operator

$$\_,\_ : \mathsf{IntruderSet} \times \mathsf{IntruderSet} \rightarrow \mathsf{IntruderSet}$$

which is a multiset union operator that is associative, commutative and has identity $\emptyset$*.*

Remark 1 *Given a message term t (i.e., a term of sort* Msg*) and the intruder knowledge* IK *(i.e., a term of sort* IntruderSet*) if two occurrences of the constraint term* $t{\in}\mathcal{I}$*, or two occurrences of the constraint term* $t{\notin}\mathcal{I}$*, or a constraint term* $t{\in}\mathcal{I}$ *and a constraint term* $t{\notin}\mathcal{I}$ *occur at the same time in the multiset* IK*, then we can discard such intruder knowledge as invalid, because of the learn-only-once restriction. Therefore, sort* IntruderSet *is treated as a set instead of a multiset.*

Another important aspect is how a state is represented. In the NPA tool, complex definitions of states are possible. However, in Maude-NPA we simply consider that a state is a set of strands (i.e., a term of sort StrandSet) together with the knowledge of the intruder in that state (i.e., a term of sort IntruderSet).

Definition 4 (State Signature) *We define the state signature* $\Sigma_{State} = \Sigma \cup \Sigma_{Strand} \cup \Sigma_{\mathcal{I}} \cup \{\ \{\_\},\ \_\&\_\ \}$*, the sorts* StateElm *and* State*, declare subsort relations* StateElm $<$ State*, and* StrandSet $<$ StateElm*, introduce an operator*

$$\{\_\} : \mathsf{IntruderSet} \rightarrow \mathsf{StateElm}$$

and extend the associative and commutative operator & *with identity* $\emptyset$ *to the* State *supersort*

$$\_\&\_ : \mathsf{State} \times \mathsf{State} \rightarrow \mathsf{State}.$$

Remark 2 *Given a term SS of sort* State*, there is only one subterm* $\{IK\}$

in SS where IK is a term of sort IntruderSet *and IK is not invalid as defined in Remark 1.*

Example 4 *Continuing Example 3. A possible state is (we annotate each strand in the state with the strand identifier from Example 3 that it comes from):*

$$[\ pk(B, A; n(A, r))^+,\ pk(A, n(A, r); n(B, r'))^- \mid pk(B, n(B, r'))^+\ ]\ \&\quad (s1)$$

$$[\ pk(B, A; Z)^-,\ pk(A, Z; n(B, r'))^+ \mid pk(B, n(B, r'))^-\ ]\ \&\quad (s2)$$

$$\{\ pk(B, n(B, r')) \notin \mathcal{I},\ IK\ \}$$

where the message $pk(B, n(B, r'))$ *will be sent by the initiator (first strand) and received by the responder (second strand). Another possible state is*

$$[\ pk(B, A; n(A, r))^+ \mid pk(A, n(A, r); n(B, r'))^-,\ pk(B, n(B, r'))^+\ ]\ \&\quad (s1)$$

$$[\ pk(B, A; Z)^-,\ pk(A, Z; n(B, r'))^+ \mid pk(B, n(B, r'))^-\ ]\ \&\quad (s2)$$

$$[\ nil \mid (n(A, r); n(B, r'))^-,\ pk(A, n(A, r); n(B, r'))^+\ ]\ \&\quad (s5)$$

$$[\ n(A, r)^-,\ n(B, r')^- \mid (n(A, r); n(B, r'))^+\ ]\ \&\quad (s3)$$

$$\{\ pk(B, n(B, r')) \notin \mathcal{I},\ pk(A, n(A, r); n(B, r')) \notin \mathcal{I},\ (n(A, r); n(B, r')) \notin \mathcal{I},\ IK\ \}$$

where the intruder is ready to send the message $n(A, r); n(B, r')$ *(fourth strand) to itself (third strand) in order to be able to send the message* $pk(A, n(A, r); n(B, r'))$ *later to the initiator (first strand).*

The following definition formalizes the notion of an initial state, which is relevant for the rest of the paper.

Definition 5 (Initial State) *A term SS of sort* State *is* initial *if every strand in SS is in its initial position, denoted by* $[nil \mid L]$*, and the intruder is not required to know anything, i.e., the intruder knowledge does not contain any constraint of the form* $t \in \mathcal{I}$*.*

5.2 Outline of the General Algorithm

The outline of the general search algorithm for flaw detection in the Maude-NPA is as follows.

Input:

(1) The protocol-specific signature Σ.
(2) The protocol specification $\mathcal{P}$ described as a set of strands.

(3) A sequence of "seed" terms $D = \langle sd_1, \ldots, sd_n \rangle$. These seed terms are indeed grammar productions that define an initial grammar to be further extended.
(4) A term SS_{bad} of sort State describing the insecure goal state, i.e., containing some strands required for the protocol run and any positive and negative facts about the intruder knowledge.

Output: The algorithm tries to deduce whether the protocol is safe for SS_{bad} or not. If the protocol is unsafe, Maude-NPA terminates with an intruder learning sequence (i.e., the attack trace). If the protocol is safe, the algorithm can often terminate, thanks to the drastic reduction on the search space given by the grammars, but it may in some cases loop. This provides a semi-decidable algorithm.

Algorithm: First, build the fixpoint $G^!_{sd_i}$ of the grammar $G^0_{sd_i}$ associated to each seed term sd_i w.r.t. the grammar generation process $\Rrightarrow_{\mathcal{P},G^k_{sd_i},s}$ defined below, i.e.,

$$G^0_{sd_i} \Rrightarrow_{\mathcal{P},G^0_{sd_i},s} G^1_{sd_i} \Rrightarrow_{\mathcal{P},G^1_{sd_i},s} G^2_{sd_i} \cdots G^{h-1}_{sd_i} \Rrightarrow_{\mathcal{P},G^{h-1}_{sd_i},s} G^h_{sd_i} \equiv G^!_{sd_i}$$

This process may not terminate for some seed terms and may not produce a grammar for some others. The grammar sequence $\mathcal{G}$ contains the fixpoint $G^!_{sd_i}$ of all those grammars $G^0_{sd_i}$ for which the grammar generation process successfully terminates, i.e., $\mathcal{G} = \langle G^!_{sd_{i_1}}, \ldots, G^!_{sd_{i_m}} \rangle$ where $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$ and for each $j \in \{i_1, \ldots, i_m\}$, $G^0_{sd_j} \Rrightarrow_{\mathcal{P},G^0_{sd_j},s} G^1_{sd_j} \cdots G^{h-1}_{sd_j} \Rrightarrow_{\mathcal{P},G^{h-1}_{sd_j},s} G^!_{sd_j}$. Second, for the grammar sequence $\mathcal{G}$, check reachability of SS_{bad} using the backwards reachability relation

$$\langle SS_{bad}, \epsilon \rangle \approx\!\!\!\!\!\!\rightarrow^*_{\sigma,\mathcal{P},\mathcal{G}} \langle SS_{ini}, w \rangle$$

defined below, where w is the concrete message exchange sequence.

Example 5 *Let us consider again the Needham-Schroeder Protocol of Example 3. A final attack state pattern to be given as input to the system can be:*

$$[\ pk(B, A; Z)^-,\ pk(A, Z; n(B, r'))^+,\ pk(B, n(B, r'))^-\ |\ nil\ ]\ \&$$
$$\{\ n(B, r') \in \mathcal{I},\ IK\ \}$$

This attack state pattern represents a situation in which B has completed the expected communication with someone and the intruder has learned B's nonce. For this insecure goal state, the reachability analysis returns several possible solutions, for instance the following initial state corresponding to Lowe's attack (Lowe, 1996) (we again annotate each strand in the state with the strand

identifier from Example 3 that it comes from):

$$[\ nil \mid pk(c, A; n(A,r))^+,\ pk(A, n(A,r); n(B,r'))^-,\ pk(c, n(B,r'))^+\ ]\ \& \quad (s1)$$

$$[\ nil \mid pk(c, A; n(A,r))^-,\ (A; n(A,r))^+\ ]\ \& \quad (s6)$$

$$[\ nil \mid (A; n(A,r))^-,\ pk(B, (A; n(A,r))^+\ ]\ \& \quad (s5)$$

$$[\ nil \mid pk(B, A; n(A,r))^-,\ pk(A, n(A,r); n(B,r'))^+,\ pk(B, n(B,r'))^-\ ]\ \& \quad (s2)$$

$$[\ nil \mid pk(c, n(B,r'))^-,\ n(B,r')^+\ ]\ \& \quad (s6)$$

$$[\ nil \mid n(B,r')^-,\ pk(B, n(B,r'))^+\ ]\ \& \quad (s5)$$

$$\{\ n(B,r') \notin \mathcal{I},\ pk(c, n(B,r')) \notin \mathcal{I},\ pk(A, n(A,r); n(B,r')) \notin \mathcal{I},$$
$$pk(c, A; n(A,r)) \notin \mathcal{I},\ (A; n(A,r)) \notin \mathcal{I},\ pk(B, (A; n(A,r)) \notin \mathcal{I},$$
$$pk(B, n(B,r')) \notin \mathcal{I} \qquad \}$$

And the concrete message exchange sequence w that is obtained by the reachability analysis is the following:

$$pk(c, A; n(A,r))^+ \,.\, pk(c, A; n(A,r))^- \,.\, (A; n(A,r))^+ \,.\, (A; n(A,r))^- \,.$$
$$pk(B, (A; n(A,r))^+ \,.\, pk(B, (A; n(A,r))^- \,.\, pk(A, n(A,r); n(B,r'))^+ \,.$$
$$pk(A, n(A,r); n(B,r'))^- \,.\, pk(c, n(B,r'))^+ \,.\, pk(c, n(B,r'))^- \,.\, n(B,r')^+ \,.$$
$$n(B,r')^- \,.\, pk(B, n(B,r'))^+ \,.\, pk(B, n(B,r'))^-$$

This attack (Lowe, 1996) corresponds to the following informal message exchange sequence:

(1) $A \hookrightarrow I : \{A, N_A\}_c$
A sends a nonce N_A, together with his/her name to the intruder I, encrypted with I's key c. I decrypts the message to get A's name and nonce.

(2) $I_A \hookrightarrow B : \{A, N_A\}_{K_B}$
I initiates communication with B impersonating A.

(3) $B \hookrightarrow A : \{N_A, N_B\}_{K_A}$
B sends a nonce N_B and previous A's nonce N_A to A, encrypted with A's key. A decrypts the message and checks whether it finds the previous nonce N_A or not. If A finds N_A, assumes that a connection has been established with B.

(4) $A \hookrightarrow I : \{N_B\}_c$
A thinks this is a response from I and responds with B's nonce. I now can use B's nonce to impersonate A.

(5) $I \hookrightarrow B : \{N_B\}_{K_B}$
I completes the protocol with B impersonating A.

To execute a protocol $\mathcal{P}$ (described as a set of strands) we associate to $\mathcal{P}$ a rewrite theory $R_{\mathcal{P}}$. The protocol rules $R_{\mathcal{P}}$ are obtained from the strands in $\mathcal{P}$ in an automatic way. However, we must carefully choose the rules $R_{\mathcal{P}}$ in order to reduce the backwards narrowing search space that they produce. Let us consider a naïve first approach. In a forward or backwards execution of the strands, we would need the following three rules

$$\begin{aligned}\mathbb{R} = \{\ & [\ L \mid M^-, L'\ ]\ \&\ \{M{\in}\mathcal{I}, IK\} \to [\ L, M^- \mid L'\ ]\ \&\ \{M{\in}\mathcal{I}, IK\}, && (1)\\ & [\ L \mid M^+, L'\ ]\ \&\ \{IK\} \to [\ L, M^+ \mid L'\ ]\ \&\ \{IK\}, && (2)\\ & [\ L \mid M^+, L'\ ]\ \&\ \{M{\notin}\mathcal{I}, IK\} \to [\ L, M^+ \mid L'\ ]\ \&\ \{M{\in}\mathcal{I}, IK\}\ \} && (3)\end{aligned}$$

where $L, L', L_1, L_1', L_2, L_2', L_2''$ are variables of sort SMsgList, M is a variable of sort Msg, and IK is a variable of sort IntruderSet. Rule (1) synchronizes an input message with a message already learned by the intruder, symbolizing that the intruder knows such message at that moment. Note that when this rule is applied backwards, it can introduce new facts $t{\in}\mathcal{I}$ by unification with an intruder knowledge variable IK. Rule (2) accepts output messages but the intruder knowledge is not increased, symbolizing that the message has been generated but the intruder doesn't require such message for an attack. Rule (3) accepts output messages and the intruder knowledge is positively increased. Again, this rule can introduce new facts $t{\in}\mathcal{I}$ by unification with an intruder knowledge variable IK.

In a regular forward execution of the protocol, we start with the strands describing the protocol in the initial state, e.g., strands (s1) and (s2) of Example 3 for the NSPK example. We then use rules (1)–(3) to move the bars of the strands to the right until reaching the final state. In a backwards execution, we use strands in a final state position and rules (1)–(3) in reverse. However, in an intruder attack we can have many partially executed strands together with many intruder strands from the Dolev-Yao attacker's capabilities, i.e., strands (s3)–(s6). Thus, an initial or final state in our tool might involve an unbounded number of strands, which would be unfeasible. To avoid this problem we can use a more perspicuous set of rewrite rules describing the protocol, where the necessary strands are introduced dynamically. The key idea is to specialize rule (3) using the different protocol strands.

In a backwards execution, which is the one we are interested in, we add the following specializations of rule (3):

$$R_{\mathcal{P}} = \mathbb{R} \cup \{[\, l_1 \,|\, u^+, l_2\,] \,\&\, \{u{\notin}\mathcal{I}, IK\} \to \{u{\in}\mathcal{I}, IK\} \text{ s.t. } [\, l_1,\, u^+,\, l_2\,] \in \mathcal{P}\} \quad (4)$$

where l_1, l_2 are terms of sort SMsgList, u is a term of sort Msg, and IK is a

variable of sort IntruderSet.

Definition 6 (Protocol Rewrite Theory) *We associate to $\mathcal{P}$ a rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, \phi_\mathcal{P}, E_\mathcal{P}, R_\mathcal{P})$, where $R_\mathcal{P}$ contains the protocol rules obtained from the strands $\mathcal{P}$ using rules (1), (2), (3), and (4), $E_\mathcal{P}$ contains the protocol-specific equational theory E plus equations for associativity, commutativity and identity of the $\_\&\_$ operator, associativity and identity of the $\_,\_$ operator for sort* SMsgList *and associativity, commutativity and identity of the $\_,\_$ operator for sort* IntruderSet*; and the frozenness function $\phi_\mathcal{P}$ is $\phi_\mathcal{P}(\_^-) = \{1\}$, $\phi_\mathcal{P}(\_\notin\mathcal{I}) = \{1\}$, $\phi_\mathcal{P}(f) = \{1, \ldots, ar(f)\}$ for $f \in \Sigma$, and $\phi_\mathcal{P}(f) = \emptyset$ for the remaining. Note that expressions of the form $m\in\mathcal{I}$ inherit the frozenness restrictions from the expressions (m^- or m^+) they are derived from via the rewrite rules $R_\mathcal{P}$.*

Note that these frozenness restrictions for $\Sigma_\mathcal{P}$ imply that input messages, some of the messages stored in the intruder knowledge, and all the arguments of the messages themselves are assumed to be strongly $\rightarrow_{\vec{E}}$-irreducible. In Example 3, it implies that no input message of the form $pk(A, Z)^-$ or $sk(A, Z)^-$ where Z is a variable of sort Msg, can appear in a strand, e.g., Z could be instantiated to $sk(A, Y)$ and then $pk(A, sk(A, Y))^-$ is not $\rightarrow_{\vec{E}}$-irreducible. Note also that $\mathcal{R}_\mathcal{P}$ is defined as a topmost theory, i.e., terms of sort State can be rewritten and narrowed only at the top position using $R_\mathcal{P}$. And note that the rewrite theory $\mathcal{R}_\mathcal{P}$ is used in the Maude-NPA in a backwards way, i.e., we will therefore use the backwards narrowing relation $\rightsquigarrow_{R_\mathcal{P}^{-1}, E_\mathcal{P}}$ where $R_\mathcal{P}^{-1} = \{r \rightarrow l \mid l \rightarrow r \in R_\mathcal{P}\}$.

Example 6 *Given the strands of Example 3, the associated strand rules are the following.*

- *First the fixed rules of $\mathbb{R}$.*

(r1) $[\ L \mid M^-, L'\ ]\ \&\ \{M\in\mathcal{I}, IK\} \rightarrow [\ L, M^- \mid L'\ ]\ \&\ \{M\in\mathcal{I}, IK\}$
(r2) $[\ L \mid M^+, L'\ ]\ \&\ \{IK\} \rightarrow [\ L, M^+ \mid L'\ ]\ \&\ \{IK\}$
(r3) $[\ L \mid M^+, L'\ ]\ \&\ \{M\notin\mathcal{I}, IK\} \rightarrow [\ L, M^+ \mid L'\ ]\ \&\ \{M\in\mathcal{I}, IK\}$

The strands of Example 3 are transformed into the following rules. They model the inclusion of new strands.

- *Strand (s1) is transformed into the following two rules:*

(r4) $[\ pk(B, A; n(A, r))^+,\ pk(A, n(A, r); Z)^- \mid pk(B, Z)^+\ ]\ \&$
$\{\ pk(B, Z)\notin\mathcal{I},\ IK\ \}$
$\rightarrow \{\ pk(B, Z)\in\mathcal{I},\ IK\ \}$
For instance, this rule describes, when executed in a backwards way, that the intruder learns a term $pk(B, Z)$ because of a new strand being executed. It represents also the introduction of a new protocol run in parallel, i.e., a new session.

(r5) $[\ nil \mid pk(B, A; n(A, r))^+,\ pk(A, n(A, r); Z)^-,\ pk(B, Z)^+\ ]\ \&$

$\{\ pk(B, A; n(A, r)) \notin \mathcal{I},\ IK\ \}$

$\rightarrow \{\ pk(B, A; n(A, r)) \in \mathcal{I},\ IK\ \}$

- *Strand (s2) is transformed into the following rule:*

(r6) $[\ pk(B, A; Z)^- \mid pk(A, Z; n(B, r))^+,\ pk(B, n(B, r))^-\ ]\ \&$

$\{\ pk(A, Z; n(B, r)) \notin \mathcal{I},\ IK\ \}$

$\rightarrow \{\ pk(A, Z; n(B, r)) \in \mathcal{I},\ IK\ \}$

- *Strand (s3) is transformed into the following rule:*

(r7) $[\ M_1^-,\ M_2^- \mid (M_1; M_2)^+\ ]\ \&\ \{\ (M_1; M_2) \notin \mathcal{I},\ IK\ \} \rightarrow \{\ (M_1; M_2) \in \mathcal{I},\ IK\ \}$

- *Strand (s4) is transformed into the following two rules:*

(r8) $[\ (M_1; M_2)^-,\ M_1^+ \mid M_2^+\ ]\ \&\ \{\ M_2 \notin \mathcal{I},\ IK\ \} \rightarrow \{\ M_2 \in \mathcal{I},\ IK\ \}$

(r9) $[\ (M_1; M_2)^- \mid M_1^+,\ M_2^+\ ]\ \&\ \{\ M_1 \notin \mathcal{I},\ IK\ \} \rightarrow \{\ M_1 \in \mathcal{I},\ IK\ \}$

- *Strand (s5) is transformed into the following rule:*

(r10) $[\ M^- \mid pk(Y, M)^+\ ]\ \&\ \{\ pk(Y, M) \notin \mathcal{I},\ IK\ \} \rightarrow \{\ pk(Y, M) \in \mathcal{I},\ IK\ \}$

- *Strand (s6) is transformed into the following rule:*

(r11) $[\ M^- \mid sk(c, M)^+\ ]\ \&\ \{\ sk(c, M) \notin \mathcal{I},\ IK\ \} \rightarrow \{\ sk(c, M) \in \mathcal{I},\ IK\ \}$

Then, given the following state:

$[\ pk(B, A; n(A, r))^+,\ pk(A, n(A, r); n(B, r))^- \mid pk(B, n(B, r))^+\ ]\ \&$

$[\ pk(B, A; Z)^-,\ pk(A, Z; n(B, r))^+ \mid pk(B, n(B, r))^-\ ]\ \&$

$\{\ pk(B, n(B, r)) \in \mathcal{I}\ \}$

Using backwards narrowing modulo $E_{\mathcal{P}}$, we can apply rule r8 and obtain the following state:

$[\ pk(B, A; n(A, r))^+,\ pk(A, n(A, r); n(B, r))^- \mid pk(B, n(B, r))^+\ ]\ \&$

$[\ pk(B, A; Z)^-,\ pk(A, Z; n(B, r))^+ \mid pk(B, n(B, r))^-\ ]\ \&$

$[\ (M_1; pk(B, n(B, r)))^-,\ M_1^+ \mid pk(B, n(B, r))^+\ ]\ \&$

$\{\ pk(B, n(B, r)) \notin \mathcal{I}\ \}$

where the third strand says that the message $pk(B, n(B, r))$ received by the second strand is learned (and thus produced) by the intruder, who obtained such message from another message $(M_1; pk(B, n(B, r)))$, that he/she still needs to resolve.

