A Rewriting-Based Inference System for the NRL Protocol Analyzer: Grammar Generation

Santiago Escobar[*]
sescobar@dsic.upv.es
Technical University of Valencia
Valencia, Spain

Catherine Meadows
meadows@itd.nrl.navy.mil
Naval Research Laboratory
Washington, DC, USA

José Meseguer
meseguer@cs.uiuc.edu
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. It probably outranks any of the existing tools in the sheer range of the types of attacks it is able to model and discover. However, the techniques in NPA lack an independent formal specification and model, and instead are closely intertwined with other NPA features. The main contribution of this paper is to rectify this problem by giving for the first time a precise formal specification of one of the main features of the NPA inference system: its grammar-based techniques for invariant generation, as well as a backwards reachability analysis method that captures some of the key features of the NPA. 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.

Categories and Subject Descriptors

D.2.4 [**Software**]: Software Engineering—*Software/Program Verification*; D.3.1 [**Software**]: Programming Languages—*Formal Definitions and Theory*; F.3.1 [**Theory of Computation**]: Logics and Meanings of Programs—*Specifying and Verifying and Reasoning About Programs*; F.4.2 [**Theory of Computation**]: Mathematical Logic and Formal Languages—*Grammars and Other Rewriting Systems* ; F.4.3 [**Theory of Computation**]: Mathematical Logic and Formal Languages—*Formal Languages*

[*]S. Escobar has been partially supported by the EU (FEDER) and the Spanish MEC, under grant TIN 2004-7943-C04-02.

General Terms

Languages, Security, Theory, Verification

Keywords

Formal Methods, Protocol Verification, Rewriting Logic

1. INTRODUCTION

The NRL Protocol Analyzer (NPA) [11] 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. It probably outranks any of the existing tools in the sheer range of the types of attacks it is able to model and discover, ranging from the discovery of an authentication attack based on the cancellation properties of encryption and decryption [9], to the reproduction of Bellovin's attack on a version of the Encapsulating Security Protocol that used cipher block chaining [19] to the discovery of a sophisticated type confusion attack [13] that resulted in the redesign of a draft for a protocol standard.

One of the reason that the NPA has been able to discover and reproduce such a wide variety of attacks is its ability to use inductive techniques to drastically limit the size of an originally infinite search space. The idea, described in detail in [10], is to inductively generate formal languages. A user starts by giving the NPA a term that he/she wishes to prove that the intruder can't learn. This term is called a *seed term.* The NPA then generates a language containing the seed term such that, if the intruder learns a member of the language, then it must have already known a member of the language, thus inductively proving the entire language unreachable. The NPA does this by starting out with a language containing a single production describing the seed term. It then generates a set of terms that the intruder must have already known in order to be able to produce the seed term, and uses such a set of terms to construct new language productions. It then tests each production to see that 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 the case, and tests the new productions and retests the old productions in the same fashion. It continues in this way, generating new production rules and testing them, until it reaches a fixpoint.

Example 1. To give a simple example of how this works, consider a protocol with only two operations, encryption, represented by $e(K, X)$, and decryption $d(K, X)$, that satisfy the equation $d(K, e(K, X)) = X$ and $e(K, d(K, X)) = X$. 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. In order to keep the example simple, we assume that intruders do not perform operations themselves, 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 search, so we ask what rules could produce m, and how. According to the honest principal rule, we have that the intruder can learn m only if it previously knows $e(k, m)$. 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 $e(k, e(k, m))$. We see a pattern emerging, which suggests the set of terms belonging to the following formal language L.

$$\begin{array}{l} (g0.0)\ m \in L \\ (g0.1)\ W{\in}L \mapsto e(k, W){\in}L \end{array}$$

We now want to verify that intruder knowledge of any irreducible term of L implies previous knowledge of some member of L. We do this by running the NPA first on m, and verifying that it requires intruder knowledge of $e(k, m)$, which is a member of L by $g0.0$ and $g0.1$. We then run the NPA on $e(k, W)$, and verify that it requires previous intruder knowledge of $e(k, e(k, W))$, which is also a member of L, by $g0.1$ and the fact that $W \in L$. ■

The types of inductive invariants thus generated have been shown [12] to be related to structures used in other formalisms, such as strand space ideals [7] and rank functions [8]. Indeed, we believe that many of the techniques that have been developed for language generation for the NPA could be applied with profit to these other systems. Moreover, the NPA's reliance on rewriting and narrowing suggests that it could by profitably compared with other protocol analysis systems that rely on rewriting. However, the adaptation 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 where closely intertwined with other NPA features.

The main contribution of this paper is to rectify this problem by giving for the first time a precise formal specification of the NPA's grammar-based techniques for invariant generation, as well as a backwards reachability method, which captures many of the key features of NPA backwards reachability. This formal specification is given within the well-known rewriting framework [20, 14] 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. Besides allowing for a formal specification in a system accessible to a large community of researchers, we expect this approach to have the added benefit of allowing us to make use of theorems from rewriting logic to prove open conjectures about properties of the NPA inference system. Indeed, we are already working on a proof of termination of a key portion of the NPA language generation system. We believe also that a formal specification and proof of correctness of the NPA formal language specification should prove helpful in simplifying the rather complex language generation process, as we expect to be able to use such a proof to assist us in proving that simplifications preserve correctness.

Besides the above-mentioned related work on inductive invariants [7, 8], our 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 [1]. 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 [5, 17, 3, 2]. Indeed, as pointed out in [16], narrowing can be viewed as a general mechanism unifying many of those analyses. Our rewriting semantics makes clear that the NPA tool —and its planned extension to support 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 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 rewriting-based analysis tool that takes into account the algebraic properties of the underlying cryptographic functions. A precise specification of the semantics of the 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, the formal specification will also provide a good basis for a mathematical proof of the correctness of both the NPA inference system and this next-generation tool.