The key requirement for an inference system that can make use of the grammar generation technique is that it supports the notion of an intruder who knows some terms in the present and past, and will learn more terms in the future. The reachability inference system offered by the Maude-NPA meets this requirement.

Before defining the main backwards reachability relation $\rightsquigarrow_{\mathcal{P},\mathcal{G}}$, we must introduce a procedure for the unification of variables in the intruder knowledge, necessary for a correct reachability algorithm, as shown in the following example.

Example 7 *Consider a protocol with a unique strand* $[\ t^+\ ]$ *where* t *is a fixed message without* Fresh *variables. And consider an initial bad state such as* $\{\ X{\in}\mathcal{I},\ Y{\in}\mathcal{I}\ \}$ *declaring that the intruder is able to learn two pieces of information* X *and* Y *and no strand is provided. If we run the protocol, we expect the initial state* $[\ nil \mid t^+\ ]\ \&\ [\ nil \mid t^+\ ]\ \&\ \{\ t{\notin}\mathcal{I},\ t{\notin}\mathcal{I}\ \}$ *declaring that the intruder was able to learn the message* t *twice. However, because of the learn-only-once restriction, such an initial state is invalid (see Remark 1) and the appropriate initial state is* $[\ nil \mid t^+\ ]\ \&\ \{\ t{\notin}\mathcal{I}\ \}$*, computable only if the variables* X *and* Y *are unified at some point.*

Therefore, before applying a backwards narrowing step using the protocol rules, we must check if some variables in the intruder knowledge can be unified and in such a case, create two versions of the same state, one where they are indeed unified and another one where they are necessarily different, denoted by the intruder constraint $X{\not\approx}Y$ given in Definition 3.

Definition 7 (Splitting of Intruder Knowledge) *Given a term* SS *of sort* State*, the relation* $\Rightarrow_{\sigma,splitIK}$ *is defined as follows:*

$$SS\ \&\ \{\ IK\ \} \Rightarrow_{id,splitIK} SS\ \&\ \{\ IK,\ X{\not\approx}Y\ \}\ \textit{and}$$

$$SS\ \&\ \{\ IK\ \} \Rightarrow_{\sigma,splitIK} \sigma(SS\ \&\ \{\ IK'\ \})$$

$$\textit{if}\ \exists X, Y \in \mathcal{X}\ \textit{and}\ \sigma\ \textit{s.t.}\ (X{\in}\mathcal{I}) \in IK,\ (Y{\in}\mathcal{I}) \in IK,\ (X{\not\approx}Y) \notin IK,$$

$$\sigma = mgu(X, Y),\ \textit{and}\ IK'\ \textit{is}\ IK\ \textit{without the term}\ (Y{\notin}\mathcal{I}).$$

A sequence $S \Rightarrow_{\sigma_1,splitIK} S_1 \Rightarrow_{\sigma_n,splitIK} \cdots \Rightarrow_{\sigma_n,splitIK} S_k$ *is denoted by* $S \Rightarrow^*_{\sigma,splitIK} S_k$ *where* $\sigma = \sigma_1 \circ \cdots \circ \sigma_n$.

Remark 3 *Given a term* SS *of sort* State*, if we have a constraint of the form* $t{\not\approx}t$ *in the intruder knowledge* IK *of* SS*, then we can discard* SS *as invalid.*

We define the backwards narrowing reachability relation $\approx\!\!\!\!\!\!\rightarrow_{\sigma,\mathcal{P},\mathcal{G}}$.

Definition 8 (Backwards Reachability) *Given a term SS of sort* State*, the backwards reachability relation $\approx\!\!\!\!\!\!\rightarrow_{\mathcal{P},\mathcal{G}}$ is defined by the equivalence*

$$\langle SS, w\rangle \approx\!\!\!\!\!\!\rightarrow_{\sigma\circ\sigma',\mathcal{P},\mathcal{G}} \langle SS'', \sigma\circ\sigma'(m^{\pm} \,.\, w)\rangle$$

$$\textit{iff } SS \Rightarrow^{!}_{\sigma,splitIK} SS',\ SS' \rightsquigarrow^{\bullet}_{\sigma',\phi_{\mathcal{P}},R_{\mathcal{P}}^{-1},E_{\mathcal{P}}} SS'',\ \textit{and } SS'' \textit{ is } \mathcal{G}\textit{-safe}$$

where SS', SS'' are also terms of sort State*, w is the concrete message exchange sequence described using the list concatenation operator $\_.\_$ which is associative and has identity ϵ, and $m^{\pm}$ is the concrete message expression moved from past to future, i.e., there is a strand $[\ l \mid m^{\pm},\ l'\ ]$ in SS'' such that $[\ l,\ m^{\pm} \mid l'\ ]$ appeared in SS' or $[\ l \mid m^{\pm},\ l'\ ]$ has been added to SS''.*

Its auxiliary relations are depicted in Figure 3. We will write $\approx\!\!\!\!\!\!\rightarrow_{\mathcal{P},\mathcal{G}}$ when the concrete computed substitution is not relevant. The auxiliary (backwards) narrowing relation $\rightsquigarrow^{\bullet}_{\sigma,\phi_{\mathcal{P}},R_{\mathcal{P}}^{-1},E_{\mathcal{P}}}$ is defined in Appendix A. And the notion of $\mathcal{G}$*-safe* state is defined as follows.

Definition 9 ($\mathcal{G}$-safe Protocol State) *Given a term SS of sort* State *and a grammar sequence $\mathcal{G}$, we say that SS is* $\mathcal{G}$-safe *if, for IK the intruder knowledge in SS, for each strand $[\ l \mid l'\ ] \in SS$, and for each m^{-} in l, the term $(m{\notin}\mathcal{I})$ does not appear in IK and $\langle \mathcal{G}, filter^{\notin}(IK)\rangle \not\vdash (m{\in}\mathcal{L})$, where $filter^{\notin}(IK)$ returns all the constraints of the form $\_{\notin}\mathcal{I}$ in IK. We write G-safe instead of $\mathcal{G}$-safe when we want to emphasize that a single grammar G (possibly not in $\mathcal{G}$) is used.*

The intuition behind the notion of a $\mathcal{G}$*-safe* state is that a state is $\mathcal{G}$-safe if it is not discarded by the learn-only-once restriction and it is not captured by the grammars $\mathcal{G}$, where by being captured we mean that there is an input message in a strand of the state that is a member of the language defined by a grammar G in the sequence $\mathcal{G}$.

Recall that messages inside the operators $\_^{-}$ and $\_{\notin}\mathcal{I}$ are *frozen*, since we assume that they are always strongly $\rightarrow_{\vec{E}}$-irreducible.

Although grammars are explained in detail in Section 6, we give the following example of how they are used to cut the search space for motivational purposes.

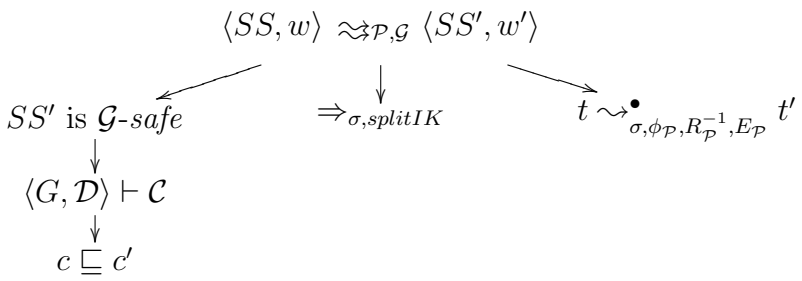

Fig. 3. Dependencies between relations and operators in the Reachability Phase

Example 8 *Continuing Example 6. For the state:*

$$[\ pk(B, A; n(A, r))^+,\ pk(A, n(A, r); n(B, r))^- \mid pk(B, n(B, r))^+\ ]\ \&$$
$$[\ pk(B, A; Z)^-,\ pk(A, Z; n(B, r))^+ \mid pk(B, n(B, r))^-\ ]\ \&$$
$$[\ (M_1; pk(B, n(B, r)))^-,\ M_1^+ \mid pk(B, n(B, r))^+\ ]\ \&$$
$$\{\ pk(B, n(B, r))\notin \mathcal{I}\ \}$$

we can use the following grammar G (where X, Y are variables of sort Msg*, B is a variable of sort* Name*, and r is a variable of sort* Fresh*):*

$$Y\notin\mathcal{I},\ Y\not\preceq n(B, r) \mapsto X; Y\in\mathcal{L}$$

to conclude that this state is not G-safe, i.e., the input message $M_1; pk(B, n(B, r))$ in the third strand is a member of the formal tree language of G and so it is unreachable for an intruder. This grammar G describes a formal tree language containing any concatenation message $t_1; t_2$ of two messages t_1 and t_2, where subterm t_2 is positively not known by the intruder (i.e., $t_2\notin\mathcal{I}$) and t_2 is not a nonce (i.e., $t_2\not\preceq n(B, r)$).

5.5 Soundness and Completeness

Soundness of the reachability algorithm is now clear by the definition of the backwards narrowing relation $\approx\!\!\!\!\rightarrow^*_{\mathcal{P},\mathcal{G}}$.

Theorem 1 (Soundness) *Let $\mathcal{P}$ be a protocol and let SS_{bad} be a final attack state pattern. If $\langle SS_{bad}, \epsilon\rangle \approx\!\!\!\!\rightarrow^*_{\sigma,\mathcal{P},\mathcal{G}} \langle SS_{ini}, w\rangle$ and SS_{ini} is an initial state term, then $SS_{ini} \rightarrow^*_{R_{\mathcal{P}},E_{\mathcal{P}}} \sigma(SS_{bad})$.*

To establish completeness of the reachability algorithm, we need an auxiliary result.

Theorem 2 (Topmost Strong Completeness) *(Meseguer and Thati, 2004) Let $\mathcal{R} = (\Sigma, \phi, E, R)$ be a topmost rewrite theory, $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, and let σ be a substitution such that $\sigma(t) \rightarrow^*_{R,E} t'$. Then, there are substitutions θ, τ and a term t'' such that $t \rightsquigarrow^*_{\theta,R,E} t''$, $\sigma(t) \equiv \tau(\theta(t))$, and $t' \equiv \tau(t'')$.*

Completeness of the reachability analysis follows from Theorem 2 and Theorem 9 below, which provides soundness of grammars, i.e., soundness of the $\mathcal{G}$-safe test.

Theorem 3 (Completeness) *Let $\mathcal{P}$ be a protocol and let $\mathcal{G}$ be a set of generated grammars. Let σ be a substitution, SS_{ini} be an initial state term, and SS_{bad} be a final state term such that $\sigma(SS_{ini}) \rightarrow^*_{R_\mathcal{P},E_\mathcal{P}} \sigma(SS_{bad})$. Then, there are substitutions θ, τ and an initial* State *term SS'_{ini} such that $\langle SS_{bad}, \epsilon\rangle \quad \approx\!\!\!\!\!\!\rightarrow^*_{\theta,\mathcal{P},\mathcal{G}} \quad \langle \theta(SS'_{ini}), w\rangle$, $\sigma(SS_{bad}) \equiv \tau(\theta(SS_{bad}))$, and $\sigma(SS_{ini}) \equiv \tau(\theta(SS'_{ini}))$.*

Proof. There are three issues here: (i) the relation $\Rightarrow_{\rho,splitIK}$, (ii) topmost narrowing using $\rightsquigarrow^\bullet_{\sigma,\phi_\mathcal{P},R^{-1}_\mathcal{P},E_\mathcal{P}}$, and (iii) the $\mathcal{G}$-safe test. If we have an attack $\sigma(SS_{ini}) \rightarrow^*_{R_\mathcal{P},E_\mathcal{P}} \sigma(SS_{bad})$ for a substitution σ, there cannot be more than one occurrence of a constraint $t \not\in \mathcal{I}$ in the intruder knowledge of SS_{ini} due to the learn-only-once restriction of Remark 1. If there are some variables in SS_{bad} that σ unifies, then the relation $\Rightarrow_{\rho,splitIK}$ would perform such unification of variables. Otherwise, the relation $\Rightarrow_{\rho,splitIK}$ would introduce constraints of the form $t \not\approx s$ that do not affect the reachability process. The rewrite theory used in our backwards reachability analysis is not topmost in the strict sense, but given the fact that it rewrites a flat set modulo associativity, commutativity, and identity, it is *ACU-topmost* in the sense of (Meseguer and Thati, 2004) and is therefore semantically equivalent to a topmost theory in the strict sense, so that Theorem 2 applies. This means that the general backwards search narrowing analysis is complete. It now remains to be shown that completeness is not lost by using the strategy associated to the grammar sequence $\mathcal{G}$, i.e., the $\mathcal{G}$-safe test. This follows from Theorem 9 in Section 6.6, since the paths removed by the grammars $\mathcal{G}$ are paths leading to *unreachable* states, i.e., states unreachable from the initial state. □

6 Grammar Generation

Now, we focus on how grammars are generated by the Maude-NPA tool. In Section 6.1, we informally explain how the Maude-NPA tool generates grammars defining formal tree languages. For generating grammars, we do not need the strand representation of protocols defined in Section 4, so we provide in Section 6.2 an abstraction of the protocols into rewriting rules. In Section 6.3,

we introduce the notation that we use to specify grammars and how they are transformed into a rewrite theory for membership purposes. In Section 6.4, we define how membership of a message into a grammar's language is performed. In Section 6.5, we formally describe how grammars are generated in terms of narrowing and rewriting. We provide the dependencies between the different operators used to generate grammars in Figure 5 below. Soundness of the grammars, i.e., that they describe states unreachable for an intruder, is proved in Section 6.6. We show how the grammar generation process is applied to the Needham-Schroeder example throughout this entire section.

6.1 How Maude-NPA Generates Languages

In this section we describe the Maude-NPA's language generation strategy in broad outline, using Example 2 as an illustration. Recall that in Example 2, we had a protocol rule $X \rightarrow d(k, X)$, an equation property $d(K, e(K, X)) = X$, and the (informally described) grammar $\mathcal{L} \equiv \{\ L \mapsto m,\ L \mapsto e(k, L)\ \}$ arose.

The Maude-NPA starts out with a simple seed term. This seed term defines an initial language stating only that the seed term is in the language. In Example 2, this seed term m is represented by the initial grammar $\{L \mapsto m\}$. The Maude-NPA strategy for generating languages involves three stages: (i) the term generation stage, (ii) the rule verification stage, and (iii) the rule generation stage.

In the *term generation* stage, the Maude-NPA takes each term defined by a language production and finds a complete set S of paths to the state in which the intruder knows that term, where by "complete" we mean that any path from an initial state to that term must contain a path from S as a subpath. In Example 2, there is only one path to the seed term m, namely the path in which the intruder sends $e(k, m)$ to an honest principal.

In the *rule verification* stage, the Maude-NPA examines each path and determines which paths already require the intruder to know a member of the language in order to produce the goal, and thus removes those paths from consideration. In the first iteration of Example 2, the single generated path only requires that the intruder knows $e(k, m)$ in order to learn the goal m. At this stage $e(k, m)$ has not yet been defined to be a member of the language, so that path stays in.

In the *rule generation* stage, the Maude-NPA looks at the remaining paths, and generates a new set of grammar rules according to a set of heuristics. For instance, one such heuristic says that, if a path contains a term containing a word in the language as a subterm, then replace that subterm by a variable W and add a condition saying that W is in the language, to obtain a new grammar

rule. In Example 2, this heuristic generates the grammar rule $L \mapsto e(k, L)$, where the non-terminal L can be seen as such a variable W.

After the rule generation stage, the Maude-NPA reiterates the three stages until either all paths are eliminated in the rule verification stage, in which case it has successfully defined a language, or it can define no new language rules in the rule generation stage, in which case it has failed to define a language. It can also conceivably fail to terminate adding new rules forever.

In Example 2, the Maude-NPA successfully defines a language. In the second iteration, the Maude-NPA shows that the sole path generated by each generic term defined by a language rule contains a member of the language. At this point it terminates with success.

In actual fact, the Maude-NPA interleaves the term generation stage and the rule verification stage. And, as we shall see, both the rule verification and rule generation stages can be considerably more complex than for the simple example given here. However, this example should help the reader understand in broad outline the more detailed ideas that we present below.

6.2 Simplified Protocol Rules and Their Execution

To facilitate the grammar generation process, we use an abstract version of the protocol rules of Section 5.3. We use a more abstract and simple description of the protocol rules that extracts from the strands the information of what terms the intruder must know in order to send a term. This approach is similar to other rule-based or clause-based approaches (Weidenbach, 1999; Blanchet, 2001; Genet and Klay, 2000) and the multiset rewriting formalism (Bistarelli et al., 2005). A detailed comparison is left for future work, since there are many differences, e.g., we don't really have multisets but sets, and the number of messages is not increasing (non-monotonic or non-cumulative), etc. The reason is that our abstraction keeps only information on how the intruder is able to produce a concrete message and thus removes all the unnecessary data about strands and the intruder knowledge. Moreover, it doesn't represent the actual intruder knowledge nor the reachability process as in other approaches. Note that the instantiation restriction for variables of sort Fresh defined in Section 4 does not apply here, and thus nonces can be merged.

We break each strand up into substrands, with one substrand for each place where a negative node directly precedes a positive node. We represent the substrand as a protocol rule of the form $u_1, \ldots, u_n \rightarrow v$, where $v, u_1, \ldots, u_n$ are terms of sort Msg, the $u_1, \ldots, u_n$ are all the negative nodes preceding v, and v is a positive node. Thus, the left-hand side of a protocol rule describes what the intruder must know in order to produce v. If the intruder observes or

creates $u_1, \ldots, u_n$, then can send them to a principal in the appropriate order, and that principal will then produce v, and if the principal does not receive $u_1, \ldots, u_n$, then will not produce v.

Definition 10 (Simplified Protocol Rewrite Theory) *The simplified protocol rewrite theory is defined as* $\mathcal{R}_{\mathcal{SP}} = (\Sigma_{\mathcal{SP}}, \phi_{\mathcal{SP}}, E_{\mathcal{SP}}, R_{\mathcal{SP}})$. *The signature is defined as* $\Sigma_{\mathcal{SP}} = \Sigma_{\mathcal{G}} \cup \{ \_,\_ \}$ *($\Sigma_{\mathcal{G}}$ is defined in Section 6.3.1 below) where the operator* $\_,\_ : \mathsf{MsgSet} \times \mathsf{MsgSet} \rightarrow \mathsf{MsgSet}$ *is a set union operator that is associative, commutative, and has identity $\emptyset$, and we assume that there is a subsort relation* $\mathsf{Msg} < \mathsf{MsgSet}$. *The rewrite rules are defined as*

$$R_{\mathcal{SP}} = \{ neg(l) \rightarrow m \;\; s.t. \;\; [\, l, m^+, l' \,] \in \mathcal{P} \;\}$$

where

$$neg(l) = \begin{cases} (m, neg(l')) & \text{if } l \equiv (m^-, l') \\ neg(l') & \text{if } l \equiv (m^+, l') \\ \emptyset & \text{if } l \equiv nil \end{cases}$$

The frozenness function is $\phi_{\mathcal{SP}}(\_ \mapsto \_) = \{1\}$ *and* $\phi_{\mathcal{SP}}(f) = \phi_{\mathcal{G}}(f)$ *for the remaining* $f \in \Sigma_{\mathcal{G}}$ *(symbol* $\_ \mapsto \_ \in \Sigma_{\mathcal{G}}$ *describes a grammar rule and $\phi_{\mathcal{G}}$ is the frozenness function for grammars; both are defined in Section 6.3). And the equational theory* $E_{\mathcal{SP}}$ *is defined as the protocol-specific equations E explained in Section 4, together with the equations for associativity, commutativity and identity of the* $\_,\_$ *operator.*

Recall that the frozenness restriction $\phi_{\mathcal{SP}}(\_ \mapsto \_) = \{1\}$ implies that the first argument of the symbol $\_ \mapsto \_$ is frozen. The rewrite theory $\mathcal{R}_{\mathcal{SP}}$ is used in the Maude-NPA in a backwards way, i.e., we will therefore use $\leadsto_{R^{-1}_{\mathcal{SP}}, E_{\mathcal{SP}}}$ where $R^{-1}_{\mathcal{SP}} = \{r \rightarrow l \mid l \rightarrow r \in R_{\mathcal{SP}}\}$.

Example 9 *Consider the signature Σ of Example 3. In the rewrite theory* $\mathcal{R}_{\mathcal{SP}} = (\Sigma_{\mathcal{P}}, \phi_{\mathcal{SP}}, E_{\mathcal{SP}}, R_{\mathcal{SP}})$, *the rules* $R_{\mathcal{SP}}$ *are listed below (rules p1-p7):*

- *Strand (s1) is transformed into the following rules:*

(p1) $\emptyset \rightarrow pk(B,\ A; n(A, r))$

(p2) $pk(A, n(A, r); Z) \rightarrow pk(B, Z)$

- *Strand (s2) is transformed into the following rule:*

(p3) $pk(B, A; Z) \rightarrow pk(A, Z; n(B, r))$

Rules p4-p7 describe the intruder abilities according to strands (s3)–(s6):

Strand (s3) is transformed into the following rule:

(p4) $M_1, M_2 \rightarrow M_1; M_2$

- *Strand (s4) is transformed into the following two rules:*

(p5) $M_1; M_2 \rightarrow M_1$

(p6) $M_1; M_2 \rightarrow M_2$

- *Strand (s5) is transformed into the following rule:*

(p7) $M \rightarrow pk(Y, M)$

- *Strand (s6) is transformed into the following rule:*

(p8) $M \rightarrow sk(c, M)$

The following results relate the narrowing relations associated to the two rewrite theories $\mathcal{R}_{\mathcal{P}}$ and $\mathcal{R}_{S\mathcal{P}}$ of a protocol $\mathcal{P}$. First, we introduce an auxiliary definition for relating a protocol state and an abstract protocol state.