After some preliminaries in Section 2, we informally explain how the NPA tool finds grammars defining languages in Section 3. In Section 4, we introduce some notation for protocols and grammars. In Section 5, we show how grammars are generated and how they can be used to cut down the search space in reachability analysis. We use the Needham-Schroeder Protocol [18] as the guiding example. We conclude in Section 6.

2. PRELIMINARIES

This section gives the definitions of 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 [20, 14] 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 relation $\Rightarrow: T \times T$ on elements T (e.g. $\to_R: \mathcal{T}_\Sigma(\mathcal{X}) \times \mathcal{T}_\Sigma(\mathcal{X})$), we say that the element $a \in T$ is $\Rightarrow$-*irreducible* if there is no other element $b \in T$ such that $a \Rightarrow b$. We write the transitive and reflexive closure of $\Rightarrow$ as $\Rightarrow^*$. We write $a \Rightarrow^! b$ to denote that $a \Rightarrow^* b$ and b is $\Rightarrow$-irreducible.

We say the relation $\Rightarrow$ is *terminating* if there is no infinite sequence $a_1 \Rightarrow a_2 \Rightarrow \cdots \Rightarrow \cdots$. We say the relation $\Rightarrow$ 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 $(S^* \times S)$-indexed family of $\{\Sigma_{w,s}\}_{(w,s)\in S^*\times S}$ operations. We denote $f \in \Sigma_{w,s}$ by $f : w \to s$. In this paper, we use lowercase letters $f, g, h, \ldots$ to denote symbols in Σ. We define a relation $\simeq$ on S as the smallest equivalence relation such that $s \leq s'$ implies $s \simeq s'$. 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 every sort s we assume that there is a sort $[s]$ such that $s \simeq s'$ implies $s' \leq [s]$. Furthermore, for each $f : s_1 \times \ldots \times s_n \to s$ we assume there is also an $f : [s_1] \times \ldots \times [s_n] \to [s]$. We require the signature Σ to be *sensible*, i.e., whenever we have $f : w \to s$ and $f : w' \to s'$ with w, w' of equal length, then $w \simeq w'$ implies $s \simeq s'$.

A Σ-*algebra* is defined by an S-indexed family of sets $A = \{A_s\}_{s\in S}$ such that $s \leq s'$ implies $A_s \subseteq A_{s'}$, and for each function $f : w \to s$ with $w = s_1 \times \ldots \times s_n$ a function $f_A^{w,s} : A_{s_1} \times \ldots \times A_{s_n} \to A_s$. Further, we require that subsort overloaded operations agree, i.e., for each $f : w \to s$ and $(a_1, \ldots, a_n) \in A^w$ we require $f_A^{w,s}(a_1, \ldots, a_n) = f_A^{[w],[s]}(a_1, \ldots, a_n)$, where if $w = s_1 \times \ldots \times s_n$, then $[w] = [s_1] \times \ldots \times [s_n]$. We assume a family $\mathcal{X} = \{\mathcal{X}_s\}_{s\in S}$ of infinite sets of variables such that $s \not\simeq s'$ implies $\mathcal{X}_s \cap \mathcal{X}_{s'} = \emptyset$, and the variables in $\mathcal{X}$ are different from constant symbols in Σ. In this paper, 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 $\mathcal{T}_{\Sigma_s}$ and $\mathcal{T}_\Sigma(\mathcal{X})_s$, 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 $t \in \mathcal{T}_\Sigma(\mathcal{X})$. A *non-variable* term is simply a 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. The root of a term is at 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 variables to terms of the same sort, and which is different from the identity 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 $\sigma = [X_1/t_1, \ldots, X_n/t_n]$. 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 $\mathcal{R}an(\sigma) \cap V = \emptyset$.

A Σ-*equation* is an expression of the form $t =_E t'$ where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_{[s]}$ for an appropriate $[s]$. Order-sorted equational logic has a sound and complete inference system $E \vdash_\Sigma$ (see [15]) inducing a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$: $t =_E t'$ if and only if $E \vdash_\Sigma t = t'$; under the assumption that all sorts S in Σ are non-empty, i.e., $\forall s \in S : \mathcal{T}_\Sigma(\mathcal{X})_s \neq \emptyset$.

The E-*subsumption* preorder $\preceq_E$ on $T_\Sigma(X)$ is defined by $t \preceq_E t'$ (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 in the following way. We say $\sigma \preceq_E \sigma'$ if $\sigma(X) \preceq_E \sigma'(X)$ for each $X \in \mathcal{X}$.

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

A *rewrite rule* is an expression of the form $l \to r$ where $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_{[s]}$ for an appropriate $[s]$. 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 $\phi : \Sigma \to \mathcal{P}(\mathcal{N}at)$ specifies the *frozen* arguments $\phi(f) \subseteq \{1, \ldots, ar(f)\}$. We say that position p in t is *frozen* if $\exists q < p$ such that $p = q.i.q'$ and $i \in \phi(root(t|_q))$. In this paper, we do not impose the condition $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ for each rule $l \to r \in \mathcal{R}$.

We define the *one-step rewrite relation* on $\mathcal{T}_\Sigma(\mathcal{X})$ as follows: $t \xrightarrow{p}_R t'$ (or simply $\to_R$) if there is a nonfrozen position $p \in \mathcal{P}os_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p = \sigma(l)$ and $t' = t[\sigma(r)]_p$. We say a rewrite theory is terminating (confluent) if the relation $\to_R$ is terminating (confluent). The *one-step R, E-rewrite relation* on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as follows: $t \xrightarrow{p}_{R,E} t'$ (or simply $\to_{R,E}$) if there is a nonfrozen position $p \in \mathcal{P}os_\Sigma(t)$, a rule $l \to r$ in R, 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: $t \overset{p}{\leadsto}_{\sigma,R} t'$ (or simply $\leadsto_{\sigma,R}$ or $\leadsto_R$) if there is a nonfrozen position $p \in \mathcal{P}os_\Sigma(t)$, a (possibly renamed) rule $l \to r$ in R, and a unifier σ such that $\sigma(t|_p) = \sigma(l)$ and $t' = \sigma(t[r]_p)$. The *R, E-narrowing relation* on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as follows, where we assume there is a complete unification algorithm for E: $t \overset{p}{\leadsto}_{\sigma,R,E} t'$ (or simply $\leadsto_{\sigma,R,E}$ or $\leadsto_{R,E}$) if there is a nonfrozen position $p \in \mathcal{P}os_\Sigma(t)$, a (possibly renamed) rule $l \to r$ in R where we assume $\mathcal{V}ar(t) \cap Var(l) = \emptyset$, and $\sigma \in CSU(t|_p =_E l, V)$ for a set of variables V containing $\mathcal{V}ar(t)$ and $\mathcal{V}ar(l)$, such that $t' = \sigma(t[r]_p)$. When the equations in the equational theory E can be viewed as terminating and confluent rewrite rules $\vec{E}$, then the narrowing relation $\leadsto_{\hat{E}}$ with $\hat{E} = \vec{E} \cup \{x \simeq x \to True\}$ provides a sound and complete unification algorithm, i.e. $\sigma \in CSU(t =_E t', W)$ iff $t \simeq t' \leadsto^!_{\sigma,\hat{E}} True$.

3. HOW THE NPA FINDS LANGUAGES

In this section, we describe the NPA's language finding strategy in broad outline, using Example 1 as an illustration. The NPA starts out with a simple seed term. This seed term defines an initial language, with only one grammar rule, which says that the seed term is in the language. In Example 1, this seed term is the rule $(g0.0)$, i.e., $\emptyset \mapsto m{\in}L$.

The NPA strategy for generating languages consists of three stages: the term generation stage, the rule verification stage, and the rule generation stage.

In the term generation stage, the 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. Thus, in Example 1, there is only one path to the seed term m, the path in which the intruder sends $e(k, m)$ to an honest principal.

In the rule verification stage, the NPA examines each path and determines which paths require the intruder to know a member of the language in order to produce the goal. It removes those paths from consideration. In the first iteration of Example 1 the single path generated does 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 NPA looks at the remaining paths, and generates a new set of language rules according to a set of heuristics. One such heuristic says that, if a path contains a term containing a word in the language as a subterm, replace that by a variable W and a condition saying that W is in the language to obtain a new language rule. In Example 1, this heuristic generates the rule $(g0.1)$, i.e., $W{\in}L \mapsto e(k, W){\in}L$.

After the rule generation stage, the 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.

In Example 1, the NPA successfully defines a language. In the second iteration, the NPA shows that the sole path generated to 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 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 the broad outline of the work we present below.

4. PROTOCOL NOTATION AND GRAMMARS

In this section we introduce the protocol notation that we use, and the notation used to specify grammars.

In this paper we consider two things, 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 the NPA to *reduce the search space* obtained by narrowing; this search space is generated when performing backwards reachability analysis from a bad state, i.e, a state representing that an intruder was able to attack the protocol. In what follows we will describe our *generic* notation for grammars and protocols by two signatures $\Sigma_{\mathcal{G}}$ and $\Sigma_{\mathcal{P}}$, with $\Sigma_{\mathcal{G}} \subseteq \Sigma_{\mathcal{P}}$. The operators in these signatures 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. Similarly, special algebraic properties of a protocol may be specified with equations E, and there will be a set of protocol-specific rules $R_{\mathcal{P}}$. Our inference system will therefore be parametric on the protocol-specific syntax, equations, and rules, that is on the triple $(\Sigma, E, R_{\mathcal{P}})$.

4.1 Protocol Notation

To facilitate the exposition, we use a simplified notation for protocol rules. In the NPA, protocol rules are specified in terms of state transitions that describe the input as messages received by a principal and values of its local state variables, and the output as messages sent by that principal and new values assigned to its local state variables. For the purposes of language generation though, we are mainly interested in what terms a principal must have received in the past in order to generate a term. Thus we use a more abstract definition of protocol rule that captures this information without concerning ourselves overmuch with local state information, except for final states.

We use a notation similar to that of strand spaces [7], in which a local execution of a protocol by a principal is indicated by a graph in which nodes representing input messages are assigned a negative sign, and nodes representing output messages are assigned a positive sign. We differ from the strand space model in that we also assign a final positive node to each strand representing the final state, and an initial negative node to each strand representing the initial state. We break each strand up into substrands, with one substrand for each place where a negative node directly precedes the following node. We represent the substrand as a protocol rule of the form $u_1, \ldots, u_n \to v_1, \ldots, v_k$ where $v_1, \ldots, v_k, u_1, \ldots, u_n \in \mathcal{T}_\Sigma(\mathcal{X})$, $u_1, \ldots, u_n$ are all the negative nodes preceding, $v_1, \ldots, v_n$ are positive nodes, and no negative node directly precedes v_2 through v_n. Thus, the left-hand side of a protocol rule describes what the intruder must know in order to produce any one of $v_1, \ldots, v_n$. If he/she observes or creates $u_1, \ldots, u_n$, he/she can send them to the principal in the appropriate order, and that principal will then produce $v_1, \ldots, v_n$, and if the principal does not receive $u_1, \ldots, u_n$ he/she will not produce $v_1, \ldots, v_n$.