Definition 11 (Abstract Protocol Representation) *We define a transformation function* $mset : \mathsf{State} \rightarrow \mathsf{MsgSet}$ *from a term* SS *of sort* State *into a term of sort* MsgSet *as*

$$mset(SS) = \{m \mid (m{\in}\mathcal{I}) \in IK, \ \textit{or} \\ (m{\notin}\mathcal{I}) \notin IK \ \textit{and} \ \exists[\ l \mid l'\ ] \in SS \ \textit{s.t.}\ m \in neg(l)\}$$

where IK *is the intruder knowledge appearing in the term* $\{IK\}$ *in* SS.

Intuitively, $mset(SS)$ denotes all the messages the intruder has to learn, i.e., input messages m^- in the strands in SS and each message $m{\in}\mathcal{I}$ in the intruder knowledge IK in SS.

In the following, $\leadsto^{\{0,1\}}_{\sigma,R,E}$ denotes zero or one narrowing steps of the relation $\leadsto_{\sigma,R,E}$. Note that, since the abstract form $\mathcal{R}_{S\mathcal{P}}$ of $\mathcal{P}$ can bind variables of sort Fresh (i.e., merge nonces), unlike $\mathcal{R}_{\mathcal{P}}$, we can prove only one direction of the following statement.

Proposition 1 *Given a protocol* $\mathcal{P}$*, its associated rewrite theory* $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, \phi_{\mathcal{P}}, E_{\mathcal{P}}, R_{\mathcal{P}})$*, its associated rewrite theory* $\mathcal{R}_{S\mathcal{P}} = (\Sigma_{S\mathcal{P}}, \phi_{S\mathcal{P}}, E_{S\mathcal{P}}, R_{S\mathcal{P}})$*, and two terms* SS *and* SS' *of sort* State*, if*

$$SS \ \leadsto^{\bullet}_{\sigma,\phi_{\mathcal{P}},R^{-1}_{\mathcal{P}},E_{\mathcal{P}}} \ SS'$$

then

$$mset(SS) \ \leadsto^{\bullet\ \{0,1\}}_{\sigma',\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \ mset(SS')$$

where $\sigma' \equiv \sigma{\downarrow}_{\mathcal{V}ar(mset(SS))}$.

Proof. By considering each possible protocol rule in $R_{\mathcal{P}}$; recall that they are applied in a backwards way.

- $[\ L \mid M^-, L'\ ]\ \&\ \{M{\in}\mathcal{I}, IK\} \rightarrow [\ L, M^- \mid L'\ ]\ \&\ \{M{\in}\mathcal{I}, IK\}$.
  Immediate, since $mset(SS') \equiv mset(SS)$, i.e., $\sigma(M) \in neg(\sigma(L)) \subseteq mset(SS)$ and $\sigma(M) \in mset(SS')$ due to its inclusion in the intruder knowledge.
- $[\ L \mid M^+, L'\ ]\ \&\ \{IK\} \rightarrow [\ L, M^+ \mid L'\ ]\ \&\ \{IK\}$.
  Immediate, since $mset(SS') \equiv mset(SS)$.

- $[\ L \mid M^+, L'\ ]\ \&\ \{M \notin \mathcal{I}, IK\} \rightarrow [\ L, M^+ \mid L'\ ]\ \&\ \{M \in \mathcal{I}, IK\}$.
  This rule accepts an output message u in a strand $[\ l_1, u^+ \mid l_2\ ]$ in SS'. By definition, there is a rule $u_1, \ldots, u_k \rightarrow m$ in $R_{S\mathcal{P}}$ such that $\sigma(u) =_{E_\mathcal{P}} \sigma(m)$ and $neg(l_1) = \{\sigma(u_1), \ldots, \sigma(u_k)\}$. Therefore, we have $mset(SS) \leadsto^{\bullet}_{\sigma', \phi_{S\mathcal{P}}, R^{-1}_{S\mathcal{P}}, E_{S\mathcal{P}}} mset(SS')$ such that $\sigma' = \sigma{\downarrow}_{\mathcal{V}ar(m)}$ and $mset(SS') = (mset(SS) - \{\sigma(u)\}) \cup \{\sigma(u_1), \ldots, \sigma(u_k)\}$.
- $[\ l_1 \mid u^+, l_2\ ]\ \&\ \{u \notin \mathcal{I}, IK\} \rightarrow \{u \in \mathcal{I}, IK\}$.
  This rule introduces a new strand $[\ l_1 \mid u^+, l_2\ ]$ in SS', i.e., $SS' = SS \cup \{\sigma([\ l_1 \mid u^+, l_2\ ])\}$. By definition, there is a rule $u_1, \ldots, u_k \rightarrow m$ in $R_{S\mathcal{P}}$ such that $\sigma(u) =_{E_\mathcal{P}} \sigma(m)$ and $neg(\sigma(l_1)) = \{\sigma(u_1), \ldots, \sigma(u_k)\}$. Therefore, we have $mset(SS) \leadsto^{\bullet}_{\sigma', \phi_{S\mathcal{P}}, R^{-1}_{S\mathcal{P}}, E_{S\mathcal{P}}} mset(SS')$ such that $\sigma' = \sigma{\downarrow}_{\mathcal{V}ar(m)}$ and $mset(SS') = (mset(SS) - \{\sigma(u)\}) \cup \{\sigma(u_1), \ldots, \sigma(u_k)\}$. $\square$

And the main theorem relating the narrowing relations associated to $\mathcal{R}_\mathcal{P}$ and $\mathcal{R}_{S\mathcal{P}}$ is the following one.

Theorem 4 (Correspondence) *Given a protocol $\mathcal{P}$, its associated rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, \phi_\mathcal{P}, E_\mathcal{P}, R_\mathcal{P})$, its associated rewrite theory $\mathcal{R}_{S\mathcal{P}} = (\Sigma_{S\mathcal{P}}, \phi_{S\mathcal{P}}, E_{S\mathcal{P}}, R_{S\mathcal{P}})$, a* State *term SS, and an initial* State *term SS_{ini}, if $SS \leadsto^{\bullet\,!}_{\sigma, \phi_\mathcal{P}, R^{-1}_\mathcal{P}, E_\mathcal{P}} SS_{ini}$ then $mset(SS) \leadsto^{\bullet\,!}_{\sigma', \phi_{S\mathcal{P}}, R^{-1}_{S\mathcal{P}}, E_{S\mathcal{P}}} \emptyset$ where $\sigma' \equiv \sigma{\downarrow}_{\mathcal{V}ar(mset(SS))}$.*

Proof. By induction on the number n of rewriting steps $SS \leadsto^{\bullet\,n}_{\sigma, \phi_\mathcal{P}, R^{-1}_\mathcal{P}, E_\mathcal{P}} SS_{ini}$.

- $(n = 0)$ Immediate, since SS is an initial State term and $mset(SS) = \emptyset$.
- $(n > 0)$ We have $SS \leadsto^{\bullet}_{\theta, \phi_\mathcal{P}, R^{-1}_\mathcal{P}, E_\mathcal{P}} SS' \leadsto^{\bullet\,n-1}_{\rho, \phi_\mathcal{P}, R^{-1}_\mathcal{P}, E_\mathcal{P}} SS_{ini}$ and $\sigma = \theta \circ \rho$. By Proposition 1, we have that there is θ' such that $mset(SS) \leadsto^{\bullet\,\{0,1\}}_{\theta', \phi_{S\mathcal{P}}, R^{-1}_{S\mathcal{P}}, E_{S\mathcal{P}}} mset(SS')$ and $\theta' = \theta{\downarrow}_{\mathcal{V}ar(mset(SS))}$. Then, by induction hypothesis, there is ρ' such that $mset(SS') \leadsto^{\bullet\,!}_{\rho', \phi_{S\mathcal{P}}, R^{-1}_{S\mathcal{P}}, E_{S\mathcal{P}}} \emptyset$ and $\rho' = \rho{\downarrow}_{\mathcal{V}ar(mset(SS'))}$, and therefore $mset(SS) \leadsto^{\bullet\,!}_{\sigma', \phi_{S\mathcal{P}}, R^{-1}_{S\mathcal{P}}, E_{S\mathcal{P}}} \emptyset$ where $\sigma' = \theta' \circ \rho'$. $\square$

6.3 Grammar Notation and Execution

This section introduces our grammar notation and explains how grammars are executed as rewrite theories.

Grammars are described by means of three basic kinds of constraints. To motivate our notation, we informally explain how the different constraints are generated.

(1) Seed terms may be described in two different ways, either as terms the

intruder is not expected to know, or as the result of performing operations on terms the intruder does not yet know. An example of the latter case is a term such as $pk(X, Y)$, where Y is not in the intruder knowledge. This last fact will be denoted by the constraint $Y \notin \mathcal{I}$ which is the same notation used for the intruder knowledge in Section 5.1. Therefore, facts of the form $(t \notin \mathcal{I})$ will make up our first type of constraints.

(2) Another possibility is that the intruder will be able to learn some instance of the seed term, i.e., the user was wrong when he/she believed the seed term was unknown to the intruder, and so we shall have to introduce some exceptions in our definition of the seed term, e.g., $(t_1 \not\preceq p_1)$, ..., $(t_k \not\preceq p_k)$. Facts $(t \not\preceq p)$ will be our second type of constraint.

(3) Once we have identified the seed term, we will use it to construct grammar rules stating that a term t is in the language if some subterm s of t is in the language, e.g., $(X; Z)$ is in $\mathcal{L}$ if Z is in $\mathcal{L}$. For example, we start out with a seed term saying that $pk(X, Y)$ is in the language if Y is not known by the intruder. Thus we start by trying to find the conditions under which the intruder knows $pk(X, Y)$. Applying rule $p7$ in Example 9 (in a backwards way and modulo encryption/decryption equations in E) gives us that this can be achieved if the intruder knows $sk(Z, pk(X, Y))$. The term $sk(Z, pk(X, Y))$ is not in the language, but we can make it so by introducing a rule that says that $sk(W, T)$ is in the language $\mathcal{L}$ if T is. This motivates the use of the third type of constraint, $(t \in \mathcal{L})$.

In what follows we define grammar rules and constraints more formally. Given a grammar G in the sequence $\mathcal{G}$, a grammar rule is written $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in \mathcal{L}$, with terms $t_1, \ldots, t_n$ of sort Msg and constraints $c_1, \ldots, c_k$ of sort Ctr. The intuitive idea of a rule $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in \mathcal{L}$ is that in order for any of the terms $t_1, \ldots, t_n$ to be in the language of the grammar G, say [9] $\mathcal{L}$, then the constraints $c_1, \ldots, c_k$ must be satisfied, i.e., $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in \mathcal{L}$ is understood as $t_1 \in \mathcal{L} \vee \cdots \vee t_n \in \mathcal{L}$ if $c_1 \wedge \cdots \wedge c_k$.

Definition 12 (Grammar Signature) *We define the sort* Ctr*, the signature* $\Sigma_{\mathcal{G}} = \Sigma \cup \{\_ \mapsto \_\} \cup \Sigma_{\mathsf{Ctr}}$ *where* $\_ \mapsto \_ : \mathsf{CtrSet} \times \mathsf{LCtr} \rightarrow \mathsf{GRule}$*, and the signature* $\Sigma_{\mathsf{Ctr}} = \{\ \_,\_\ ,\ \_ \in \mathcal{L}\ ,\ \_ \not\preceq \_\ ,\ \_ \notin \mathcal{I}\ \}$*. Within the sort* Ctr *we represent the three kinds of constraints by means of subsorts* $\mathsf{LCtr}\ \mathsf{DCtr}\ \mathsf{ICtr} < \mathsf{Ctr}$*:*

(i) constraints of the form $(t \notin \mathcal{I})$ *are constructed with the symbol*

$$(\_ \notin \mathcal{I}) : \mathsf{Msg} \rightarrow \mathsf{ICtr},$$

[9] We should write $t \in \mathcal{L}_G$ to denote that t is in the language of the grammar G. Since we always make explicit the grammar G that it is being used, we can simply write $t \in \mathcal{L}$.

(ii) constraints of the form $(t \not\preceq p)$ *are constructed with the symbol*

$$(\_ \not\preceq \_) : \mathsf{Msg} \times \mathsf{Msg} \rightarrow \mathsf{DCtr},$$

(iii) constraints of the form $(t \in \mathcal{L})$ *are constructed with the symbol*

$$(\_ \in \mathcal{L}) : \mathsf{MsgSet} \rightarrow \mathsf{LCtr}.$$

The operator $\_,\_ : \mathsf{CtrSet} \times \mathsf{CtrSet} \rightarrow \mathsf{CtrSet}$ *is a set union operator that is associative, commutative, and has identity* $\emptyset$*, and we assume that there is a subsort relation* $\mathsf{Ctr} < \mathsf{CtrSet}$*.*

Remark 4 *We assume that the pattern* p *in a constraint* $t \not\preceq p$ *always has fresh variables and therefore no substitution computed by our inference system can bind pattern* p *in a constraint* $t \not\preceq p$*.*

Example 10 *Consider the simplified protocol rules of Example 9. We generate all the intermediate and final grammars shown in Figure 4. Grammars* $G^0_{sd_1}, G^0_{sd_2}, G^0_{sd_3}, G^0_{sd_4}$ *are the seed terms provided for Example 3. Grammars* $G^!_{sd_1}, G^!_{sd_2}, G^!_{sd_3}, G^!_{sd_4}$ *represent the fixpoint of each* $G^0_{sd_i}$ *and each grammar rule is marked with a number* $gi.j$*, since they will be used in the rest of the paper.*

Informally speaking, (g1.1) implies that any expression of the form $pk(B, W)$ *is in the language* $\mathcal{L}$ *associated to the grammar* $G^!_{sd_1}$ *if the subterm at the position of the variable* W *is also in the language* $\mathcal{L}$*, i.e.,* $pk(B, W) \in \mathcal{L}$ *if* $W \in \mathcal{L}$*. Similarly, (g1.5) implies that a term of the form* $X; Y$ *is in* $\mathcal{L}$ *if the subterm at the position of* Y *is not in the intruder knowledge and is not of the form* $n(B, r)$*.*

6.3.1 Grammar Execution

To perform membership of a term in the language defined by a grammar, we associate to a grammar G a rewrite theory $\mathcal{R}_G$.

Definition 13 (Grammar Rewrite Theory) *Given a grammar* G*, we associate a rewrite theory* $\mathcal{R}_G = (\Sigma_\mathcal{G}, \phi_\mathcal{G}, E_\mathcal{G}, R_G)$*, where* $R_G = \{\mathcal{C} \rightarrow \mathcal{C}' \mid (\mathcal{C} \mapsto \mathcal{C}') \in G\}$*,* $E_\mathcal{G}$ *contains the protocol-specific equations* E *plus equations for associativity, commutativity and identity of the* $\_,\_$ *operator, and the frozenness function* $\phi_\mathcal{G}$ *is* $\phi_\mathcal{G}(\_ \mapsto \_) = \{2\}$*,* $\phi_\mathcal{G}(\_,\_) = \phi_\mathcal{G}(\_ \in \mathcal{L}) = \phi_\mathcal{G}(\_ \not\preceq \_) = \phi_\mathcal{G}(\_ \notin \mathcal{I}) = \emptyset$*, and* $\phi_\mathcal{G}(f) = \{1, \ldots, ar(f)\}$ *for the remaining* $f \in \Sigma$*.*

Note that the restrictions for Σ allow rewriting or narrowing steps only at the top of terms of sort Msg, which also implies that every term of sort Msg is strongly $\rightarrow_{\vec{E}}$-irreducible. Note also that R_G is a topmost theory.

$G^0_{sd_1}$
$Y \notin \mathcal{I} \mapsto (X;Y) \in \mathcal{L}$

$G^1_{sd_1}$
$Z \in \mathcal{L} \mapsto pk(c,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto pk(A, n(A,r); sk(B,Z)) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (X;Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (Z;Y) \in \mathcal{L}$
$Y \notin \mathcal{I} \mapsto (X;Y) \in \mathcal{L}$

$G^!_{sd_1}$
(g1.1) $Z \in \mathcal{L} \mapsto pk(B,Z) \in \mathcal{L}$
(g1.2) $Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
(g1.3) $Z \in \mathcal{L} \mapsto X;Z \in \mathcal{L}$
(g1.4) $Z \in \mathcal{L} \mapsto Z;Y \in \mathcal{L}$
(g1.5) $Y \notin \mathcal{I},\ Y \not\preceq n(B,r) \mapsto X;Y \in \mathcal{L}$

$G^0_{sd_2}$
$X \notin \mathcal{I} \mapsto (X;Y) \in \mathcal{L}$

$G^1_{sd_2}$
$Z \in \mathcal{L} \mapsto pk(c,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto pk(A, n(A,r); sk(B,Z)) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (X;Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (Z;Y) \in \mathcal{L}$
$X \notin \mathcal{I} \mapsto (X;Y) \in \mathcal{L}$

$G^2_{sd_2}$
$Z \in \mathcal{L} \mapsto pk(B,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (X;Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (Z;Y) \in \mathcal{L}$
$X \notin \mathcal{I} \mapsto (X;Y) \in \mathcal{L}$
$Z \notin \mathcal{I} \mapsto pk(B,c;Z) \in \mathcal{L}$

$G^3_{sd_2}$
$Z \in \mathcal{L} \mapsto pk(B,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (X;Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (Z;Y) \in \mathcal{L}$
$X \notin \mathcal{I},\ X \not\preceq n(B,r) \mapsto (X;Y) \in \mathcal{L}$
$Z \notin \mathcal{I},\ Z \not\preceq n(B''',r) \mapsto pk(B,B';Z) \in \mathcal{L}$
$Z \notin \mathcal{I},\ Z \not\preceq n(B,r) \mapsto (c;Z) \in \mathcal{L}$

$G^!_{sd_2}$
(g2.1) $Z \in \mathcal{L} \mapsto pk(B,Z) \in \mathcal{L}$
(g2.2) $Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
(g2.3) $Z \in \mathcal{L} \mapsto X;Z \in \mathcal{L}$
(g2.4) $Z \in \mathcal{L} \mapsto Z;Y \in \mathcal{L}$
(g2.5) $X \notin \mathcal{I},\ X \not\preceq n(B,r) \mapsto X;Y \in \mathcal{L}$
(g2.6) $Z \notin \mathcal{I},\ Z \not\preceq n(B',r) \mapsto B;Z \in \mathcal{L}$

$G^0_{sd_3}$
$Z \notin \mathcal{I} \mapsto pk(A,Z) \in \mathcal{L}$

$G^1_{sd_3}$
$Z \in \mathcal{L} \mapsto pk(c,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto pk(A, n(A,r); sk(B,Z)) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (X;Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (Z;Y) \in \mathcal{L}$
$Z \notin \mathcal{I},\ Z \not\preceq (Z'; n(A',r)) \mapsto pk(A,Z) \in \mathcal{L}$
$Z \notin \mathcal{I},\ Z \not\preceq (Z'; n(A',r')) \mapsto pk(A, n(A,r); Z) \in \mathcal{L}$

$G^!_{sd_3}$
(g3.1) $Z \in \mathcal{L} \mapsto pk(B,Z) \in \mathcal{L}$
(g3.2) $Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
(g3.3) $Z \in \mathcal{L} \mapsto X;Z \in \mathcal{L}$
(g3.4) $Z \in \mathcal{L} \mapsto Z;Y \in \mathcal{L}$
(g3.5) $Z \notin \mathcal{I},\ Z \not\preceq n(B,r),\ Z \not\preceq Z'; n(B',r') \mapsto pk(A,Z) \in \mathcal{L}$
(g3.6) $Z \notin \mathcal{I},\ Z \not\preceq n(B,r'),\ Z \not\preceq Z'; n(B',r'') \mapsto n(A,r); Z \in \mathcal{L}$

$G^0_{sd_4}$
$Z \notin \mathcal{I} \mapsto sk(A,Z) \in \mathcal{L}$

$G^1_{sd_4}$
$Z \in \mathcal{L} \mapsto pk(c,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto pk(A, n(A,r); sk(B,Z)) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (X;Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (Z;Y) \in \mathcal{L}$
$Z \notin \mathcal{I} \mapsto sk(A,Z) \in \mathcal{L}$

$G^!_{sd_4}$
(g4.1) $Z \in \mathcal{L} \mapsto pk(B,Z) \in \mathcal{L}$
(g4.2) $Z \in \mathcal{L} \mapsto sk(A,Z) \in \mathcal{L}$
(g4.3) $Z \in \mathcal{L} \mapsto X;Z \in \mathcal{L}$
(g4.4) $Z \in \mathcal{L} \mapsto Z;Y \in \mathcal{L}$
(g4.5) $Z \notin \mathcal{I} \mapsto sk(A,Z) \in \mathcal{L}$

Fig. 4. All the Grammars obtained for the Needham-Schroeder example

Recall that, since $\phi_{S\mathcal{P}}(\_ \mapsto \_) = \{1\}$ and $\phi_{\mathcal{G}}(\_ \mapsto \_) = \{2\}$, a term

$$c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in \mathcal{L}$$

has the first argument frozen when it is rewritten or narrowed using the abstract protocol rules $R_{S\mathcal{P}}$, but it has instead the second argument frozen when it is rewritten or narrowed using the rules R_G of a grammar G. Of course, the rules $R_{S\mathcal{P}}$ and R_G will be used in different contexts and for different purposes as explained later. Furthermore, note that in the Maude-NPA R_G is used in a backwards way, i.e., we will consider the relation $\rightarrow_{\phi_{\mathcal{G}}, R_G^{-1}, E_{\mathcal{G}}}$ for a grammar G, and we can use the relation $\rightarrow^!_{\phi_{\mathcal{G}}, R_G^{-1}, E_{\mathcal{G}}}$ without any risk of non-termination thanks to the shape of the rules in R_G, although the relation is non-confluent and several normal forms must be explored.