We now use the protocol rules to define the protocol rewrite theory formally below. The protocol rewrite theory is defined as $\mathcal{P} = (\Sigma_{\mathcal{P}}, \phi_{\mathcal{P}}, E_{\mathcal{P}}, R_{\mathcal{P}})$ where $\Sigma_{\mathcal{P}} = \Sigma_{\mathcal{G}} \cup \{\_,\_\}$, $R_{\mathcal{P}}$ contains the concrete protocol rules, $E_{\mathcal{P}}$ contains the equational theory E of the concrete protocol problem, and $\phi_{\mathcal{P}}(\_{\mapsto}\_) = \{1\}$ and $\phi_{\mathcal{P}}(f) = \phi_{\mathcal{G}}(f)$ for the remaining $f \in \Sigma_{\mathcal{P}}$ ($\phi_{\mathcal{G}}$ is defined in Section 4.2 below). Recall that the frozenness restriction $\phi_{\mathcal{P}}(\_{\mapsto}\_) = \{1\}$ implies that the first argument of the symbol $\_{\mapsto}\_$ is frozen (symbol $\_{\mapsto}\_$ describes a grammar rule, as shown in Section 4.2 below). The operator $\_,\_ : MsgSet \times MsgSet \to MsgSet$ is a set union operator that is associative, commutative, and has identity $\emptyset$, and we assume that there is a subsort relation $Msg < MsgSet$. We define the equational theory $E_{\mathcal{P}}$ as the encryption equations E explained below, together with the equations for associativity, commutativity and identity of the $\_,\_$ operator. For w a term of sort $MsgSet$ and t a term of sort Msg, we write $t \in w$ iff there exist u, v such that $w \equiv u, t, v$. Note that the rewrite theory $R_{\mathcal{P}}$ is used in the NPA in a backwards way, i.e., we will therefore speak of $\leadsto_{R_{\mathcal{P}}^{-1}}$ where $R_{\mathcal{P}}^{-1} = \{r \to l \mid l \to r \in R_{\mathcal{P}}\}$.

Example 2. Consider the following signature Σ for the Needham-Schroeder Protocol [18]:

$$pk : Name \times Msg \to Enc \quad sk : Name \times Msg \to Enc$$
$$final : Name \times Msg \to FinalState$$
$$c : \to Name \qquad n : Name \times RandomInt \to Nonce$$
$$\_;\_ : Msg \times Msg \to Msg$$

where c plays the role of the intruder's key, together with the following subsort relations

$$Name\ Nonce\ Enc\ FinalState < Msg$$

and the encryption/decryption equations E:

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

In the following, we will use letters A, B for variables of sort $Name$, letters r, r' for variables of sort $RandomInt$, and letters M, M_1, M_2, Z for variables of sort Msg; whereas letters X, Y represent also variables but their sort will depend on the concrete position in a term.

In the rewrite theory $\mathcal{P} = (\Sigma_{\mathcal{P}}, \phi_{\mathcal{P}}, E_{\mathcal{P}}, R_{\mathcal{P}})$, with rules $R_{\mathcal{P}}$ listed below (rules p1-p10), the following rules correspond to the informal description of the protocol in [18]:

p1 $\emptyset \to pk(B,\ A; n(A, r))$
This rule represents A initiating the protocol by sending its name and a nonce encrypted with B's public key to B.

p2 $pk(A, n(A, r); Z) \to pk(B, Z)$
p3 $pk(A, n(A, r); Z) \to final(A, B; n(A, r))$
These rules represent A receiving B's response and sending its final message.

p4 $pk(B, A; Z) \to pk(A, Z; n(B, r))$
This rule represents B receiving A's first message, checking that it is the public key encryption of A's name concatenated with some value, and then sending to A the concatenation of that value with its own nonce, encrypted with A's public key.

p5 $pk(B, A; Z), pk(B, n(B, r')) \to final(B, A; n(B, r'))$
This rule represents B verifying that the final message that it receives is its nonce encrypted with its public key, and reaching its final state. Note that this rule also gives the first message that B must receive as well.

The following rules describe the intruder's ability to concatenate, deconcatenate, encrypt and decrypt according to the Dolev-Yao attacker's capabilities [6]:

p6 $M_1, M_2 \to M_1; M_2$

p7 $M_1; M_2 \to M_1$

p8 $M_1; M_2 \to M_2$

p9 $M \to sk(c, M)$

p10 $M \to pk(Y, M)$

Note that rule (p9) allows intruder's decryption only using his own key c. Note also that rules (p3) and (p5) are not used for grammar generation but only for reachability analysis. ■

4.2 Grammar Notation

This section will be concerned solely with describing how languages are defined, but to motivate this it will help to understand how languages are generated. The basic idea behind generating a language is that we start with a seed term, and use backwards narrowing via the protocol rules to construct a complete set of system states that must precede a state in which the intruder knows that term. Each such state is defined by the terms known by the intruder in that state. We then use heuristics to define language rules that imply that at least one term known by the intruder is in the language. Then we take the term defined by the rule, and use backwards narrowing again to find a complete set of system states, that precedes that term. Some of these states may already contain terms in the language, and for others we may need to introduce new rules using the heuristics.

Grammars are described by means of three basic kinds of constraints. We motivate these as follows. Seed terms may be described in two different ways, either as terms the intruder is not expected to know (such as secret keys), or as the result of performing operations on terms the intruder does not know. An example of the latter case is a term such as $pk(X, Y)$, where Y is not in the intruder's knowledge set, denoted by $(Y \notin \mathcal{I})$. Since we are using backwards search, we will not have a precise picture of the intruder's knowledge set, but we use the convention that, if the intruder learns a term in the future, he/she cannot know it in the present. Thus, in our backwards search, we keep track of the set of terms w that the intruder will learn in the future and conclude that a term t satisfies $t \notin \mathcal{I}$ if $t \in w$. This $(Y \notin \mathcal{I})$ will be our first type of constraint.