6.4 Membership in a Grammar's Language: Relations $\langle \mathcal{G}, \mathcal{C} \rangle \vdash (u{\in}\mathcal{L})$ and $c \sqsubseteq c'$

Often one needs to check whether a term t is in the language, say $\mathcal{L}$, generated by one grammar G in the sequence $\mathcal{G}$, i.e., $\mathcal{G} \vdash (t{\in}\mathcal{L})$. More generally, one needs to check whether $\langle \mathcal{G}, \mathcal{D} \rangle \vdash \mathcal{C}$, where $\mathcal{C}$ is the set of constraints of sort CtrSet that are being tested for satisfaction, and $\mathcal{D}$ is a set of premised conditions of sort CtrSet. This implies performing a form of backwards rewriting using the rewrite theory $\mathcal{R}_G$ associated to a grammar G in the sequence $\mathcal{G}$ together with some constraint cancellation between $\mathcal{C}$ and the premises $\mathcal{D}$ using some disequality reasoning capabilities for constraints of the form $t{\not\preceq}p$.

Definition 14 (Constraint Order) *Given $\mathcal{C}$ and $\mathcal{D}$ of sort CtrSet, we define the partial order relation $\mathcal{C} \sqsubseteq \mathcal{D}$ to hold iff for each $c_i \in \mathcal{C}$, there is a $d_j \in \mathcal{D}$ such that $c_i \sqsubseteq d_j$, and where $c \sqsubseteq c'$ is defined on individual constraints as follows (for u, t, s terms of sort Msg):*

$$(u{\in}\mathcal{L}) \sqsubseteq (u{\in}\mathcal{L})$$

$$(u{\notin}\mathcal{I}) \sqsubseteq (u{\notin}\mathcal{I})$$

$$(u{\not\preceq}p) \sqsubseteq (u{\not\preceq}p') \text{ if } \nexists\theta : u \equiv \theta(p) \text{ and } p \preceq p'$$

$$(u{\not\preceq}p) \sqsubseteq d_j \text{ for any } d_j \text{ of sort } \mathsf{Ctr} \text{ if } \nexists\theta : \theta(u) \equiv \theta(p)$$

Note that the order $\sqsubseteq$ is transitive by the transitivity of $\preceq$.

Definition 15 (Membership in a Grammar's Language) *Given $\mathcal{C}$ and $\mathcal{D}$ of sort CtrSet, we define*

$$\langle \mathcal{G}, \mathcal{D} \rangle \vdash \mathcal{C} \text{ iff there is } G \in \mathcal{G} \text{ s.t. } (\mathcal{C} \rightarrow^{!}_{\phi_{\mathcal{G}}, R_G^{-1}, E_{\mathcal{G}}} \mathcal{C}') \wedge (\mathcal{C}' \sqsubseteq \mathcal{D})$$

We write $\langle G, \mathcal{C}' \rangle \vdash \mathcal{C}$ instead of $\langle \mathcal{G}, \mathcal{C}' \rangle \vdash \mathcal{C}$ when we want to emphasize that a single grammar G (possibly not in $\mathcal{G}$) is used.

Recall that $\mathcal{V}ar(u) \cap \mathcal{V}ar(p) = \emptyset$ for each $u{\not\preceq}p$, see Remark 4. In the following examples, we write $t \rightarrow_{gi.j^{-1}} s$ to indicate that $t \rightarrow_{\phi_{\mathcal{G}}, R_G^{-1}, E_{\mathcal{G}}} s$ is rewritten using a grammar rule $gi.j$ in G.

Example 11 *Continuing Example 10, we give some examples of membership tests. For the membership test:*

$$\langle\ G^{!}_{sd_3},\ Y{\in}\mathcal{L}\ \rangle \vdash (pk(c, Y); M_2){\in}\mathcal{L}$$

we have

$$(pk(c,Y);M_2)\in\mathcal{L} \;\rightarrow_{g3.4^{-1}}\; pk(c,Y)\in\mathcal{L} \;\rightarrow_{g3.1^{-1}}\; Y\in\mathcal{L}$$

and since $Y\in\mathcal{L}$ *is already a premise, we have that* $(pk(c,Y);M_2)$ *is a member of* $G^{!}_{sd_3}$*. For the following test:*

$$\langle\ G^{!}_{sd_3},\ \emptyset\ \rangle \vdash (A;n(A,r))\not\preceq(Z';n(B,r'))$$

we have that it does not hold because indeed $A;n(A,r)\preceq Z';n(B,r')$*. And for the test*

$$\langle\ G^{!}_{sd_3},\ n(B,r)\notin\mathcal{I}\ \rangle \vdash pk(A,n(B,r))\in\mathcal{L}$$

we have

$$pk(A,n(B,r))\in\mathcal{L} \rightarrow_{g3.5^{-1}} n(B,r)\notin\mathcal{I}, n(B,r)\not\preceq n(B',r'), n(B,r)\not\preceq B''; n(B'',r'')$$

and these constraints are not satisfied, since $n(B,r)\preceq n(B',r')$*.*

The rest of this section studies a different property, called *satisfiability*, that we show is related to soundness of grammars. Informally, satisfiability of a set $\mathcal{C}$ of constraints means that they are still valid under instantiation. However, this is not true in general because of the constraints of the form $t\not\preceq p$ in $\mathcal{C}$, so we have to restrict satisfiability of $\mathcal{C}$ to satisfiability of another (greater) set $\mathcal{D}$ of constraints such that constraints of the form $\_\not\preceq\_$ in $\mathcal{D}$ imply constraints of the form $\_\not\preceq\_$ in $\mathcal{C}$. This is useful when proving that if a word belongs to the grammar of a language under some constraints $\mathcal{C}$, then any possible instantiation σ satisfying $\mathcal{C}$ (or a greater set $\mathcal{D}$) makes such word still a member. The following definition determines satisfiability of constraints of the form $t\not\preceq p$, which can be understood as that there is still some (future) substitution σ that can make true that $\sigma(t)\not\preceq p$, possibly the identity substitution.

Definition 16 ($\not\preceq$-Satisfiable Constraints) *Given a term* $\mathcal{C}$ *of sort* CtrSet*, we say that* $\mathcal{C}\equiv c_1,\ldots,c_k$ *is* $\not\preceq$-satisfiable *if for each* $c_i\equiv(u\not\preceq p)$*, we have that either: (i)* $\nexists\sigma_i:\sigma_i(u)\equiv\sigma_i(p)$*, or (ii)* $\exists\sigma_i:\sigma_i(u)\equiv\sigma_i(p)$ *and* $\nexists\theta_i:u\equiv\theta_i(p)$*.*

Example 12 *The constraint* $n(A,r)\not\preceq(Z';n(B,r'))$ *is* $\not\preceq$*-satisfiable, since the root symbol of both terms is different. The constraint* $(A;n(A,r))\not\preceq(Z';n(B,r'))$ *is not* $\not\preceq$*-satisfiable, since there is* σ *s.t.* $(A;n(A,r))\equiv\sigma(Z';n(B,r'))$*. However, the more generic constraint* $Y\not\preceq(Z';n(B,r'))$ *is* $\not\preceq$*-satisfiable, since there is no* σ *s.t.* $Y\equiv\sigma(Z';n(B,r'))$ *even though there is* θ *s.t.* $\theta(Y)\equiv\theta(Z';n(B,r'))$*.*

Corollary 1 *Given two terms* $\mathcal{C}$ *and* $\mathcal{D}$ *of sort* CtrSet *if* $(\mathcal{C}\sqsubseteq\mathcal{D})$ *and* $\mathcal{D}$ *is* $\not\preceq$*-satisfiable, then* $\mathcal{C}$ *is* $\not\preceq$*-satisfiable.*

Proof. The only relevant type of constraints is $u\not\preceq t$. If $(\mathcal{C}\sqsubseteq\mathcal{D})$, then for each $(u\not\preceq t)$ in $\mathcal{C}$, either (i) $\nexists\theta:\theta(u)\equiv\theta(t)$ and $(u\not\preceq t)$ is $\not\preceq$-satisfiable independently

of $\mathcal{D}$, or (ii) $(u \not\preceq t) \sqsubseteq (u \not\preceq s)$ for $(u \not\preceq s)$ in $\mathcal{D}$ such that $\exists \sigma : \sigma(u) \equiv \sigma(s)$ and $\nexists \theta : u \equiv \theta(s)$, and then, since $t \preceq s$, $\exists \sigma' : \sigma'(u) \equiv \sigma'(t)$, and $\nexists \theta' : u \equiv \theta'(t)$. $\square$

Corollary 2 *Given a term $\mathcal{C}$ of sort* CtrSet *and a substitution σ, if $\sigma(\mathcal{C})$ is $\not\preceq$-satisfiable, then $\mathcal{C}$ is $\not\preceq$-satisfiable.*

Proof. By Corollary 1, since $\sigma(\mathcal{C}) \sqsubseteq \mathcal{C}$ for constraints of the form $u \not\preceq p$ due to the restriction $\mathcal{V}ar(p) \not\subseteq \mathcal{D}om(\sigma)$. $\square$

The following results prove that membership is closed under substitution, provided some conditions hold.

Lemma 1 (Constraint Substitution Closure) *Given two terms $\mathcal{C}$ and $\mathcal{D}$ of sort* CtrSet *and a substitution σ, if $(\mathcal{C} \sqsubseteq \mathcal{D})$ and $\sigma(\mathcal{D})$ is $\not\preceq$-satisfiable, then $\sigma(\mathcal{C}) \sqsubseteq \sigma(\mathcal{D})$.*

Proof. Recall that, by Remark 4, the substitution σ cannot bind variables of the pattern p in a constraint $t \not\preceq p$, and thus we don't apply σ to the pattern p in a constraint $t \not\preceq p$. By considering each case associated to $(\mathcal{C} \sqsubseteq \mathcal{D})$:

- In the case "$(u{\in}\mathcal{L}) \sqsubseteq (u{\in}\mathcal{L})$", we clearly have that for each σ, $(\sigma(u){\in}\mathcal{L}) \sqsubseteq (\sigma(u){\in}\mathcal{L})$.
- In the case "$(u{\notin}\mathcal{I}) \sqsubseteq (u{\notin}\mathcal{I})$", we also have that for each σ, $(\sigma(u){\notin}\mathcal{I}) \sqsubseteq (\sigma(u){\notin}\mathcal{I})$.
- In the case "$(u \not\preceq t) \sqsubseteq (u \not\preceq s)$ if $\nexists \theta : u \equiv \theta(t)$ and $t \preceq s$", we have that $\nexists \theta : \sigma(u) \equiv \theta(t)$, since $t \preceq s$ and $\sigma(u) \not\preceq s$ is $\not\preceq$-satisfiable (i.e., $\nexists \theta : \sigma(u) \equiv \theta(s)$).
- In the case "$(u \not\preceq t) \sqsubseteq d_j$ for any d_j of sort Ctr if $\nexists \theta : \theta(u) \equiv \theta(t)$", we also have that for each σ, $(\sigma(u) \not\preceq t) \sqsubseteq d_j$, since $\nexists \theta : \theta(\sigma(u)) \equiv \theta(t)$. $\square$

Theorem 5 (Membership Substitution Closure) *Given a grammar G and two terms $\mathcal{C}$ and $\mathcal{D}$ of sort* CtrSet*, if $\langle G, \mathcal{D} \rangle \vdash \mathcal{C}$, then for each substitution σ such that $\sigma(\mathcal{D})$ is $\not\preceq$-satisfiable, we have $\langle G, \sigma(\mathcal{D}) \rangle \vdash \sigma(\mathcal{C})$.*

Proof. $\langle G, \mathcal{D} \rangle \vdash \mathcal{C}$ implies that there is $\mathcal{C}'$ s.t. $\mathcal{C} \rightarrow^{!}_{\phi_\mathcal{G}, R_G^{-1}, E_\mathcal{G}} \mathcal{C}'$, and $\mathcal{C}' \sqsubseteq \mathcal{D}$. Therefore, $\sigma(\mathcal{C}) \rightarrow^{!}_{\phi_\mathcal{G}, R_G^{-1}, E_\mathcal{G}} \sigma(\mathcal{C}')$, since no extra variables are introduced by the rewriting relation $\rightarrow_{\phi_\mathcal{G}, R_G^{-1}, E_\mathcal{G}}$ except those in patterns of constraints of the form $t \not\preceq p$, but they are not bound by σ by definition. Finally, by Lemma 1, $\sigma(\mathcal{C}') \sqsubseteq \sigma(\mathcal{D})$. $\square$

The following result proves that membership is implied by a greater set of premises.

Theorem 6 (Membership Implication) *Given a grammar G and three terms $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{D}'$ of sort* CtrSet*, if $\langle G, \mathcal{D} \rangle \vdash \mathcal{C}$ and $\mathcal{D} \sqsubseteq \mathcal{D}'$, then $\langle G, \mathcal{D}' \rangle \vdash \mathcal{C}$.*

Proof. $\langle G, \mathcal{D}\rangle \vdash \mathcal{C}$ implies that there is $\mathcal{C}'$ s.t. $\mathcal{C} \rightarrow^{!}_{\phi_{\mathcal{G}}, R_G^{-1}, E_{\mathcal{G}}} \mathcal{C}'$, and $\mathcal{C}' \sqsubseteq \mathcal{D}$. Thus, $\mathcal{C}' \sqsubseteq \mathcal{D}'$. □

6.5 The Grammar Generation Inference System

In this section, we formally describe how grammars are generated. To motivate the procedure, we briefly rephrase the informal description of how the Maude-NPA generates languages given at Section 6.1, but using the notation and operators formally described below and using Example 2 and the Needham-Schroeder Example. We depict in Figure 5 the dependencies between the different operators.

The Maude-NPA starts the grammar generation process with a seed term sd_i. This seed term defines an initial grammar $G^0_{sd_i}$ stating that the seed term is in the language, i.e., the user believes it is unreachable for the intruder.

Definition 17 (Seed Term) *A seed term, i.e., the first grammar rule provided as the seed of a grammar to be generated, is either of the form $t|_q \notin \mathcal{I} \mapsto t \in \mathcal{L}$ where $t|_q$ is a variable of t, or of the form $\emptyset \mapsto t \in \mathcal{L}$.*

Example 13 *In Example 2, the unique seed term sd_1 is represented by the initial grammar $G^0_{sd_1} \equiv \emptyset \mapsto m \in \mathcal{L}$, denoting that the intruder cannot learn the message m.*

Example 14 *For the Needham-Schroeder Example, the seed terms are the grammars $G^0_{sd_1} \equiv Y \notin \mathcal{I} \mapsto (X;Y) \in \mathcal{L}$, $G^0_{sd_2} \equiv X \notin \mathcal{I} \mapsto (X;Y) \in \mathcal{L}$, $G^0_{sd_3} \equiv Z \notin \mathcal{I} \mapsto pk(A,Z) \in \mathcal{L}$, and $G^0_{sd_4} \equiv Z \notin \mathcal{I} \mapsto sk(A,Z) \in \mathcal{L}$, all shown in Figure 4. For instance, the initial grammar $G^0_{sd_1}$ denotes all the terms $(t_1;t_2)$ such that we know that t_2 is not known by the intruder at the current state in a protocol run.*

The Maude-NPA mechanism for generating languages is represented by the operator

$$G^k_j \Rrightarrow_{\mathcal{P}, G^k_j, s} G^{k+1}_j$$

where G^{k+1}_j is the new grammar generated from G^k_j. The fixpoint of $G^0_{sd_i}$ w.r.t. the operator $G^k_j \Rrightarrow_{\mathcal{P}, G^k_j, s} G^{k+1}_j$ is denoted by $G^!_{sd_i}$. An attempt to transform a grammar $G^k_{sd_i}$ that is not in its fixpoint form is performed, resulting in a new grammar $G^{k+1}_{sd_i}$ or failing to produce a new grammar rule, which implies discarding such grammar $G^k_{sd_i}$. The fixpoint of all the seed terms for which we have obtained a fixpoint are kept in a grammar sequence $\mathcal{G} = \langle G^!_{sd_{i_1}}, \ldots, G^!_{sd_{i_m}} \rangle$, where $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$. Note that the word s in the operator $G^k_j \Rrightarrow_{\mathcal{P}, G^k_j, s} G^{k+1}_j$ is a global strategy parameter that can be instantiated to either S1 or S2 and that determines how new grammar rules are

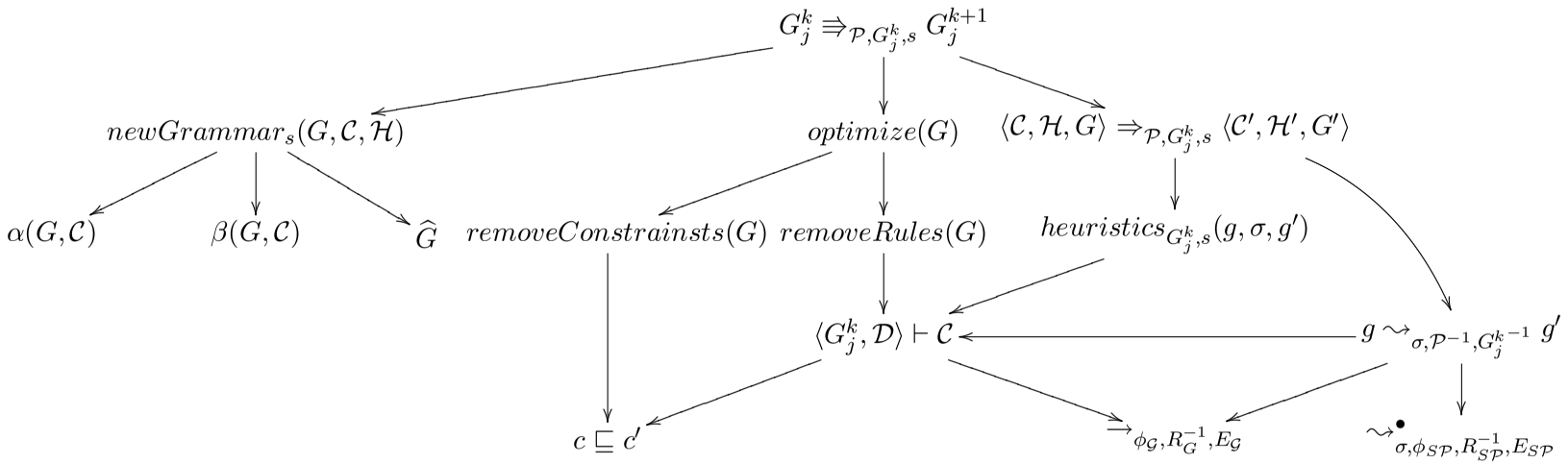

Fig. 5. Dependencies between relations and operators in the Grammar Generation Phase

generated; see Section 6.5.6 for details. This strategy parameter is fixed during the whole generation of a grammar. All the generated grammars, i.e., seed terms, intermediate grammars, and the fixpoint grammars, are characterized as follows.

Definition 18 (General Grammar Rule Shape) *A grammar rule g of a grammar G is always of one of the following forms:*

$$\textit{(positive)} \qquad t{\not\preceq}s_1, \ldots, t{\not\preceq}s_n \mapsto t{\in}\mathcal{L}\ (n \geq 0)$$

$$\textit{(negative-S1)}\ t|_q{\notin}\mathcal{I}, t|_q{\not\preceq}s_1, \ldots, t|_q{\not\preceq}s_n \mapsto t{\in}\mathcal{L}\ (t|_q{\in}\mathcal{X},\ n \geq 0)$$

$$\textit{(negative-S2)}\quad t|_q{\notin}\mathcal{I}, t{\not\preceq}s_1, \ldots, t{\not\preceq}s_n \mapsto t{\in}\mathcal{L}\ (t|_q{\in}\mathcal{X},\ n \geq 0)$$

$$\textit{(recursive)} \qquad t|_q{\in}\mathcal{L} \mapsto t{\in}\mathcal{L}\ (t|_q{\in}\mathcal{X})$$

Intuitively, we can have the following grammar productions:

Positive Messages in the language of the grammar that do not need extra requirements about negative information of the intruder knowledge, such as $\emptyset \mapsto m{\in}\mathcal{L}$ of Example 13. They originated as a seed term.