Another possibility is that the intruder will be able to learn some instance of the seed term, because they are generated by other principals, so we shall have to introduce some exceptions in our definition of the seed term, e.g. $(Y \not\geq s_1)$, ..., $(Y \not\geq s_2)$. This will be our second type of constraint.

Once we have identified the seed term, we will use it to construct grammar rules that say that a term t is in the language if some subterm s of t is in the language, e.g. $(X; Z) \in L$ if $Z \in L$. For example, we start out with a grammar rule that says that $pk(X, Y)$ is in the language if $(Y \notin \mathcal{I})$. Thus we start by trying to find the conditions under which the intruder knows $pk(X, Y)$, where $(Y \notin \mathcal{I})$. Applying rule ($p10$) in Example 2 (in a backwards way and modulo encryption/decryption equations in E) gives us that 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 $sk(W, T) \in L$ if $T \in L$. This motivates the use of the third constraint, $t \in L$.

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 $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in L$, 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 $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in L$ is that in order for any of the terms $t_1, \ldots, t_n$ to be in the language of the grammar G, say[1] L, then the constraints $c_1, \ldots, c_k$ must be satisfied, i.e., $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in L$ is understood as $t_1 \in L \vee \cdots \vee t_n \in L$ if $c_1 \wedge \cdots \wedge c_k$.

[1]In the paper, we should write $t \in L_G$ to denote whether t is in the language of the grammar G. However, we always clarify the grammar G that it is being used, and thus we simply write $t \in L$.

The signature $\Sigma_{\mathcal{G}}$ is defined as $\Sigma_{\mathcal{G}} = \Sigma \cup \{\_\mapsto\_\} \cup \Sigma_{Ctr}$ where $\_\mapsto\_ : CtrSet \times LCtr \rightarrow GRule$. For sort Ctr, we have the following three kinds of constraints, with subsorts $LCtr\ DCtr\ ICtr < Ctr$:

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

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

(ii) constraints of the form $(t{\not\preceq}s)$ constructed with the symbol

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

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

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

We also have the signature $\Sigma_{Ctr} = \{\ \_,\_\ ,\ \_{\in}L\ ,\ \_{\not\preceq}\_\ ,\ \_{\notin}\mathcal{I}\ \}$. Furthermore, the conditions $c_1, \ldots, c_k$ of a grammar rule can contain exactly one constraint of either type (i) or type (iii) and, moreover, that unique constraint is always of the form $(X{\in}L)$ or $(X{\notin}\mathcal{I})$ (this requirement will be important in Section 5.1.2).

The operator $\_,\_ : CtrSet \times CtrSet \rightarrow CtrSet$ is a set union operator that is associative, commutative, and has identity $\emptyset$, and we assume that there is a subsort relation $Ctr < CtrSet$. We define $ICtrSet$ similarly to $CtrSet$, but using as subsort $ICtr$ instead of Ctr and assuming the subsort relation $ICtrSet < CtrSet$; this will be useful in Section 5.2. For $\mathcal{C}$ a term of sort $CtrSet$ and c a term of sort Ctr, we write $c \in \mathcal{C}$ iff there exist u, v such that $\mathcal{C} \equiv u, c, v$.

Example 3. Continuing Example 2, in the Needham-Schroeder Protocol, we generate the following grammar $G^{!}_{sd_1}$ starting with the seed term $sd_1 = Y{\notin}\mathcal{I} \mapsto X;Y{\in}L$ (we explain how $G^{!}_{sd_1}$ is generated in Example 5):

g1.1 $Z{\in}L \mapsto pk(B,Z){\in}L$

g1.2 $Z{\in}L \mapsto sk(Y,Z){\in}L$

g1.3 $Z{\in}L \mapsto Z;Y{\in}L$

g1.4 $Z{\in}L \mapsto X;Z{\in}L$

g1.5 $Y{\notin}\mathcal{I},\ Y{\not\preceq}n(B,r) \mapsto X;Y{\in}L$

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

To execute a grammar G as a rewrite rule program, we associate to G 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 equations for associativity, commutativity and identity of the $\_,\_$ operator, $\phi_{\mathcal{G}}(\_\mapsto\_) = \{2\}$, $\phi_{\mathcal{G}}(\_,\_) = \phi_{\mathcal{G}}(\_{\in}L) = \phi_{\mathcal{G}}(\_{\not\preceq}\_) = \phi_{\mathcal{G}}(\_{\notin}\mathcal{I}) = \emptyset$, and $\phi_{\mathcal{G}}(f) = \{1, \ldots, ar(f)\}$ for the remaining $f \in \Sigma$. Recall that these restrictions for Σ allow rewrite or narrowing steps only at the top of terms of sort Msg. Note that, since $\phi_{\mathcal{P}}(\_\mapsto\_) = \{1\}$ and $\phi_{\mathcal{G}}(\_\mapsto\_) = \{2\}$, a term $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n){\in}L$ has the first argument frozen when it is rewritten or narrowed using the protocol rules $R_{\mathcal{P}}$, 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 $R_{\mathcal{P}}$ and R_G will be used in different contexts and for different purposes as explained later. Note also that in the NPA R_G is used in a backwards way, i.e., we will consider the relation $\rightarrow_{R_G^{-1}}$.

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

Often, one needs to check whether a term is in the language, say L, generated by one grammar G in the sequence $\mathcal{G}$, i.e., $\mathcal{G} \vdash t{\in}L$. 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 $\mathcal{D}$ and some disequality capabilities for constraints of the form $t{\not\preceq}s$. By definition,

$\langle\mathcal{G},\mathcal{D}\rangle \vdash \mathcal{C}$

iff there is $G \in \mathcal{G}$ s.t. $(\mathcal{C} \rightarrow^{!}_{\mathcal{R}_G^{-1}} \mathcal{C}') \wedge (\mathcal{C}' \sqsubseteq \mathcal{D})$

where, 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 $c_i \sqsubseteq d_j$, where for X a variable of sort Msg and u, t, s terms of sort Msg, by definition,

$$(X{\in}L) \sqsubseteq (X{\in}L)$$

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

$$(u{\not\preceq}t) \sqsubseteq (u{\not\preceq}s) \text{ if } t \preceq s$$

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

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. In the following examples, we write $t \rightarrow_{gi.j^{-1}} s$ to indicate that $t \rightarrow_{R_G^{-1}} s$ is rewritten using a grammar rule $gi.j$ in G.

Example 4. Continuing Example 3, in the Needham-Schroeder Protocol, we also generate the following grammar $G^{!}_{sd_3}$ starting with the seed term $sd_3 = Y{\notin}\mathcal{I} \mapsto pk(X,Y){\in}L$ (we explain how $G^{!}_{sd_3}$ is generated in Example 5):

g3.1 $Z{\in}L \mapsto pk(c,Z){\in}L$

g3.2 $Z{\in}L \mapsto pk(A,n(A,r),sk(B,Z)){\in}L$

g3.3 $Z{\in}L \mapsto sk(Y,Z){\in}L$

g3.4 $Z{\in}L \mapsto Z;Y{\in}L$

g3.5 $Z{\in}L \mapsto X;Z{\in}L$

g3.6 $Z{\notin}\mathcal{I},\ Z{\not\preceq}n(B,r),\ Z{\not\preceq}Z';n(B,r)$
$\mapsto pk(A,Z){\in}L$

g3.7 $Z{\notin}\mathcal{I},\ Z{\not\preceq}n(B,r),\ Z{\not\preceq}Z';n(B,r)$
$\mapsto pk(A,n(A,r);Z){\in}L$

g3.8 $Z{\notin}\mathcal{I},\ Z{\not\preceq}n(B,r),\ Z{\not\preceq}Z';n(B,r)$
$\mapsto pk(A,n(A,r);n(A,r);Z){\in}L$

g3.9 $Z{\notin}\mathcal{I},\ Z{\not\preceq}n(B,r),\ Z{\not\preceq}Z';n(B,r)$
$\mapsto n(A,r);Z{\in}L$

$$\begin{tikzcd}
 & & & & \mathcal{G}_i \Rrightarrow_{\mathcal{P},\mathcal{G}_i,s} \mathcal{G}_{i+1} \arrow[dlll] \arrow[d] \arrow[dr] & & \\
 & newGrammar_s(G,\mathcal{C},\mathcal{H}) \arrow[dl] \arrow[d] \arrow[dr] & & & optimize(G) \arrow[dl] \arrow[d] & \langle \mathcal{C},\mathcal{H},G \rangle \Rrightarrow_{\mathcal{P},\mathcal{G}_i,G^k_j,s} \langle \mathcal{C}',\mathcal{H}',G' \rangle \arrow[d] \arrow[ddr, bend left] & \\
\alpha(G,\mathcal{C}) & \beta(G,\mathcal{C}) & \widehat{G} & removeContrainsts(G) \arrow[dd] & removeRules(G) \arrow[d] & heuristics_{G^k_j,s}(g,\sigma,g') \arrow[dl] & \\
 & & & & \langle \mathcal{G}_i, \mathcal{D} \rangle \vdash \mathcal{C} \arrow[dl] \arrow[d] & & g \leadsto_{\sigma,\mathcal{P},\mathcal{G}_i,G^k_j} g' \arrow[ll] \arrow[dl] \arrow[d] \\
 & & & c \sqsubseteq c' & \rightarrow_{\mathcal{R}^{-1}_G} & \leadsto_{\mathcal{R}^{-1}_{G^k_j}} & \leadsto^{\bullet}_{\sigma,R^{-1}_{\mathcal{P}},E_{\mathcal{P}}}
\end{tikzcd}$$

Figure 1: Dependencies between relations and operators in the Grammar Generation phase

For the following membership test:

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

we have

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

and $Y{\in}L$ is already a premise, thus $pk(c,Y);M_2$ is in the language L. 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 $A;n(A,r) \preceq Z';n(B,r)$. And for the test

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

we have

$$pk(A,n(B,r)){\in}L\ \rightarrow_{g3.6^{-1}}\ \begin{array}{l} n(B,r){\notin}\mathcal{I},\ n(B,r){\not\preceq}n(B',r'), \\ n(B,r){\not\preceq}Z';n(B',r') \end{array}$$

and these constraints are not satisfied, since the term $n(B,r){\notin}\mathcal{I}$ is a premise and there is no unifier θ such that $n(B,r) \equiv \theta(Z';n(B',r'))$ but there is σ such that $n(B,r) \equiv \sigma(n(B',r'))$. ■

5. THE INFERENCE SYSTEM

The outline of the general algorithm of flaw detection in the NPA is as follows.

Input:

1. the signature Σ,
2. the protocol specification $R_{\mathcal{P}}$,
3. a sequence of grammar rules $D = \langle sd_1, \ldots, sd_n \rangle$ called *seed terms*,
4. a term t_{bad} of sort $MsgSet$ describing the undesired (bad) state of the protocol, where for the purpose of this paper, a state[2] is a set of messages and final terms; and
5. a term $t_{\mathcal{I}}$ of sort $ICtrSet$ describing terms to be known by the intruder in the undesired state.