Negative Messages in the language of the grammar that require negative information of the intruder knowledge, such as $Y{\notin}\mathcal{I} \mapsto (X;Y){\in}\mathcal{L}$ of Figure 4. They originated also as a seed term.

Constrained Messages of the positive and negative previous forms but that include some syntactic restriction on the message, where:

- (Positive) For messages of the positive form, only restrictions of the form $(t{\not\preceq}s)$ are included, such as $pk(A,Z){\not\preceq}pk(A', n(B,r)) \mapsto pk(A,Z){\in}\mathcal{L}$. They are generated only by applications of the generation strategy S2.
- (Negative) For messages of the negative form, restrictions of either the form $(t|_p{\not\preceq}s)$ or the form $(t{\not\preceq}s)$ are included.
  - (Negative-S1) For strategy S1, constraints $(t|_p{\not\preceq}s)$ are included, such as $Z{\notin}\mathcal{I},\ Z{\not\preceq}n(B,r) \mapsto pk(A,Z){\in}\mathcal{L}$.
  - (Negative-S2) For strategy S2, constraints $(t{\not\preceq}s)$ are included, such as $Z{\notin}\mathcal{I},\ pk(A,Z){\not\preceq}pk(A', n(B,r)) \mapsto pk(A,Z){\in}\mathcal{L}$.

Recursive Messages that involve a recursive membership call within the grammar, such as $Z{\in}\mathcal{L} \mapsto pk(B,Z){\in}\mathcal{L}$ of Figure 4.

Example 15 *For the Needham-Schroeder Example, all the grammars shown in Figure 4 are characterized as described by Definition 18.*

The transformation of a grammar G_j^k into G_j^{k+1}, i.e., $G_j^k \Rrightarrow_{\mathcal{P},G_j^k,s} G_j^{k+1}$, is performed in two steps. First, we obtain the fixpoint of the operator

$$\langle \mathcal{C}, \mathcal{H}, G\rangle \Rightarrow_{\mathcal{P},G_j^k,s} \langle \mathcal{C}', \mathcal{H}', G'\rangle$$

which starts with $\langle \emptyset, \emptyset, G_j^k \rangle$ and ends with $\langle \mathcal{C}'', \mathcal{H}'', \emptyset \rangle$, where $\mathcal{C}''$ is a set of constraints of the form $t \not\preceq p$ and $\mathcal{H}''$ is a set of grammar rules. Note that if $\mathcal{C}'' \equiv \emptyset$ and $\mathcal{H}'' \equiv \emptyset$, then we say that grammar G_j^k has reached its fixpoint. But if $\langle \mathcal{C}, \mathcal{H}, G \rangle$ is a normal form w.r.t. the relation $\Rightarrow_{\mathcal{P}, G_j^k, s}$ and $G \neq \emptyset$, then we say that grammar G_j^k failed and is removed.

Example 16 *For the Needham-Schroeder Example and the grammar $G^0_{sd_3}$ of Figure 4, containing only the grammar rule $Z \notin \mathcal{I} \mapsto pk(A, Z) \in \mathcal{L}$, we have the transformation step $\langle \emptyset, \emptyset, G^0_{sd_3} \rangle \Rightarrow_{\mathcal{P}, G^0_{sd_3}, S1} \langle \mathcal{C}, \mathcal{H}, \emptyset \rangle$ where*

$$\mathcal{C} = \{\ Z \not\preceq A; n(A, r),\ Z \not\preceq Z'; n(A, r)\ \}$$

and

$$\begin{aligned} \mathcal{H} = \{ & Z \in \mathcal{L} \mapsto pk(c, Z) \in \mathcal{L}, \\ & Z \in \mathcal{L} \mapsto pk(A, n(A, r); sk(B, Z)) \in \mathcal{L}, \\ & Z \in \mathcal{L} \mapsto sk(A, Z) \in \mathcal{L}, \\ & Z \in \mathcal{L} \mapsto Z; Y \in \mathcal{L}, \\ & Z \in \mathcal{L} \mapsto X; Z \in \mathcal{L}, \\ & Z \notin \mathcal{I} \mapsto pk(A, n(A, r); Z) \in \mathcal{L}\} \end{aligned}$$

The generation of constraints $\mathcal{C}$ and rules $\mathcal{H}$ is explained in the following.

Second, we use the operator

$$newGrammar_s(G_j^k, \mathcal{C}'', \mathcal{H}'')$$

to combine these sets $\mathcal{C}''$ and $\mathcal{H}''$ with the previous grammar G_j^k to produce a preliminary version of G_j^{k+1} and optimize this preliminary version of G_j^{k+1} using the operator $optimize(G)$, which removes redundant constraints of the form $t \not\preceq p$ and redundant grammar rules.

Example 17 *Continuing Example 16. The combination of $\mathcal{C}$ and $\mathcal{H}$ into $G^0_{sd_3}$, i.e., $newGrammar_{S1}(G^0_{sd_3}, \mathcal{C}, \mathcal{H})$, provides the following grammar*

$Z \in \mathcal{L} \mapsto pk(c, Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto pk(A, n(A, r); sk(B, Z)) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto sk(A, Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (X; Z) \in \mathcal{L}$
$Z \in \mathcal{L} \mapsto (Z; Y) \in \mathcal{L}$
$Z \notin \mathcal{I},\ Z \not\preceq (A'; n(A', r)),\ Z \not\preceq (Z'; n(A'', r')) \mapsto pk(A, Z) \in \mathcal{L}$
$Z \notin \mathcal{I},\ Z \not\preceq (A'; n(A', r')),\ Z \not\preceq (Z'; n(A'', r'')) \mapsto pk(A, n(A, r); Z) \in \mathcal{L}$

and the optimization of $newGrammar_{S1}(G^0_{sd_3}, \mathcal{C}, \mathcal{H})$ provides the grammar $G^1_{sd_3}$ of Figure 4.

The operator $\langle \mathcal{C}, \mathcal{H}, G \rangle \Rightarrow_{\mathcal{P}, G_j^k, s} \langle \mathcal{C}', \mathcal{H}', G' \rangle$ includes the three stages mentioned before: (i) the term generation stage, (ii) the rule verification stage,

and (iii) the rule generation stage.

The *term generation* stage is performed by the backwards narrowing operator

$$g \rightsquigarrow_{\sigma,\mathcal{P}^{-1},G_j^{k-1}} g'$$

which takes a grammar rule g and returns a grammar rule g' representing a preceding state in the protocol. We call g' a pre-grammar rule. The pre-grammar rule g' is computed by the backwards narrowing relation $g \rightsquigarrow^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} g'$, which is just a slightly modified version of $\rightsquigarrow_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}}$ (see Appendix A).

Example 18 *For the Needham-Schroeder Example and the grammar $G^0_{sd_3}$ of Figure 4, the term generation stage using the relation $\rightsquigarrow^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}}$ provides the following pre-grammar rules w.r.t. the simplified protocol rules $R_{S\mathcal{P}}$ of Example 9:*

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{[Z/(A';n(A',r))],\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad (A';n(A',r))\notin\mathcal{I} \mapsto \emptyset\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto pk(A',n(A',r);Z)\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto pk(A',n(A',r);sk(B,pk(A,Z)))\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{[Z/(Z';n(B,r))],\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad (Z';n(B,r))\notin\mathcal{I} \mapsto pk(B,A;Z')\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto pk(A,Z);M_2\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto M_1;pk(A,Z)\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{[A/c],\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto pk(c,Z)\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto pk(c,pk(A,Z))\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto Z\in\mathcal{L}$$

$$Z\notin\mathcal{I} \mapsto pk(A,Z)\in\mathcal{L} \quad \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad Z\notin\mathcal{I} \mapsto sk(A',pk(A,Z))\in\mathcal{L}$$

The *rule verification* stage is embedded into the $g \rightsquigarrow_{\sigma,\mathcal{P}^{-1},G_j^{k-1}} g'$ relation as a test called *G-expandable*. That is, the Maude-NPA takes the grammar rule $g \equiv \mathcal{C} \mapsto t\in\mathcal{L}$, computes each path preceding message t in the protocol using $t \rightsquigarrow^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} s_1,\ldots,s_n$, and tests several things for each s_i, for instance whether $s_i\notin\mathcal{I}$ appears in $\mathcal{C}$ or whether $\langle G,\sigma(\mathcal{C})\rangle \vdash (s_i\in\mathcal{L})$ for each s_i. In such case, the preceding state represented by $s_1,\ldots,s_n$ is not G-expandable and this path is discarded. Otherwise, we apply the rule generation stage to this grammar rule $\sigma(\mathcal{C}) \mapsto (s_1,\ldots,s_n)\in\mathcal{L}$.

Example 19 *In Example 2, we obtain* $\emptyset \mapsto m{\in}\mathcal{L} \rightsquigarrow_{\sigma,\mathcal{P}^{-1},G^{-1}} \emptyset \mapsto e(k,m){\in}\mathcal{L}$ *as the backwards narrowing step* $m \rightsquigarrow^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} e(k,m)$ *using the abstract version* $R_{S\mathcal{P}}$ *of the protocol rules. We test that* $\emptyset \mapsto e(k,m){\in}\mathcal{L}$ *is* G_0*-expandable by checking that* $\langle G_0, \emptyset\rangle \vdash (e(k,m){\in}\mathcal{L})$ *returns false. And then, apply the rule generation stage to* $\emptyset \mapsto e(k,m){\in}\mathcal{L}$.

Example 20 *Continuing Example 18, the following backwards narrowing steps are not* $G^0_{sd_3}$*-expandable and are discarded.*

$$Z{\notin}\mathcal{I} \mapsto pk(A,Z){\in}\mathcal{L} \rightsquigarrow^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} Z{\notin}\mathcal{I} \mapsto Z{\in}\mathcal{L}$$
$$Z{\notin}\mathcal{I} \mapsto pk(A,Z){\in}\mathcal{L} \rightsquigarrow^{\bullet}_{[A/c],\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} Z{\notin}\mathcal{I} \mapsto pk(c,Z){\in}\mathcal{L}$$

The *rule generation* stage is performed by the operator $heuristics_{G^k_j,s}(g,\sigma,g') = \langle \mathcal{C}, \mathcal{H}\rangle$, which applies several heuristics to add either new constraints $\mathcal{C}$ of the form $t{\not\trianglelefteq}p$ or new grammar rules $\mathcal{H}$.

Example 21 *In Example 2, one of the heuristics generates the grammar rule* $Y{\in}\mathcal{L} \mapsto e(k,Y){\in}\mathcal{L}$ *from* $\emptyset \mapsto e(k,m){\in}\mathcal{L}$.

Example 22 *Continuing Example 20, for each* $G^0_{sd_3}$*-expandable pre-grammar rule the heuristics generate the following constraint or new grammar rule, where* $g \equiv Z{\notin}\mathcal{I} \mapsto pk(A,Z){\in}\mathcal{L}$*:*

$$\sigma_1 \equiv [Z/(A';n(A',r))] \quad g_1 \equiv (A';n(A',r)){\notin}\mathcal{I} \mapsto \emptyset{\in}\mathcal{L}$$
$$heuristics_{G^0_{sd_3},S1}(g,\sigma_1,g_1) = \{\ Z{\not\trianglelefteq}(A';n(A',r)),\ \emptyset\ \}$$

$$\sigma_2 \equiv id \quad g_2 \equiv Z{\notin}\mathcal{I} \mapsto pk(A',n(A',r);Z){\in}\mathcal{L}$$
$$heuristics_{G^0_{sd_3},S1}(g,\sigma_2,g_2) = \{\ \emptyset,\ Z{\notin}\mathcal{I} \mapsto pk(A',n(A',r);Z){\in}\mathcal{L}\ \}$$

$$\sigma_3 \equiv id \quad g_3 \equiv Z{\notin}\mathcal{I} \mapsto pk(A',n(A',r);sk(B,pk(A,Z))){\in}\mathcal{L}$$
$$heuristics_{G^0_{sd_3},S1}(g,\sigma_3,g_3) = \{\emptyset, Y{\in}\mathcal{L} \mapsto pk(A',n(A',r);sk(B,Y)){\in}\mathcal{L}\}$$

$$\sigma_4 \equiv [Z/(Z';n(B,r))] \quad g_4 \equiv (Z';n(B,r)){\notin}\mathcal{I} \mapsto pk(B,A;Z'){\in}\mathcal{L}$$
$$heuristics_{G^0_{sd_3},S1}(g,\sigma_4,g_4) = \{\ Z{\not\trianglelefteq}(Z';n(B,r)),\ \emptyset\ \}$$

$$\sigma_5 \equiv id \quad g_5 \equiv Z{\notin}\mathcal{I} \mapsto pk(A,Z);M_2{\in}\mathcal{L}$$
$$heuristics_{G^0_{sd_3},S1}(g,\sigma_5,g_5) = \{\ \emptyset,\ Y{\in}\mathcal{L} \mapsto Y;M_2{\in}\mathcal{L}\ \}$$

$$\sigma_6 \equiv id \quad g_6 \equiv Z{\notin}\mathcal{I} \mapsto M_1;pk(A,Z){\in}\mathcal{L}$$
$$heuristics_{G^0_{sd_3},S1}(g,\sigma_6,g_6) = \{\ \emptyset,\ Y{\in}\mathcal{L} \mapsto M_1;Y{\in}\mathcal{L}\ \}$$

$$\sigma_7 \equiv id \quad g_7 \equiv Z{\notin}\mathcal{I} \mapsto pk(c,pk(A,Z)){\in}\mathcal{L}$$
$$heuristics_{G^0_{sd_3},S1}(g,\sigma_7,g_7) = \{\ \emptyset,\ Y{\in}\mathcal{L} \mapsto pk(c,Y){\in}\mathcal{L}\ \}$$

$$\sigma_8 \equiv id \quad g_8 \equiv Z \notin \mathcal{I} \mapsto sk(A', pk(A, Z)) \in \mathcal{L}$$
$$heuristics_{G^0_{sd_3}, S1}(g, \sigma_8, g_8) = \{\ \emptyset,\ Y \in \mathcal{L} \mapsto sk(A', Y) \in \mathcal{L}\ \}$$

that are exactly the constraints and new rules shown in Example 16.

Recall that the entire grammar generation procedure is iterated by the operator $\Rightarrow_{\mathcal{P}, G^k_j, s}$ until it reaches a fixpoint. If the term generation stage (i.e., $g \rightsquigarrow_{\sigma, \mathcal{P}^{-1}, G^{k-1}_j} g'$) does not produce any new state or the rule verification stage (i.e., the *G-expandable* test) cuts all the generated states, then we say that grammar G^k_j has reached its fixpoint. Otherwise, if the heuristics were not able to generate either new constraints or new rules, then the operator $\Rightarrow_{\mathcal{P}, G^k_j, s}$ fails and the grammar G^k_j is discarded.

Note that we cannot guarantee at the moment whether the grammar generation process might terminate with success, terminate with failure, or not terminate. Examples 29 and 30 below motivate when the grammar generation process might terminate with failure or do not terminate. But even if the grammar generation process terminates for each seed term (with success or failure), we cannot detect whether we have a finite search space using the protocol rewrite theory $R_{\mathcal{P}}$. A detailed study of the conditions on the protocol and the seed terms to have a terminating grammar generation process and a finite search space is left for future work. However, practical experience shows that the grammar generation process terminates (with success or failure) for many protocols and for some of them we have a finite search space.

In the following, we formally define the relations and operators involved in grammar generation in a top-down (almost) left-to-right order following Figure 5.

6.5.1 Generating a New Grammar: The Relation $G^k_j \Rrightarrow_{\mathcal{P}, G^k_j, s} G^{k+1}_j$

We generate a new grammar G^{k+1}_j from a grammar G^k_j using the main grammar transformation relation $\Rrightarrow_{\mathcal{P}, G^k_j, s}$.

Definition 19 (Generating a New Grammar) *Given a grammar G^k_j, the set of strand $\mathcal{P}$, and a generation strategy s, we generate a new grammar G^{k+1}_j as follows:*

$$G^k_j \Rrightarrow_{\mathcal{P}, G^k_j, s} G^{k+1}_j \ \text{if}\ \langle \emptyset, \emptyset, G^k_j \rangle \Rightarrow^!_{\mathcal{P}, G^k_j, s} \langle \mathcal{C}, \mathcal{H}, \emptyset \rangle,$$
$$G^{k+1}_j = optimize(newGrammar_s(G^k_j, \mathcal{C}, \mathcal{H})),$$
$$\text{and}\ G^{k+1}_j \not\equiv G^k_j$$

Recall that $\langle \mathcal{C}, \mathcal{H}, G\rangle \Rightarrow_{\mathcal{P},G_j^k,s} \langle \mathcal{C}', \mathcal{H}', G'\rangle$ produces a set of constraints and grammar rules for G_j^k, the operator $newGrammar_s(G_j^k, \mathcal{C}, \mathcal{H})$ combines $\mathcal{C}$ and $\mathcal{H}$ into G_j^k to create a new grammar, and the operator $optimize(G)$ removes redundant information. The word s is a global strategy parameter that can be instantiated to either $S1$ or $S2$ and is only relevant for the heuristics application, see Section 6.5.6 below.

Example 23 *The global grammar generation process for the Needham-Schroeder Example 3 using the relation $G_j^k \Rrightarrow_{\mathcal{P},G_j^k,s} G_j^{k+1}$ is as follows, where the grammars used were given in Figure 4:*

$$G^0_{sd_1} \Rrightarrow_{\mathcal{P},G^0_{sd_1},S1} G^1_{sd_1} \Rrightarrow_{\mathcal{P},G^1_{sd_1},S1} G^!_{sd_1}$$
$$G^0_{sd_2} \Rrightarrow_{\mathcal{P},G^0_{sd_2},S1} G^1_{sd_2} \Rrightarrow_{\mathcal{P},G^1_{sd_2},S1} G^2_{sd_2} \Rrightarrow_{\mathcal{P},G^2_{sd_2},S1} G^3_{sd_2} \Rrightarrow_{\mathcal{P},G^3_{sd_2},S1} G^!_{sd_2}$$
$$G^0_{sd_3} \Rrightarrow_{\mathcal{P},G^0_{sd_3},S1} G^1_{sd_3} \Rrightarrow_{\mathcal{P},G^1_{sd_3},S1} G^!_{sd_3}$$
$$G^0_{sd_4} \Rrightarrow_{\mathcal{P},G^0_{sd_4},S1} G^1_{sd_4} \Rrightarrow_{\mathcal{P},G^1_{sd_4},S1} G^!_{sd_4}$$

6.5.2 Adding New Grammar Rules: The Operator $newGrammar_s(G, \mathcal{C}, \mathcal{H})$

The intuition behind this operator is that the new grammar rules $\mathcal{H}$ can be added to the previous grammar G without problems, since they extend the language of the grammar, but the constraints $\mathcal{C}$ of the form $t \not\trianglelefteq s$ pose a problem, since they restrict the language of the grammar. Therefore, we must add those constraints $\mathcal{C}$ to the rules in $G \cup \mathcal{H}$, but only to those rules without a constraint of the form $t \in \mathcal{L}$, which are the ultimate rules used for testing membership. For adding these constraints, we must consider the strategy used for generating $\mathcal{C}$ and $\mathcal{H}$, i.e., strategy $S1$ or $S2$. For strategy $S1$, we must add constraints $\_ \not\trianglelefteq \_$ only to those rules with a constraint of the form $Y \notin \mathcal{I}$ (called negative in Definition 18), since strategy $S1$ uses such kind of rules, and adapt each constraint $\_ \not\trianglelefteq \_$ to the variable Y. For strategy $S2$, we must add constraints $\_ \not\trianglelefteq \_$ to rules with and without a constraint of the form $Y \notin \mathcal{I}$ but each constraint $\_ \not\trianglelefteq \_$ must be adapted to the term in the right-hand side of the grammar rule.

Definition 20 (Adding New Grammar Rules) *Given a set G of grammar rules, a set $\mathcal{C}$ of constraints of the form $t \not\trianglelefteq s$, and a set $\mathcal{H}$ of grammar rules, the function $newGrammar_s(G, \mathcal{C}, \mathcal{H})$ joins the set of grammar rules in G and the new rules in $\mathcal{H}$ with the constraints $\mathcal{C}$:*

$$newGrammar_{S1}(G, \mathcal{C}, \mathcal{H}) = \alpha(G, \mathcal{C}) \cup \alpha(\mathcal{H}, \mathcal{C}) \cup \widehat{G} \cup \widehat{\mathcal{H}}$$
$$newGrammar_{S2}(G, \mathcal{C}, \mathcal{H}) = \beta(G, \mathcal{C}) \cup \widehat{G} \cup \widehat{\mathcal{H}}$$

The operator $\alpha(G, \mathcal{C})$ adds constraints $\mathcal{C}$ to grammar rules in G of the form

negative-S1:

$$\alpha(G,\mathcal{C}) = \{c_1,\ldots,c_k,\theta_\alpha(\mathcal{C}) \mapsto t{\in}\mathcal{L} \mid (c_1,\ldots,c_k \mapsto t{\in}\mathcal{L}) \in G \\ \wedge\ \exists! c_i, \exists Y \in \mathcal{X} : c_i \equiv (Y{\notin}\mathcal{I})\}$$

where the substitution θ_α is obtained as follows: $\mathcal{C}$ consists of several (possibly renamed) $d_{j_1},\ldots,d_{j_m}$ of the form $(W_{j'}{\not\preceq}s_{j'})$ with variables $W_1,\ldots,W_m$ and terms $s_1,\ldots,s_m$. We then define $\theta_\alpha(W_{j'}) = Y$ for $j' \in \{1,\ldots,m\}$ and θ_α is the identity elsewhere.