Output: The algorithm tries to deduce whether the protocol is safe for t_{bad} and $t_{\mathcal{I}}$ or not. In the latter case, the algorithm provides an attack trace. When the protocol is safe, the algorithm can often terminate, thanks to the drastic reduction on the search space given by the grammars, but may in some cases loop.

Algorithm: First, build the grammar sequence $\mathcal{G} = \langle G^!_{sd_1}, \ldots, G^!_{sd_n} \rangle$ as the fixpoint of the relation $\mathcal{G}_i \Rrightarrow_{\mathcal{P},\mathcal{G}_i,s} \mathcal{G}_{i+1}$ described in Section 5.1.1 starting with $\mathcal{G}_0 = \langle \{sd_1\}, \ldots, \{sd_n\} \rangle$ for $sd_1, \ldots, sd_n$ the chosen seed terms. This part may not terminate. Second, check reachability of t_{bad} by the protocol using some backwards reachability relation that keeps track of the terms the intruder learns in future. In this paper we use the relation $\langle t_{bad}, \epsilon \rangle \approx\!\!>^*_{\mathcal{P},\mathcal{G}} \langle t', w \rangle$ described in Section 5.2, as an example. The whole algorithm is a semi-decidable process: Either it finds an attack state of the form $\langle \emptyset, w \rangle$ (we return the attack trace w), it terminates with no attack state (we return that the protocol is safe), or it does not terminate (we return unknown).

Example 5. Let us consider again the Needham-Schroeder Protocol of Example 2. The chosen seed terms for this protocol are:

$$\begin{array}{ll} sd_1 = Y{\notin}\mathcal{I} \mapsto X;Y{\in}L & sd_2 = X{\notin}\mathcal{I} \mapsto X;Y{\in}L \\ sd_3 = Y{\notin}\mathcal{I} \mapsto pk(X,Y){\in}L & sd_4 = Y{\notin}\mathcal{I} \mapsto sk(X,Y){\in}L \end{array}$$

The global scheme of the application of the previous algorithm to this example is as follows, where $G^0_{sd_i} = \{sd_i\}$ for $i \in \{1,2,3,4\}$, and $G^!$ represents that the grammar generation process for the single grammar G has reached a fixpoint:

$$\begin{array}{l}
\mathcal{G}_0 = \langle G^0_{sd_1}, G^0_{sd_2}, G^0_{sd_3}, G^0_{sd_4} \rangle \\
\quad \Rrightarrow_{\mathcal{P},\mathcal{G}_0,S1}\ \mathcal{G}_1 = \langle G^1_{sd_1}, G^0_{sd_2}, G^0_{sd_3}, G^0_{sd_4} \rangle \\
\quad \Rrightarrow_{\mathcal{P},\mathcal{G}_1,S1}\ \mathcal{G}_2 = \langle G^!_{sd_1}, G^0_{sd_2}, G^0_{sd_3}, G^0_{sd_4} \rangle \\
\quad \vdots \\
\quad \Rrightarrow_{\mathcal{P},\mathcal{G}_6,S1}\ \mathcal{G}_7 = \langle G^!_{sd_1}, G^!_{sd_2}, G^!_{sd_3}, G^0_{sd_4} \rangle \\
\quad \Rrightarrow_{\mathcal{P},\mathcal{G}_7,S1}\ \mathcal{G}_8 = \langle G^!_{sd_1}, G^!_{sd_2}, G^!_{sd_3}, G^!_{sd_4} \rangle
\end{array}$$

where we perform one step for sd_4, two steps for sd_1 and sd_3, and three for sd_2. Note that $G^!_{sd_1}$ and $G^!_{sd_3}$ are the grammars shown in Examples 3 and 4, respectively, whereas $G^!_{sd_2}$ and $G^!_{sd_4}$ are:

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

[2]More complex definitions of states are possible for the NPA, but this suffices for the purpose of this paper.

g2.6 $Z \notin \mathcal{I},\ Z \not\approx n(B,r) \mapsto c; Z \in L$

g4.1 $Z \in L \mapsto pk(c,Z) \in L$
g4.2 $Z \in L \mapsto sk(A,Z) \in L$
g4.3 $Z \in L \mapsto Z; Y \in L$
g4.4 $Z \in L \mapsto X; Z \in L$
g4.5 $Z \notin \mathcal{I} \mapsto sk(A,Z) \in L$

Then, we can show how the grammars $\mathcal{G} = \langle G^!_{sd_1}, G^!_{sd_2}, G^!_{sd_3}, G^!_{sd_4} \rangle$ can cut many undesired backward narrowing sequences in the reachability analysis phase. For the following state,

$$\langle\ pk(B, n(B,r)),\ \mathit{final}(B, A; n(B,r))\ \rangle$$

the only backwards narrowing step that leads to a reachability analysis sequence not cut by the grammars in $\mathcal{G}$ is:

$$\begin{array}{l}\langle\ pk(B,n(B,r)),\ \mathit{final}(B,A;n(B,r)\ \rangle \\ \quad \rightsquigarrow^*_{\mathcal{P},\mathcal{G}} \langle\ n(B,r),\ pk(B,n(B,r))\ .\ \mathit{final}(B,A;n(B,r))\ \rangle\end{array}$$