The operator $\beta(G,\mathcal{C})$ adds constraints $\mathcal{C}$ to grammar rules in G of the forms positive and negative-S2:

$$\beta(G,\mathcal{C}) = \{c_1,\ldots,c_k,\theta_\beta(\mathcal{C}') \mapsto t{\in}\mathcal{L} \mid (c_1,\ldots,c_k \mapsto t{\in}\mathcal{L}) \in G \\ \wedge\ \nexists c_i, \nexists w : c_i \equiv (w{\in}\mathcal{L}) \wedge \mathcal{C}' = \{c'_i \in \mathcal{C} \mid c'_i \equiv (u_i{\not\preceq}v_i) \wedge t \preceq u_i\}\}$$

where the substitution θ_β is obtained as follows: $\mathcal{C}'$ consists of several (possibly renamed) $d_{j_1},\ldots,d_{j_m}$ of the form $(u_{j'}{\not\preceq}v_{j'})$ with terms $u_1,\ldots,u_m,v_1,\ldots,v_m$. We define θ_β such that $\theta_\beta(u_{j'}) \equiv t$ for $j' \in \{1,\ldots,m\}$ and $\mathcal{D}om(\theta_\beta) \subseteq \mathcal{V}ar(u_1) \cup \cdots \cup \mathcal{V}ar(u_m)$.

Finally, for a set S of grammar rules, we define the operator $\widehat{S}$ collecting all the grammar rules in S of the recursive form:

$$\widehat{S} = \{(t|_q{\in}\mathcal{L} \mapsto t{\in}\mathcal{L}) \in S \mid t|_q \in \mathcal{X}\}.$$

Lemma 2 (Grammar Shape Preservation) *The new grammar generated by the operator $newGrammar_s(G,\mathcal{C},\mathcal{H})$ satisfies the shape for grammar rules given in Definition 18.*

Proof. The operator $\alpha(G,\mathcal{C})$ takes negative-S1 grammar rules, i.e., of the form $t|_q{\notin}\mathcal{I}, t|_q{\not\preceq}s_1,\ldots,t|_q{\not\preceq}s_n \mapsto t{\in}\mathcal{L}$, and adds new constraints of the form $t|_q{\not\preceq}s'$ to them. Note that the set $\mathcal{C}$ of new constraints contains only constraints of the form $Y{\not\preceq}s'$ where Y is a variable because strategy $S1$ has been used. And note that the substitution θ_α applied to the new added constraints $\mathcal{C}$ ensures that they share the same variable $t|_q$ than the previous constraints in the rule. The operator $\beta(G,\mathcal{C})$ takes positive and negative grammar rules and adds new constraints of the form $t{\not\preceq}s'$ to them. Note that the subset $\mathcal{C}'$ of the set $\mathcal{C}$ of new constraints and the substitution θ_β applied to the new added constraints $\mathcal{C}'$ ensure that they share the same term t than the right-hand side of the grammar rule. The operator $\widehat{S}$ takes recursive grammar rules, i.e., $t|_q{\in}\mathcal{L} \mapsto t{\in}\mathcal{L}$, without any modification. □

Example 24 *Consider the following set $\mathcal{C}$ of new constraints (represented as*

a term of sort CtrSet*)*

$$\mathcal{C} = \{\ Z \not\preceq (A; n(A,r)),\ Z \not\preceq (Z'; n(A,r))\ \}$$

and the following set $\mathcal{H}$ *of new grammar rules*

$$\begin{aligned}\mathcal{H} = \{&Z{\in}\mathcal{L} \mapsto pk(c,Z){\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto pk(A,n(A,r);sk(B,Z)){\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto sk(A,Z){\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto Z;Y{\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto X;Z{\in}\mathcal{L},\\ &Z{\notin}\mathcal{I} \mapsto pk(A,n(A,r);Z){\in}\mathcal{L}\}\end{aligned}$$

Then, we add the set $\mathcal{C}$ *of new constraints and the set* $\mathcal{H}$ *of new grammar rules to the grammar* $G^0_{sd_3}$ *of Figure 4 in order to produce later the grammar* $G^1_{sd_3}$ *and thus compute*

$$newGrammar_{S1}(G^0_{sd_3}, \mathcal{C}, \mathcal{H}) = \alpha(G^0_{sd_3}, \mathcal{C}) \cup \alpha(\mathcal{H}, \mathcal{C}) \cup \widehat{G^0_{sd_3}} \cup \widehat{\mathcal{H}}$$

where

$$\alpha(G^0_{sd_3}, \mathcal{C}) = \{Z{\notin}\mathcal{I},\ Z \not\preceq (A'; n(A',r)),\ Z \not\preceq (Z'; n(A'',r')) \mapsto pk(A,Z){\in}\mathcal{L}\}$$

because we simply look for a rule that includes a constraint of the form $Y{\notin}\mathcal{I}$*, namely* $Z{\notin}\mathcal{I} \mapsto pk(A,Z){\in}\mathcal{L}$*, and add the (appropriately renamed) constraints* $\mathcal{C}$ *to the left part of the rule. We produce* $\alpha(\mathcal{H},\mathcal{C})$ *in a similar way*

$$\alpha(\mathcal{H}, \mathcal{C}) = \{Y{\notin}\mathcal{I}, Y \not\preceq (A'; n(A',r')), Y \not\preceq (Z'; n(A'',r'')) \mapsto pk(A, n(A,r); Y){\in}\mathcal{L}\}$$

The set $\widehat{G^0_{sd_3}}$ *is empty, i.e.,* $\widehat{G^0_{sd_3}} = \emptyset$*, since there is no rule in* $G^0_{sd_3}$ *with a constraint in the left part of the form* $(Y{\in}\mathcal{L})$*. Finally we collect all the rules in* $\mathcal{H}$ *with a constraint of the form* $(Y{\in}\mathcal{L})$

$$\begin{aligned}\widehat{\mathcal{H}} = \{&Z{\in}\mathcal{L} \mapsto pk(c,Z){\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto pk(A,n(A,r);sk(B,Z)){\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto sk(A,Z){\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto Z;Y{\in}\mathcal{L},\\ &Z{\in}\mathcal{L} \mapsto X;Z{\in}\mathcal{L}\}\end{aligned}$$

Thus, we finally obtain

$$
\begin{aligned}
&newGrammar_{S1}(G^0_{sd_3}, \mathcal{C}, \mathcal{H}) \\
&= \{Y \notin \mathcal{I}, Y \not\preceq (A'; n(A', r')), Y \not\preceq (Z'; n(A'', r'')) \mapsto pk(A, n(A, r); Y) \in \mathcal{L} \\
&\quad Z \notin \mathcal{I}, Z \not\preceq (A'; n(A', r)), Z \not\preceq (Z'; n(A'', r')) \mapsto pk(A, Z) \in \mathcal{L} \\
&\quad Z \in \mathcal{L} \mapsto pk(c, Z) \in \mathcal{L} \\
&\quad Z \in \mathcal{L} \mapsto pk(A, n(A, r); sk(B, Z)) \in \mathcal{L} \\
&\quad Z \in \mathcal{L} \mapsto sk(A, Z) \in \mathcal{L} \\
&\quad Z \in \mathcal{L} \mapsto Z; Y \in \mathcal{L} \\
&\quad Z \in \mathcal{L} \mapsto X; Z \in \mathcal{L}\}
\end{aligned}
$$

6.5.3 Optimizing Grammars: The Operator optimize(G)

The operator $optimize(G)$ removes redundant grammar rules and redundant constraints of the form $\_ \not\preceq \_$ as follows.

Definition 21 (Optimizing Grammars) *Given a grammar G, we define:*

$$optimize(G) = removeRules(removeConstraints(G))$$

where $removeConstraints(G) = \{maximal_{\sqsubseteq}(\mathcal{C}) \mapsto (t \in \mathcal{L}) \mid (\mathcal{C} \mapsto (t \in \mathcal{L})) \in G\}$ and, by definition, $maximal_{\sqsubseteq}(\mathcal{C}) \subseteq \mathcal{C}$ is the subset of constraints $c \in \mathcal{C}$ that are maximal elements in the partial order $\sqsubseteq$. And where $removeRules(G)$ is defined as follows: we first choose a grammar rule $(\mathcal{C} \mapsto (t \in \mathcal{L})) \in G$; if we have $\langle G - \{\mathcal{C} \mapsto (t \in \mathcal{L})\}, \mathcal{C}\rangle \vdash (t \in \mathcal{L})$, then we remove it from G, in any case we repeat the process until no more grammar rules can be removed (we reach a fixpoint).

The following result follows in a straightforward way from the definition.

Lemma 3 (Language Preservation) *Let G be a grammar, and $\mathcal{C}, \mathcal{D}$ be two sets of constraints. If $\langle G, \mathcal{D}\rangle \vdash \mathcal{C}$, then $\langle optimize(G), \mathcal{D}\rangle \vdash \mathcal{C}$.*

Example 25 *For the grammar $newGrammar_{S1}(G^0_{sd_3}, \mathcal{C}, \mathcal{H})$ of Example 24, its optimization is*

$$
\begin{aligned}
optimize(G) = \{&Z \notin \mathcal{I},\ Z \not\preceq (Z'; n(B, r')) \mapsto pk(A, n(A, r); Z) \in \mathcal{L} \\
&Z \notin \mathcal{I},\ Z \not\preceq (Z'; n(B, r)) \mapsto pk(A, Z) \in \mathcal{L}, \\
&Z \in \mathcal{L} \mapsto pk(c, Z) \in \mathcal{L}, \\
&Z \in \mathcal{L} \mapsto pk(A, n(A, r); sk(B, Z)) \in \mathcal{L}, \\
&Z \in \mathcal{L} \mapsto sk(A, Z) \in \mathcal{L}, \\
&Z \in \mathcal{L} \mapsto Z; Y \in \mathcal{L}, \\
&Z \in \mathcal{L} \mapsto X; Z \in \mathcal{L}
\end{aligned}
$$

since $(Y \not\trianglelefteq A'; n(A', r)) \sqsubseteq (Y \not\trianglelefteq Z'; n(A'', r'))$. *And if we had a rule* $Y \in \mathcal{L} \mapsto pk(c, n(A, r); sk(B, Y)) \in \mathcal{L}$, *this would be removed, since it can be obtained from the other rules.*

6.5.4 Generating New Grammar Rules from a Previous Grammar Rule: The Relation $\langle \mathcal{C}, \mathcal{H}, G \rangle \Rightarrow_{\mathcal{P}, G_j^k, s} \langle \mathcal{C}', \mathcal{H}', G'' \rangle$

Given a set G of grammar rules, a set $\mathcal{C}$ of constraints, and a set $\mathcal{H}$ of grammar rules, we define the transformation relation $\Rightarrow_{\mathcal{P}, G_j^k, s}$ on tuples $\langle \mathcal{C}, \mathcal{H}, G \rangle$, that extends $\mathcal{H}$ with new grammar rules and $\mathcal{C}$ with new constraints, all associated to a grammar rule $g \in G$.

Definition 22 (Generating New Rules and Constraints) *Given a grammar G to be transformed, a grammar G_j^k, a set of constraints $\mathcal{C}$, a set of grammar rules $\mathcal{H}$, and a generation strategy s, we define*

$$\langle \mathcal{C}, \mathcal{H}, G \cup \{g\} \rangle \Rightarrow_{\mathcal{P}, G_j^k, s} \langle \mathcal{C} \cup \mathcal{C}_g, \mathcal{H} \cup \mathcal{H}_g, G \rangle$$

where the set of new constraints is $\mathcal{C}_g = \cup\{\mathcal{C}_{g \rightsquigarrow_\sigma g'}\}$, the set of new grammar rules is $\mathcal{H}_g = \cup\{\mathcal{H}_{g \rightsquigarrow_\sigma g'}\}$, and $\mathcal{C}_{g \rightsquigarrow_\sigma g'}$ and $\mathcal{H}_{g \rightsquigarrow_\sigma g'}$ are defined for each backwards narrowing step $g \rightsquigarrow_{\sigma, \mathcal{P}^{-1}, G_j^{k-1}} g'$ as $heuristics_{G_j^k, s}(g, \sigma, g') = \langle \mathcal{C}_{g \rightsquigarrow_\sigma g'}, \mathcal{H}_{g \rightsquigarrow_\sigma g'} \rangle$ such that either $\mathcal{C}_{g \rightsquigarrow_\sigma g'}$ or $\mathcal{H}_{g \rightsquigarrow_\sigma g'}$ are not empty. The $\rightsquigarrow_{\sigma, \mathcal{P}^{-1}, G_j^{k-1}}$ relation is defined in the following.

Example 26 *For the grammar $G_{sd_3}^0$ of Figure 4, we obtain a set of backwards narrowing steps $sd_3 \rightsquigarrow_{\sigma, \mathcal{P}^{-1}, G_{sd_3}^{0\ -1}} g'$ from the only rule sd_3 in $G_{sd_3}^0$ and apply $heuristics_{G_{sd_3}^0, S1}(sd_3, \sigma, g')$ to each one of them. Then, the sets $\mathcal{H}_{sd_3} = \cup\{\mathcal{H}_{sd_3 \rightsquigarrow_\sigma g'}\}$ and $\mathcal{C}_{sd_3} = \cup\{\mathcal{C}_{sd_3 \rightsquigarrow_\sigma g'}\}$ are the sets $\mathcal{H}$ and $\mathcal{C}$ shown in Example 24.*

6.5.5 Generating a Pre-grammar Rule from a Previous Grammar Rule: The Backwards Narrowing Relation $g \rightsquigarrow_{\sigma, \mathcal{P}^{-1}, G_j^{k-1}} g'$

Given a rule g in a grammar G_j^k, we consider each backwards narrowing step from g producing what we call a *pre-grammar rule* g' which we will use, together with the heuristics, to generate new grammar rules that will be included into G_j^{k+1}.

A pre-grammar rule is validated using the grammar G_j^k produced up to now.

Definition 23 (G_j^k-expandable Pre-grammar Rule) *A pre-grammar rule $g \equiv \mathcal{C} \mapsto (t_1, \ldots, t_n) \in \mathcal{L}$ is* G_j^k-expandable *iff*

(1) $\mathcal{C}$ is $\not\trianglelefteq$-satisfiable; and
(2) for each t_i, we have that $\langle G_j^k, \mathcal{C}\rangle \not\vdash (t_i \in \mathcal{L})$ and $t_i \notin \mathcal{I}$ does not occur in $\mathcal{C}$.

Intuitively, a pre-grammar rule is G_j^k-*expandable* if: (i) the current constraints of the form $u \not\trianglelefteq p$ are satisfiable, (ii) none of the messages are captured by the grammar G_j^k, and (iii) none of the messages are discarded by the learn-only-once restriction. If a pre-grammar rule is G_j^k-expandable, we apply the heuristics to generate a new grammar rule such that it implies that the conditions of the pre-grammar rule are satisfied. Otherwise, we discard that pre-grammar rule.

The backwards narrowing relation producing pre-grammar rules is denoted by the arrow $\rightsquigarrow_{\sigma,\mathcal{P}^{-1},G_j^{k-1}}$, where σ is the computed unifier, $\mathcal{P}$ is the set of strands generating the set $R_{\mathcal{P}}$ of rules to be used for narrowing modulo $E_{\mathcal{P}}$, and G_j^k is used to further narrow the grammar rule and for G_j^k-expandable test. Recall that rules are always renamed to avoid variable name clashes.

Definition 24 (Generating a Pre-grammar Rule) *Given a grammar rule g, the set of strands $\mathcal{P}$, and a grammar G_j^k, the relation $\rightsquigarrow_{\sigma,\mathcal{P}^{-1},G_j^{k-1}}$ producing a pre-grammar rule g'' is defined as*

$$g \rightsquigarrow_{\sigma,\mathcal{P}^{-1},G_j^{k-1}} g''$$

if $g \rightsquigarrow^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R_{S\mathcal{P}}^{-1},E_{S\mathcal{P}}} g'$, $g' \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G_j^k},E_{\mathcal{G}}} g''$, *and g'' is* G_j^k-expandable

The narrowing relation $\rightsquigarrow^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R_{S\mathcal{P}}^{-1},E_{S\mathcal{P}}}$ used in the Maude-NPA is slightly more restrictive than the ordinary narrowing relation $\rightsquigarrow_{\sigma,\phi_{S\mathcal{P}},R_{S\mathcal{P}}^{-1},E_{S\mathcal{P}}}$ (see Appendix A for details).

Recall from Sections 6.2 and 6.3 that the symbol $\_ \mapsto \_$ has its first argument frozen in $R_{S\mathcal{P}}$, whereas it has instead its second argument frozen in R_G, i.e., given a grammar rule $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in \mathcal{L}$, the relation $\rightsquigarrow^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R_{S\mathcal{P}}^{-1},E_{S\mathcal{P}}}$ narrows only the right part, whereas the relation [10] $\rightarrow_{\phi_{\mathcal{G}},\mathcal{R}^{-1}_{G_j^k},E_{\mathcal{G}}}$ rewrites only the left part.

Example 27 *Consider the following backwards narrowing steps. For the grammar $G^0_{sd_3}$ of Figure 4, we have the following backwards narrowing step using*

[10] Note that in (Escobar et al., 2005) we wrote $g' \rightsquigarrow^{!}_{\theta,\phi_{\mathcal{G}},R^{-1}_{G_j^k},E_{\mathcal{G}}} g''$ in Definition 24. However, such normalization by narrowing usually does not terminate for the considered rewrite theory $R^{-1}_{G_j^k}$.

the protocol rule $(p2) \equiv pk(A, n(A, r); Z') \rightarrow pk(B, Z')$*:*

$$Z \notin \mathcal{I} \mapsto pk(A, Z) \in \mathcal{L}$$
$$\rightsquigarrow^{\bullet}_{id, \phi_{SP}, R^{-1}_{SP}, E_{SP}} \quad Z \notin \mathcal{I} \mapsto pk(A', n(A', r); sk(B, pk(A, Z))) \in \mathcal{L}$$

where $Z \notin \mathcal{I} \mapsto pk(A', n(A', r); sk(B, pk(A, Z))) \in \mathcal{L}$ *is a* $G^0_{sd_3}$*-expandable grammar rule and we have solved the equational unification problem* $pk(B, Z') =_{E_{SP}} pk(A, Z)$ *using the unifier* id *and the equation* $sk(Y, pk(Y, Z)) = Z$*. We also have the following backwards narrowing step for* $G^0_{sd_3}$ *using the protocol rule* $(p7) \equiv M \rightarrow pk(Y, M)$*:*

$$Z \notin \mathcal{I} \mapsto pk(A, Z) \in \mathcal{L} \ \rightsquigarrow^{\bullet}_{id, \phi_{SP}, R^{-1}_{SP}, E_{SP}} \ Z \notin \mathcal{I} \mapsto Z \in \mathcal{L}$$

but this step is not $G^0_{sd_3}$*-expandable, because term* Z *appears in a constraint* $Z \notin \mathcal{I}$*. For the grammar* $G^1_{sd_3}$ *of Figure 4, we have the following backwards narrowing step using* $(p1) \equiv \emptyset \rightarrow pk(B, A; n(A, r))$*:*

$$Z \notin \mathcal{I}, \ Z \not\preceq (Z'; n(B, r)) \mapsto pk(A, Z) \in \mathcal{L}$$
$$\rightsquigarrow^{\bullet}_{[Z/(A'; n(A', r'))], \phi_{SP}, R^{-1}_{SP}, E_{SP}} (A'; n(A', r') \notin \mathcal{I}, (A'; n(A', r') \not\preceq (Z'; n(B, r))) \mapsto \emptyset$$

but this step is not $G^0_{sd_3}$*-expandable, because* $(A'; n(A', r') \preceq (Z'; n(B, r))$*. Again for* $G^1_{sd_3}$*, we have the following backwards narrowing step using* $(p5) \equiv M_1; M_2 \rightarrow M_1$*:*

$$Y \in \mathcal{L} \mapsto pk(c, Y) \in \mathcal{L} \rightsquigarrow^{\bullet}_{id, \phi_{SP}, R^{-1}_{SP}, E_{SP}} \ Y \in \mathcal{L} \mapsto pk(c, Y); M_2 \in \mathcal{L}$$

but this step is not $G^!_{sd_3}$*-expandable because the term* $(pk(c, Y); M_2)$ *is captured by* $G^1_{sd_3}$*, i.e.,* $\langle \ G^1_{sd_3}, \ Y \in \mathcal{L} \ \rangle \vdash (pk(c, Y); M_2) \in \mathcal{L}$*.*

6.5.6 Deciding which Grammar Rule or Restriction to Generate from a Pregrammar Rule: The Operator $heuristics_{G^k_j, s}(g, \sigma, g')$

Here, we use the result of a backwards narrowing step $\rightsquigarrow_{\sigma, \mathcal{P}^{-1}, G^{k-1}_j}$ to decide which grammar rule or constraint should be generated in order to refine the language $\mathcal{L}$ associated to the grammar G^k_j. We define the operator $heuristics_{G^k_j, s}(g, \sigma, g')$ that yields a pair $\langle \mathcal{C}, \mathcal{H} \rangle$, where $\mathcal{C}$ is a set of constraints (empty or with one constraint) and $\mathcal{H}$ is a set of new grammar rules (empty or with one rule).

Definition 25 (Heuristics Generating New Rules or Constraints) *The following inference rules define the four heuristics, where* s *is a global strategy parameter that can be instantiated to either* $S1$ *or* $S2$ *(see below), the variable* Y *is a fresh new variable,* $g = \mathcal{C} \mapsto t \in \mathcal{L}$*, and* $g' = \mathcal{D} \mapsto (s_1, \ldots, s_n) \in \mathcal{L}$*:*

$$H1\ \frac{\exists s_i, p \in \mathcal{P}os(s_i) : \langle G_j^k, \mathcal{D}\rangle \vdash (s_i|_p \in \mathcal{L})}{heuristics_{G_j^k,s}(g,\sigma,g') = \langle \emptyset, \{Y \in \mathcal{L} \mapsto s_i[Y]_p \in \mathcal{L}\}\rangle}$$

$$H2a\ \frac{\exists u, d_i \in \mathcal{D} : d_i \equiv (u \notin \mathcal{I}) \wedge u \notin \mathcal{X}}{heuristics_{G_j^k,s}(g,\sigma,g') = \langle \{X \not\preceq u\}, \emptyset\rangle}$$

$$H2b\ \frac{\sigma(t) \not\equiv t \quad \nexists u, d_i \in \mathcal{D} : d_i \equiv (u \in \mathcal{L})}{heuristics_{G_j^k,s}(g,\sigma,g') = \langle \{t \not\preceq \sigma(t)\}, \emptyset\rangle}$$

$$H3\ \frac{\exists d_i \in \mathcal{D}, s_j, p \in \mathcal{P}os(s_j) : d_i \equiv (s_j|_p \notin \mathcal{I})}{heuristics_{G_j^k,s}(g,\sigma,g') = \langle \emptyset, \{Y \notin \mathcal{I} \mapsto s_j[Y]_p \in \mathcal{L}\}\rangle}$$

The intuition behind the heuristics is the following. If we have a grammar G with its associated language $\mathcal{L}$ and we consider one grammar production rule in G, e.g., $g = c_1, \ldots, c_k \mapsto t \in \mathcal{L}$, and all of the possible predecessors of message t w.r.t. the protocols rules $R_{S\mathcal{P}}$, e.g., $g' = \mathcal{D} \mapsto (s_1, \ldots, s_n) \in \mathcal{L}$ such that $t \rightsquigarrow_{\sigma,\mathcal{P}^{-1},G_j^{k-1}} s_1, \ldots, s_n$, and also none of $s_1, \ldots, s_n$ is captured by G_j^k, then we want G_j^k to be able to capture one of them and thus we try to extend (or complete) G_j^k adding production rules. Heuristics H1 and H3 make an over-approximation and introduce terms in the grammar that might be reachable for the intruder, whereas heuristics H2a and H2b restrict the grammar:

- $(H1)$ This heuristic extends the grammar G_j^k and, essentially, looks for terms that could be captured by G_j^k but are not. For Example 2, $\emptyset \mapsto m \in \mathcal{L}$ is the only rule in grammar G_0, we also have that message $e(k, m)$ is learned by the intruder from message m and since m is a proper subterm of $e(k, m)$, we add a grammar rule $Y \in \mathcal{L} \mapsto e(k, Y) \in \mathcal{L}$ that captures message $e(k, m)$.
- $(H2a)$ This heuristic detects that there is a constraint $u \notin \mathcal{I}$ in the constraints $\mathcal{D}$ of rule g', where term u is not a variable. This implies that the intruder can learn some partial data (symbols introduced by unification) and thus such partial data should be excluded from the grammar G_j^k.
- $(H2b)$ This heuristic detects also that some partial data can be learned by the intruder and also excludes them from the grammar. The difference between $H2a$ and $H2b$ is explained below.
- $(H3)$ This heuristic detects that the intruder has been able to learn a message s_j that contains a subterm that appears as a constraint $s_j|_p \notin \mathcal{I}$ in $\mathcal{D}$, which implies that it must be unknown by the intruder. Therefore, assuming that $s_j|_p$ is unknown by the intruder, we have to include s_j in the grammar and thus introduce a rule $Y \notin \mathcal{I} \mapsto s_j[Y]_p \in \mathcal{L}$.

These heuristics are applied following one of the two possible global strategies for s:

- $S1$. Apply heuristics in the following order: try $H1$, if it fails try $H2a$, if it fails try $H3$, otherwise stop the whole grammar generation process for this grammar.
- $S2$. Like $S1$, but try $H2b$ instead of $H2a$.

Note that the choice of $S1$ or $S2$ is fixed during the entire generation of a grammar.

The difference between strategies $S2$ and $S1$ (i.e., between heuristics $H2b$ and $H2a$) is that $S2$ generates more restricted grammars, thus cutting less terms, than strategy $S1$. However, strategy $S1$ can be applied in fewer situations than strategy $S2$ because of the $\_\notin\mathcal{I}$ constraint. Therefore, for a seed term of the form $\emptyset \mapsto t{:}\mathcal{L}$ only strategy $S2$ can be applied, whereas for a seed term of the form $t|_q\notin\mathcal{I} \mapsto t{:}\mathcal{L}$ it is usually better to start with strategy $S2$ and if it fails, try strategy $S1$. A deeper study of both strategies, refinements, and their automatization is left for future work.

Example 28 *Consider the grammar $G^0_{sd_3}$ of Figure 4. For the backwards narrowing step using protocol rule (p5)*

$$g \equiv Z\notin\mathcal{I} \mapsto pk(A,Z){:}\mathcal{L} \rightsquigarrow_{id,\mathcal{P}^{-1},{G^0_{sd_3}}^{-1}} g' \equiv Z\notin\mathcal{I} \mapsto pk(A,Z); M_2{:}\mathcal{L}$$

we can apply heuristic 1 *yielding*

$$heuristics_{G^0_{sd_3},S1}(g,id,g') = \langle\emptyset, \{Y{:}\mathcal{L} \mapsto Y; M_2{:}\mathcal{L}\}\rangle,$$

since the subterm $pk(A,Z)$ is already in the language of $G^0_{sd_3}$, i.e., $\langle G^0_{sd_3}, Z\notin\mathcal{I}\rangle \vdash pk(A,Z){:}\mathcal{L}$. For the following backwards narrowing step using protocol rule (p1)

$$g \equiv Z\notin\mathcal{I} \mapsto pk(A,Z){:}\mathcal{L} \rightsquigarrow_{[Z/A';n(A',r)],\mathcal{P}^{-1},{G^0_{sd_3}}^{-1}} g' \equiv A'; n(A',r)\notin\mathcal{I} \mapsto \emptyset$$

we can apply heuristic 2a *yielding*

$$heuristics_{G^0_{sd_3},S1}(g,id,g') = \langle\{Z\not\leq A; n(A,r)\}, \emptyset\rangle,$$

since we have found a constraint $c_i \equiv u \notin \mathcal{I}$ such that u is not a variable. For the backwards narrowing step using protocol rule (p2)

$$g \equiv Z\notin\mathcal{I} \mapsto pk(A,Z){:}\mathcal{L} \rightsquigarrow_{id,\mathcal{P}^{-1},{G^0_{sd_3}}^{-1}} g' \equiv Z\notin\mathcal{I} \mapsto pk(A',n(A',r);Z){:}\mathcal{L}$$

we can apply heuristic 3 *yielding*

$$heuristics_{G^0_{sd_3},S1}(g, id, g') = \langle \emptyset, \{Y \notin \mathcal{I} \mapsto pk(A', n(A', r); Y) \in \mathcal{L}\} \rangle,$$

since the subterm Z *is already in an original constraint of the form* $\_ \notin \mathcal{I}$ *in* g*. And for a grammar rule* $Z \notin \mathcal{I} \mapsto pk(Y, Z) \in \mathcal{L}$*, where* Y *and* Z *are variables of sort* Msg*, we can consider the following backwards narrowing step using protocol rule (p2), where* B *is a variable of sort* Name

$$g \equiv Z \notin \mathcal{I} \mapsto pk(Y, Z) \in \mathcal{L} \rightsquigarrow_{[Y/B], \mathcal{P}^{-1}, {G^0_{sd_3}}^{-1}} \; g' \equiv Z \notin \mathcal{I} \mapsto pk(A, n(A, r); Z) \in \mathcal{L}$$

and then we can apply heuristic 2*b yielding*

$$heuristics_{G^0_{sd_3},S2}(g, id, g') = \langle \{pk(Y, Z) \not\leq pk(B, Z)\}, \emptyset \rangle,$$

since there is no constraint of the form $\_ \in \mathcal{L}$ *and the substitution binds variable* Y *of* $pk(Y, Z)$*.*

When no heuristic can be applied, the grammar generation process stops with failure for such a seed term, as shown in the following example.

Example 29 *Consider again the Needham-Schroeder Protocol of Example 9 with the following new strand representing some initial knowledge of the intruder (where* A *is a variable of sort* Name*):*

(s7) $[A^+]$

This strand is transformed into the following simplified protocol rule for the grammar generation process

(p9) $\emptyset \rightarrow A$

Let us consider a new seed term represented by the initial grammar

$$G^0_{sd_5} \equiv \emptyset \mapsto A \in \mathcal{L}$$

This seed term means that the user believes all the principal's names are unknown for the intruder, which is obviously false because of the new introduced strand. The grammar generation process fails for this seed term, since no heuristic can be applied to a pre-grammar rule obtained from the grammar rule in $G^0_{sd_5}$*. That is, for the following backwards narrowing step obtained using the simplified protocol rules:*

$$\emptyset \mapsto A \in \mathcal{L} \quad \rightsquigarrow^{\bullet}_{id, \phi_{S\mathcal{P}}, R^{-1}_{S\mathcal{P}}, E_{S\mathcal{P}}} \quad \emptyset \mapsto \emptyset \in \mathcal{L}$$

no heuristic of Definition 25 can be applied for either strategy $S1$ *or* $S2$ *because (i) there is no term in the right-hand side of the pre-grammar rule* $\emptyset \mapsto \emptyset \in \mathcal{L}$*,*

and therefore heuristics $H1$ and $H3$ cannot be applied, and (ii) the substitution computed during the backwards narrowing step is the identity, and therefore heuristics $H2a$ and $H2b$ cannot be applied either.

The grammar generation might also not terminate adding new grammar productions, as shown in the following example.

Example 30 *Consider a protocol with a cryptographic system with only two keys k and k', where only k is shared by all principals. The following strand defines the protocol:*

(s1) $[pk(k', A; M)^-, pk(k', M)^+]$

The Dolev-Yao strands are defined as follows, where the intruder can encrypt and decrypt using only the key k:

(s2) $[M_1^-, M_2^-, (M_1; M_2)^+]$
(s3) $[(M_1; M_2)^-, M_1^+, M_2^+]$
(s4) $[M^-, pk(k, M)^+]$
(s5) $[M^-, sk(k, M)^+]$

The equational properties are the same as in Example 3. These strands are transformed into the following simplified protocol rules for grammar generation

(p1) $pk(k', A; M) \to pk(k', M)$
(p2) $M_1, M_2 \to M_1; M_2$
(p3) $M_1; M_2 \to M_1$
(p4) $M_1; M_2 \to M_2$
(p5) $M \to pk(k, M)$
(p6) $M \to sk(k, M)$

Let us consider a seed term represented by the initial grammar

$$G^0_{sd_1} \equiv Z{\notin}\mathcal{I} \mapsto pk(k', Z){\in}\mathcal{L}$$

This seed term asks whether the intruder can learn any message encrypted with the key k' provided that the message itself is unknown. The grammar generation process never terminates for this seed term, as shown in the generated grammars of Figure 6. The point is that at each grammar generation step, there are several backwards narrowing steps of the form

$$t|_p{\notin}\mathcal{I} \mapsto pk(k', t){\in}\mathcal{L} \quad \leadsto^{\bullet}_{id,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \quad t|_p{\notin}\mathcal{I} \mapsto pk(k', A; t){\in}\mathcal{L}$$

and each of these pre-grammar rules $t|_p{\notin}\mathcal{I} \mapsto pk(k', A; t){\in}\mathcal{L}$ is not captured by the previous grammar. Thus, we can apply heuristic $H3$ to each of them, obtaining a new rule where the variable $t|_p$ is replaced by a new variable, i.e., the

$G^1_{sd_1}$

$Z{\in}\mathcal{L} \mapsto pk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto sk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto X; Z{\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto Z; Y{\in}\mathcal{L}$

$Z{\notin}\mathcal{I} \mapsto pk(k', Z){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; Z){\in}\mathcal{L}$

$Z{\in}\mathcal{L} \mapsto pk(k', A; sk(k', Z)){\in}\mathcal{L}$

$G^2_{sd_1}$

$Z{\in}\mathcal{L} \mapsto pk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto sk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto X; Z{\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto Z; Y{\in}\mathcal{L}$

$Z{\notin}\mathcal{I} \mapsto pk(k', Z){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; Z){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; (A; Z)){\in}\mathcal{L}$

$Z{\in}\mathcal{L} \mapsto pk(k', A; sk(k', Z)){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto pk(k', A; (A; sk(k', Z))){\in}\mathcal{L}$

$G^3_{sd_1}$

$Z{\in}\mathcal{L} \mapsto pk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto sk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto X; Z{\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto Z; Y{\in}\mathcal{L}$

$Z{\notin}\mathcal{I} \mapsto pk(k', Z){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; Z){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; (A; Z)){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; (A; (A; Z))){\in}\mathcal{L}$

$Z{\in}\mathcal{L} \mapsto pk(k', A; sk(k', Z)){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto pk(k', A; (A; sk(k', Z))){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto pk(k', A; (A; (A; sk(k', Z)))){\in}\mathcal{L}$

$G^4_{sd_1}$

$Z{\in}\mathcal{L} \mapsto pk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto sk(k, Z){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto X; Z{\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto Z; Y{\in}\mathcal{L}$

$Z{\notin}\mathcal{I} \mapsto pk(k', Z){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; Z){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; (A; Z)){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; (A; (A; Z))){\in}\mathcal{L}$
$Z{\notin}\mathcal{I} \mapsto pk(k', A; (A; (A; (A; Z)))){\in}\mathcal{L}$

$Z{\in}\mathcal{L} \mapsto pk(k', A; sk(k', Z)){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto pk(k', A; (A; sk(k', Z))){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto pk(k', A; (A; (A; sk(k', Z)))){\in}\mathcal{L}$
$Z{\in}\mathcal{L} \mapsto pk(k', A; (A; (A; (A; sk(k', Z))))){\in}\mathcal{L}$

Fig. 6. Infinite grammar generation sequence in Example 30.

very same $t|_p{\notin}\mathcal{I} \mapsto pk(k', A; t){\in}\mathcal{L}$. *Moreover, these new rules are not implied from the previous ones, and thus they are not removed by the optimization stage. A similar argument exists for grammar productions in Figure 6 of the form* $t|_p{\in}\mathcal{L} \mapsto pk(k', A; sk(k', t)){\in}\mathcal{L}$ *and the heuristic* $H1$.

6.6 Grammar Unreachability

This section introduces the main theorems regarding soundness of the grammars, i.e., if a message m belongs to the language of a grammar $G^!_{sd_i}$ obtained from a seed term sd_i, then such message cannot be learned by the intruder.

Note that the user starts the grammar generation process providing a seed term that he/she believes that the intruder cannot learn. Therefore, these seed terms may be unsound, and must be refined during the grammar generation process to more precise sound versions. Intermediate grammars might also be unsound, since further backwards protocol steps must be explored to refine the intermediate grammar. Therefore, we can prove soundness of the grammars only at the end, for the fixpoint grammars. Only at this final state do we know that every possible protocol step has been analyzed for every possible message belonging to the current grammar.

The following properties establish that, whenever we have a protocol state SS and a message m that the intruder must learn, i.e., that appears as an input message m^- in the past part of some strand of SS, and that it is in the

language of a grammar $G^!_{sd_i}$, then for every previous state SS' of SS w.r.t. the protocol rules $R_{\mathcal{P}}$, i.e.,

$$SS \ \leadsto^{\bullet}_{\sigma,\phi_{\mathcal{P}},R^{-1}_{\mathcal{P}},E_{\mathcal{P}}} \ SS'$$

there is a message m' that the intruder must also learn, i.e., m' is an input message of the past part of some strand of SS'. This fact, together with the assumption that the intruder doesn't know anything in an initial state, are the basis of the soundness of the grammars. However, grammars are generated using the rewrite theory $R_{S\mathcal{P}}$ instead of the rewrite theory $R_{\mathcal{P}}$ and therefore we must rephrase the previous statement in terms of $R_{S\mathcal{P}}$. That is, whenever we have a message m that is in the language of a grammar $G^!_{sd_i}$, for every previous set $u_1, \ldots, u_k$ of messages obtained by the simplified protocol rules $\mathcal{R}_{S\mathcal{P}}$, i.e.,

$$t \ \leadsto^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \ u_1, \ldots, u_k$$

there is a message u_i that the intruder must learn and that it is also in the grammar $G^!_{sd_i}$.

First, we state an auxiliary corollary that follows in a straightforward way from the definition of the relation $\leadsto_{\sigma,\mathcal{P}^{-1},G_j^{k\,-1}}$.

Corollary 3 *Let G^k_j be a grammar. If $g \equiv (\mathcal{C} \mapsto t{\in}\mathcal{L})$ is a grammar rule in G^k_j that is in normal form w.r.t. the relation $\leadsto_{\mathcal{P}^{-1},G_j^{k\,-1}}$, then either:*

(1) g is in normal form w.r.t. relation $\leadsto^{\bullet}_{\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}}$; or
(2) for all g', g'', and σ such that $g \ \leadsto^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \ g'$ and $g' \ \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^k_j},E_{\mathcal{G}}} \ g''$,

then g'' is not G^k_j-expandable, i.e., $g'' \equiv (\mathcal{D} \mapsto (t_1, \ldots, t_n){\in}\mathcal{L})$ and either
(a) $\mathcal{D}$ is not $\not\leq$-satisfiable, or
(b) $\mathcal{D}$ is $\not\leq$-satisfiable and $\exists t_i$ s.t. $\langle G^k_j, \mathcal{D}\rangle \vdash (t_i{\in}\mathcal{L})$ or $(t_i{\notin}\mathcal{I})$ in $\mathcal{D}$.

The following auxiliary corollary follows in a straightforward way from the definition of the relation $\Rrightarrow_{\mathcal{P},G^k_j,s}$.

Corollary 4 *If $G^!$ is the fixpoint w.r.t. $\Rrightarrow_{\mathcal{P},G,s}$ of a grammar G, then for all $g \equiv (\mathcal{C} \mapsto t{\in}\mathcal{L})$ in $G^!$, g is in normal form w.r.t. the relation $\leadsto_{\mathcal{P}^{-1},G^{!\,-1}}$.*

The following result states that, given a narrowing sequence from a term t with a substitution ρ, then we can simulate ρ by narrowing just from t.

Theorem 7 *Let $\mathcal{R} = (\Sigma, \phi, E, R)$ be a topmost rewrite theory, $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, and ρ, σ be substitutions such that $\rho(t) \ \leadsto^*_{\sigma,R,E} \ t'$. Then, there are substitutions θ, τ and a term t'' such that $t \ \leadsto^*_{\theta,R,E} \ t''$, $\sigma(\rho(t)) \equiv \tau(\theta(t))$, and $t' \equiv \tau(t'')$.*

Proof. $\rho(t) \ \leadsto^*_{\sigma,R,E} \ t'$ implies $\sigma(\rho(t)) \ \rightarrow^*_{R,E} \ t'$, and then, by Theorem 2 there

are substitutions θ, τ and a term t'' such that $t \leadsto^{*}_{\theta,R,E} t''$, $\sigma(\rho(t)) \equiv \tau(\theta(t))$, and $t' \equiv \tau(t'')$. $\square$

Now, we state and prove the following key result for the simplified protocol rules.

Lemma 4 *If $G^!$ is the fixpoint w.r.t. $\Rrightarrow_{\mathcal{P},G,s}$ of a grammar G. Let $\mathcal{C}$ be a term of sort* CtrSet *such that it does not contain any constraint of the form $\_ \in \mathcal{L}$ and t be a term of sort* Msg *such that $\langle G^!, \mathcal{C}\rangle \vdash (t \in \mathcal{L})$. Then, for each backwards narrowing step $t \leadsto^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} u_1, \ldots, u_n$ such that $\sigma(\mathcal{C})$ is $\not\trianglerighteq$-satisfiable, there is a u_i such that $\langle G^!, \sigma(\mathcal{C})\rangle \vdash (u_i \in \mathcal{L})$ or $(u_i \notin \mathcal{I})$ occurs in $\sigma(\mathcal{C})$.*

Proof. $\langle G^!, \mathcal{C}\rangle \vdash (t \in \mathcal{L})$ implies there is $\mathcal{C}_t$ such that $\alpha \equiv (t \in \mathcal{L}) \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \mathcal{C}_t$, and $\mathcal{C}_t \sqsubseteq \mathcal{C}$. Let us take $g \equiv \mathcal{D} \mapsto s \in \mathcal{L}$ as the grammar rule applied in the first rewrite step of α. There is a substitution τ such that $t \equiv \tau(s)$ and $\alpha \equiv (t \in \mathcal{L}) \rightarrow_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \tau(\mathcal{D}) \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \mathcal{C}_t$.

Now, let us focus on g. By Theorem 7 and $\phi_{S\mathcal{P}}(\_ \mapsto \_) = \{1\}$, g can simulate $t \leadsto^{\bullet}_{\sigma,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} u_1, \ldots, u_n$, i.e., there are substitutions ρ, τ_ρ and terms $s_1, \ldots, s_n$ such that $g \leadsto^{\bullet}_{\rho,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} g'$, $g' \equiv \rho(\mathcal{D}) \mapsto (s_1, \ldots, s_n) \in \mathcal{L}$, $\sigma(t) \equiv \tau_\rho(\rho(s))$, $u_i \equiv \tau_\rho(s_i)$ for $1 \leq i \leq n$, and $\sigma(\tau(\mathcal{D})) \equiv \tau_\rho(\rho(\mathcal{D}))$.

Since $\phi_{\mathcal{G}}(\_ \mapsto \_) = \{2\}$, $\sigma(\tau(\mathcal{D})) \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \sigma(\mathcal{C}_t)$ can be partially simulated by g' and rewriting, i.e., there is a term $\mathcal{D}'$ such that $g' \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} g''$, $g'' \equiv \mathcal{D}' \mapsto (s_1, \ldots, s_n) \in \mathcal{L}$, and $\tau_\rho(\mathcal{D}') \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \sigma(\mathcal{C}_t)$. That is, we have

$$\mathcal{D} \mapsto s \in \mathcal{L} \leadsto^{\bullet}_{\rho,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} \rho(\mathcal{D}) \mapsto (s_1, \ldots, s_n) \in \mathcal{L}$$

and

$$\rho(\mathcal{D}) \mapsto (s_1, \ldots, s_n) \in \mathcal{L} \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \mathcal{D}' \mapsto (s_1, \ldots, s_n) \in \mathcal{L}$$

such that $\sigma(t) \equiv \sigma(\tau(s)) \equiv \tau_\rho(\rho(s))$, $u_i \equiv \tau_\rho(s_i)$, and

$$\sigma(t) \in \mathcal{L} \rightarrow_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \sigma(\tau(\mathcal{D})) \equiv \tau_\rho(\rho(\mathcal{D})) \rightarrow^{*}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \tau_\rho(\mathcal{D}') \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} \sigma(\mathcal{C}_t)$$

Since $G^!$ is a fixpoint grammar, by Corollary 4, we have that g is a normal form w.r.t. $\leadsto_{\mathcal{P}^{-1},G^{!-1}}$. By Corollary 3, we have that for $g \leadsto^{\bullet}_{\rho,\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} g'$ and $g' \rightarrow^{!}_{\phi_{\mathcal{G}},R^{-1}_{G^!},E_{\mathcal{G}}} g''$, there is a s_j such that either $\langle G^!, \mathcal{D}'\rangle \vdash (s_j \in \mathcal{L})$ or $(s_j \notin \mathcal{I})$ occurs in $\mathcal{D}'$. Note that $\mathcal{D}'$ is $\not\trianglerighteq$-satisfiable because $\sigma(\mathcal{C})$ is, i.e., by Lemma 1, $\sigma(\mathcal{C}_t) \sqsubseteq \sigma(\mathcal{C})$, by Corollary 1, $\sigma(\mathcal{C}_t)$ is $\not\trianglerighteq$-satisfiable, by Corollary 2, $\mathcal{C}_t$ is $\not\trianglerighteq$-satisfiable, then, since terms rooted by $\_ \not\trianglerighteq \_$ are normal forms w.r.t. $\mathcal{R}_{G^!}$, we have that all constraints of the form $\_ \not\trianglerighteq \_$ in $\tau_\rho(\mathcal{D}')$ are constraints

of $\sigma(\mathcal{C}_t)$, thus making $\tau_\rho(\mathcal{D}')$ $\not\leq$-satisfiable, and finally, by Corollary 2, $\mathcal{D}'$ is $\not\leq$-satisfiable.

Now, recall that $u_i \equiv \tau_\rho(s_i)$ for $1 \leq i \leq n$. If $(s_j \not\in \mathcal{I})$ occurs in $\mathcal{D}'$, then $(\tau_\rho(s_j) \not\in \mathcal{I})$ occurs in $\tau_\rho(\mathcal{D}')$. Since terms rooted by $\_\not\in\mathcal{I}$ are normal forms w.r.t. $\mathcal{R}_{G^!}$, $(\tau_\rho(s_j) \not\in \mathcal{I})$ occurs in $\sigma(\mathcal{C}_t)$. And, since $\sigma(\mathcal{C}_t) \sqsubseteq \sigma(\mathcal{C})$ (recall that $w \not\in \mathcal{I} \sqsubseteq w \not\in \mathcal{I}$ only), $(\tau_\rho(s_j) \not\in \mathcal{I})$ occurs also in $\sigma(\mathcal{C})$.

If $(s_j \not\in \mathcal{I})$ does not occur in $\mathcal{D}'$, we have that $\langle G^!, \mathcal{D}' \rangle \vdash (s_j \in \mathcal{L})$. Since $\mathcal{C}$ does not contain any constraint of the form $\_\in\mathcal{L}$, $\mathcal{C}_t$ neither. Since $\tau_\rho(\mathcal{D}') \rightarrow^!_{\phi_\mathcal{G}, R^{-1}_{G^!}, E_\mathcal{G}} \sigma(\mathcal{C}_t)$, we have that any constraint of the form $w \in \mathcal{L}$ appearing in $\tau_\rho(\mathcal{D}')$ is deducible from $\sigma(\mathcal{C}_t)$, i.e., $\langle G^!, \sigma(\mathcal{C}_t) \rangle \vdash (w \in \mathcal{L})$ for each $w \in \mathcal{L}$ in $\tau_\rho(\mathcal{D}')$. Thus, $\langle G^!, \tau_\rho(\mathcal{D}') \rangle \vdash (\tau_\rho(s_j) \in \mathcal{L})$ implies $\langle G^!, \sigma(\mathcal{C}_t) \rangle \vdash (\tau_\rho(s_j) \in \mathcal{L})$. Finally, by Theorem 6, $\langle G^!, \sigma(\mathcal{C}) \rangle \vdash (\tau_\rho(s_j) \in \mathcal{L})$. $\square$

Now we state and prove the following result for the protocol rules.

Theorem 8 *If $G^!$ is the fixpoint w.r.t. $\Rrightarrow_{\mathcal{P},G,s}$ of a grammar G. Let SS be a term of sort* State*. If SS is not $G^!$-safe, then for all SS' such that $SS \rightsquigarrow^\bullet_{\sigma,\phi_\mathcal{P},R^{-1}_\mathcal{P},E_\mathcal{P}} SS'$, SS' is not $G^!$-safe.*

Proof. Recall that SS being not $G^!$-safe means that there is a term t of sort Msg such that $[\ l_1,\ t^-, l_2 \mid l'\ ]$ appears in SS for terms l_1, l_2, l' of sort SMsgList, and for IK being the intruder knowledge in SS, either $(t \not\in \mathcal{I})$ appears in IK or $\langle G^!, filter^{\not\in}(IK) \rangle \vdash (t \in \mathcal{L})$.

Given

$$SS \rightsquigarrow^\bullet_{\sigma,\phi_\mathcal{P},R^{-1}_\mathcal{P},E_\mathcal{P}} SS'$$

by Proposition 1, we have

$$mset(SS) \rightsquigarrow^{\bullet\ \{0,1\}}_{\sigma',\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} mset(SS')$$

where $\sigma' \equiv \sigma{\downarrow_{\mathcal{V}ar(mset(SS))}}$. Let us consider that t is the term of sort Msg making SS not being $G^!$-safe. So, $t \in mset(SS)$. If $mset(SS') \equiv mset(SS)$, then we are done. Otherwise, $mset(SS') \equiv (mset(SS) \setminus \{t\}) \cup \{u_1, \ldots, u_k\}$ such that

$$t \rightsquigarrow^\bullet_{\sigma'',\phi_{S\mathcal{P}},R^{-1}_{S\mathcal{P}},E_{S\mathcal{P}}} u_1, \ldots, u_k$$

where $\sigma'' \equiv \sigma'{\downarrow_{\mathcal{V}ar(t)}}$. Then, by Lemma 4, there is u_i such that $\langle G^!, \sigma(filter^{\not\in}(IK)) \rangle \vdash (u_i \in \mathcal{L})$ or $(u_i \in \mathcal{L})$ occurs in $\sigma(filter^{\not\in}(IK))$. Therefore, the conclusion follows. $\square$

And our main theorem in this section is the following one.

Theorem 9 (Grammar Unreachability) *Let $\mathcal{P}$ be a protocol. Let $G^!$ be the fixpoint of a grammar G. Let SS be a* State *term. If SS is not $G^!$-safe,*

then there is no substitution σ *such that* $SS \leadsto^{\bullet}_{\sigma,\phi_{\mathcal{P}},R_{\mathcal{P}}^{-1},E_{\mathcal{P}}} SS'$ *where* SS' *is an initial state term.*

Proof. To prove this theorem we use the fact that an intruder does not know in an initial state, see Definition 5, any of the terms belonging to the grammars generated in the language. Then, the conclusion follows by repeated application of Theorem 8. □

7 Concluding Remarks

We have given a precise rewriting-based formalization of the NPA reachability analysis and language generation mechanisms that we call Maude-NPA. And we have illustrated its use by means of a well-known protocol. We have implemented both the reachability analysis and the grammar generation inference systems based on rewriting and narrowing in the Maude rewriting logic language (Clavel et al., 2002). This prototype has been used to produce all the grammar-generation examples in the paper. We have also proved several meta-logical properties of the inference system, in particular that terms produced by the grammar-generation algorithm represent unreachable states in the reachability analysis.

As pointed out in the Introduction, this work is a first step within a longer-term research project to use NPA-like mechanisms in the analysis of protocols for which attacks may make use of the algebraic properties of underlying cryptographic functions. Much work remains ahead including:

(1) Formalization of NPA's other techniques for search space reduction, including various types of partial order reduction.
(2) Generalization of our inference system to handle equational theories for the underlying cryptography; this should take the form of a modular inference system in which such equational theories are a parameter.
(3) Based on (1) and (2) above, development of a next-generation tool based on the generalized inference system and having a rewriting-based implementation.
(4) Furthermore, the meta-logical properties of the current inference system and of its generalization based on their precise rewriting semantics should be systematically studied. Besides proving the types of theorems that we have proved in this paper, but for more general types of equational theories, we also want to characterize the types of languages and protocols for which termination of the reachability analysis process is guaranteed, and to characterize the conditions under which the language generation process itself terminates. If successful, this will provide us with a class of protocols for which we can guarantee that NPA-style reachability analysis

always returns an answer.

Acknowledgments We thank David Basin, Hubert Comon, Jean Goubault-Larrecq, Luca Viganò, and the anonymous referees for the useful remarks and suggestions which helped us to improve the paper.

Santiago Escobar has been partially supported by the EU (FEDER) and Spanish MEC TIN-2004-7943-C04-02 project, the "Generalitat Valenciana" under grant GV06/285, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project.

References

Baader, F., Snyder, W., 2001. Unification theory. In: Robinson, A., Voronkov, A. (Eds.), Handbook of Automated Reasoning. Vol. 1. Elsevier Science, Ch. 8, pp. 445–532.

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

Bistarelli, S., Cervesato, I., Lenzini, G., Martinelli, F., 2005. Relating multiset rewriting and process algebras for security protocol analysis. Journal of Computer Security 13 (1), 3–47.

Blanchet, B., 2001. An efficient cryptographic protocol verifier based on prolog rules. In: 14th IEEE Computer Security Foundations Workshop (CSFW-14 2001). IEEE Computer Society, pp. 82–96.

Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M., 2003a. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In: 23^{rd} Conference on Foundations Software Technology and Theoretical Computer Science. Vol. 2914 of Lecture Notes in Computer Science. pp. 124–135.

Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M., 2003b. An NP decision procedure for protocol insecurity with XOR. In: 18^{th} Annual IEEE Symposium on Logic in Computer Science (LICS '03).

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J., 2002. Maude: specification and programming in rewriting logic. Theoretical Computer Science 285, 187–243.

Comon-Lundh, H., Shmatikov, V., 2003. Intruder deductions, constraint solving and insecurity decision in presence of exclusive-or. In: 18^{th} Annual IEEE Symposium on Logic in Computer Science (LICS '03). pp. 271–280.

Dershowitz, N., Mitra, S., Sivakumar, G., 1992. Decidable matching for convergent systems (preliminary version). In: Kapur, D. (Ed.), 11th International Conference on Automated Deduction (CADE-11). Vol. 607 of Lecture Notes in Computer Science. Springer, pp. 589–602.

Dolev, D., Yao, A., 1983. On the security of public key protocols. IEEE Transaction on Information Theory 29 (2), 198–208.

Escobar, S., Meadows, C., Meseguer, J., 2005. A rewriting-based inference system for the NRL Protocol Analyzer: Grammar generation. In: Küsters, R., Mitchell, J. (Eds.), Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering, FMSE 2005. ACM, pp. 1–12.

Fabrega, F. J. T., Herzog, J. C., Guttman, J., 1999. Strand spaces: Proving security protocols correct. Journal of Computer Security 7, 191–230.

Genet, T., Klay, F., 2000. Rewriting for cryptographic protocol verification. In: McAllester, D. A. (Ed.), 17th International Conference on Automated Deduction (CADE-17). Vol. 1831 of Lecture Notes in Computer Science. Springer, pp. 271–290.

Heather, J., Schneider, S., 2005. A decision procedure for the existence of a rank function. Journal of Computer Security 13 (2), 317–344.

Hullot, J., 1980. Canonical forms and unification. In: Bibel, W., Kowalski, R. (Eds.), 5^{th} Conference on Automated Deduction. Vol. 87 of Lecture Notes in Computer Science. Springer, pp. 318–334.

Kapur, D., Narendran, P., 1987. Matching, Unification and Complexity. ACM SIGSAM Bulletin 21 (4), 6–9.

Lowe, G., 1996. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Tools and algorithms for construction and analysis of systems (TACAS '96). Vol. 1055 of Lecture Notes in Computer Science. Springer, pp. 147–166.

Meadows, C., 1992. Applying formal methods to the analysis of a key management protocol. Journal of Computer Security 1 (1).

Meadows, C., 1996a. Analyzing the needham-schroeder public-key protocol: A comparison of two approaches. In: Bertino, E., Kurth, H., Martella, G., Montolivo, E. (Eds.), Proceedings of the 4th European Symposium on Research in Computer Security - ESORICS 96. Vol. 1146 of Lecture Notes in Computer Science. Springer, pp. 351–364.

Meadows, C., 1996b. Language generation and verification in the NRL Protocol Analyzer. In: Proceedings of the 9th IEEE Computer Security Foundations Workshop. IEEE Computer Society Press, pp. 48–61.

Meadows, C., 1996c. The NRL Protocol Analyzer: An overview. The Journal of Logic Programming 26 (2), 113–131.

Meadows, C., 2000. Invariant generation techniques in cryptographic protocol analysis. In: Proceedings of the 13th Computer Security Foundations Workshop. IEEE Computer Society Press.

Meadows, C., Cervesato, I., Syverson, P., 2004. Specification of the Group Domain of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer. Journal of Computer Security 12 (6), 893–932.

Meseguer, J., 1992. Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96 (1), 73–155.

Meseguer, J., 1998. Membership algebra as a logical framework for equational specification. In: Parisi-Presicce, F. (Ed.), Proc. WADT'97. Springer LNCS

1376, pp. 18–61.
Meseguer, J., Thati, P., 2004. Symbolic reachability analysis using narrowing and its application to the verification of cryptographic protocols. In: Martí-Oliet, N. (Ed.), Proc. 5th. Intl. Workshop on Rewriting Logic and its Applications. ENTCS, Elsevier.
Millen, J., Shmatikov, V., 2001. Constraint solving for bounded-process cryptographic protocol analysis. In: 8^{th} ACM Conference on Computer and Communications Security (CCS '01). pp. 166–175.
Needham, R., Schroeder, M., December 1978. Using encryption for authentication in large networks of computers. Communications of the ACM 21 (12), 993–999.
Peled, D., 1998. Ten years of partial order reduction. In: 10th conference on computer-aided verification (CAV'98). Vol. 1427 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 17–28.
Stubblebine, S., Meadows, C., 2000. Formal characterization and automated analysis of known-pair and chosen-text attacks. IEEE Journal on Selected Areas in Communications 18 (4), 571–581.
TeReSe (Ed.), 2003. Term Rewriting Systems. Cambridge University Press, Cambridge.
Weidenbach, C., 1999. Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (Ed.), 16th International Conference on Automated Deduction (CADE-16). Vol. 1632 of Lecture Notes in Computer Science. Springer, pp. 314–328.

A The Narrowing Relation $\leadsto^{\bullet}_{\sigma,\phi,R,E}$

The narrowing relation $\leadsto^{\bullet}_{\sigma,\phi,R,E}$ is entirely similar to the narrowing modulo E relation $\leadsto_{\sigma,\phi,R,E}$, except that the unification algorithm modulo E and its corresponding set of unifiers are modified as explained below, and the variables of sort Fresh are treated in a special way.

$t \overset{p}{\leadsto}{}^{\bullet}_{\sigma,\phi,R,E} t'$ if there is a non-ϕ-frozen position $p \in \mathcal{P}os_{\Sigma}(t)$, a (possibly renamed) rule $l \to r$ in R such that $\mathcal{V}ar(t) \cap (\mathcal{V}ar(l) \cup \mathcal{V}ar(r)) = \emptyset$, and a E-unifier $\sigma \in CSU^{\bullet}_{E,\phi}(t|_p = l, W)$ for $\mathcal{V}ar(t) \cup \mathcal{V}ar(l) \cup \mathcal{V}ar(r) \subseteq W$ such that $t' = \sigma(t[r]_p)$ and $\mathcal{V}ar_{\mathsf{Fresh}}(t) \cup \mathcal{V}ar_{\mathsf{Fresh}}(l) \cup \mathcal{V}ar_{\mathsf{Fresh}}(r) \not\subseteq \mathcal{D}om(\sigma)$

where the set of unifiers $CSU^{\bullet}_{E,\phi}(u = v, W)$ is defined by

$\sigma{\downarrow_V} \in CSU^{\bullet}_{E,\phi}(u = v, W)$ for $V = \mathcal{V}ar(u) \cup \mathcal{V}ar(v)$ and $V \subseteq W$ if $u \,{}^{\bullet}\!\simeq v \leadsto^{!}_{\mathcal{B},\sigma,\phi,{}^{\bullet}\vec{E}} True$, where we add the frozenness requirement $\phi({}^{\bullet}\!\simeq) = \{1\}$ and we assume that the equations E can be viewed as finite set of rewrite rules $\vec{E}$, and where ${}^{\bullet}\vec{E}$ is the set of rewrite rules ${}^{\bullet}\vec{E} = \vec{E} \cup \{x \,{}^{\bullet}\!\simeq x \to True\}$.

In other words, in this modified procedure, when unifying two terms u and v modulo the equations E by basic narrowing with the rules $\vec{E}$, we begin with the expression $u \stackrel{\bullet}{\simeq} v$ and try to reach the term $True$, but the frozenness of $\stackrel{\bullet}{\simeq}$ does not allow the rules in $\vec{E}$ to be applied to the term u: they can only be applied to v. Only in the end, by narrowing with the rule $x \stackrel{\bullet}{\simeq} x \rightarrow True$, is a final unification of u with the narrowed right term performed.

The reason for using this modified narrowing relation $\rightsquigarrow^{\bullet}_{\sigma,\phi,R,E}$ in the NPA is that when narrowing from input terms $t_1, \ldots, t_m$ to output terms $s_1, \ldots, s_k$, the terms $t_1, \ldots, t_m$ are assumed to be strongly $\rightarrow_{\vec{E}}$-irreducible, whereas the output terms $s_1, \ldots, s_k$ are not necessarily strongly $\rightarrow_{\vec{E}}$-irreducible. This is because the terms $t_1, \ldots, t_m$ represent messages already received by a principal so that encryptions/decryptions have already been applied and, therefore, the terms have been simplified. Instead, the terms $s_1, \ldots, s_k$, represent newly produced terms (messages) that have not yet been simplified.