For instance, we could also have

$$\begin{array}{l}\langle\ pk(B,n(B,r)),\ \mathit{final}(B,A;n(B,r)\ \rangle \\ \quad \rightsquigarrow^*_{\mathcal{P},\mathcal{G}} \langle\ M_1; pk(B,n(B,r)), \\ \qquad\qquad pk(B,n(B,r))\ .\ \mathit{final}(B,A;n(B,r))\ \rangle\end{array}$$

using protocol rule p8, but

$$\langle G^!_{sd_1}, \mathcal{C} \rangle \vdash M_1; pk(B,n(B,r)) \in L$$

for $\mathcal{C} = (pk(B,n(B,r)) \notin \mathcal{I}, \mathit{final}(B,A;n(B,r)) \notin \mathcal{I})$, since

$$\begin{array}{l}M_1; pk(B,n(B,r)) \in L \\ \quad \rightarrow_{g1.5^{-1}} pk(B,n(B,r)) \notin \mathcal{I}, pk(B,n(B,r)) \not\approx A; n(A,r)\end{array}$$

and constraint $pk(B,n(B,r)) \notin \mathcal{I}$ is already in the premises $\mathcal{C}$ and $pk(B,n(B,r)) \not\approx A; n(A,r)$ is satisfied. Thus, $G^!_{sd_3}$ cuts such backwards narrowing step in the reachability analysis. ■

In Section 5.1 we describe how the grammar sequence $\mathcal{G} = \langle G^!_{sd_1}, \ldots, G^!_{sd_n} \rangle$ is generated from $\mathcal{G}_0 = \langle \{sd_1\}, \ldots, \{sd_n\} \rangle$, and in Section 5.2 we show how attack states are detected through backwards narrowing-based analysis.

5.1 Grammar Generation

Each application of the transformation relation $\Rrightarrow_{\mathcal{P},\mathcal{G}_i,s}$ implies many other auxiliary transformations and tests. We depict in Figure 1 the dependencies between the different relations and operators that we are going to define in this section. In the following, we discuss such relations and operators in a top-down (almost) left-to-right order following Figure 1.

5.1.1 *The Relation* $\mathcal{G}_i \Rrightarrow_{\mathcal{P},\mathcal{G}_i,s} \mathcal{G}_{i+1}$

We generate a new grammar G^{k+1}_j in the sequence $\mathcal{G}_{i+1} = \langle G^!_1, \ldots, G^!_{j-1}, G^{k+1}_j, \ldots, G^{k_n}_n \rangle$ (recall that $G^!$ represents that the grammar generation process for the single grammar G has reached a fixpoint) whose language, say L, will allow us to simplify the flaw detection process in Section 5.2, i.e., the process of reachability analysis by narrowing. The auxiliary relation $\langle \mathcal{C}, \mathcal{H}, G \rangle \Rrightarrow_{\mathcal{P},\mathcal{G}_i,G^k_j,s} \langle \mathcal{C}', \mathcal{H}', G' \rangle$ produces a set of constraints and grammar rules for G^k_j to be included in G^{k+1}_j and which is defined in Section 5.1.4 below. The operator $\mathit{optimize}(G)$ is defined in Section 5.1.3 below. Finally, s is a global strategy parameter that can be instantiated to $S1$ or $S2$, as explained in Section 5.1.6 below. The transformation relation $\Rrightarrow_{\mathcal{P},\mathcal{G}_i,s}$ is defined as follows:

$$\begin{array}{l}\mathcal{G}_i \Rrightarrow_{\mathcal{P},\mathcal{G}_i,s} \mathcal{G}_{i+1} \\ \text{if } \mathcal{G}_i = \langle G^!_1, \ldots, G^!_{j-1}, G^k_j, \ldots, G^{k_n}_n \rangle, \\ \mathcal{G}_{i+1} = \langle G^!_1, \ldots, G^!_{j-1}, G^{k+1}_j, \ldots, G^{k_n}_n \rangle, \\ \text{and } G^{k+1}_j \not\equiv G^k_j; \\ \text{where } \langle \emptyset, \emptyset, G^k_j \rangle \Rrightarrow^!_{\mathcal{P},\mathcal{G}_i,G^k_j,s} \langle \mathcal{C}, \mathcal{H}, \emptyset \rangle \\ \text{and } G^{k+1}_j = \mathit{optimize}(\mathit{newGrammar}_s(G^k_j, \mathcal{C}, \mathcal{H}))\end{array}$$

5.1.2 *The Operator* $\mathit{newGrammar}_s(G, \mathcal{C}, \mathcal{H})$

Given a set of grammar rules G, a set of constraints $\mathcal{C}$ (i.e., a term of sort CtrSet), and a set of grammar rules $\mathcal{H}$, the function $\mathit{newGrammar}_s(G, \mathcal{C}, \mathcal{H})$ joins the set of grammar rules in G and the new rules in $\mathcal{H}$ with the constraints $\mathcal{C}$.

$$\begin{array}{l}\mathit{newGrammar}_{S1}(G,\mathcal{C},\mathcal{H}) = \alpha(G,\mathcal{C}) \cup \alpha(\mathcal{H},\mathcal{C}) \cup \widehat{G} \cup \widehat{\mathcal{H}} \\ \mathit{newGrammar}_{S2}(G,\mathcal{C},\mathcal{H}) = \beta(G,\mathcal{C}) \cup \widehat{G} \cup \widehat{\mathcal{H}}\end{array}$$

where:

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

where the substitution θ_α is obtained as follows: there are several (possibly renamed) $d_{j_1}, \ldots, d_{j_m} \in \mathcal{C}$ of the form $(W_{j'} \not\approx s_{j'})$ 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;

$$\begin{array}{r}\beta(G,\mathcal{C}) = \{c_1, \ldots, c_k, \theta_\beta(\mathcal{C}) \mapsto t \in L \mid (c_1, \ldots, c_k \mapsto t \in L) \in G \\ \wedge\ \nexists c_i, \nexists Y \in \mathcal{X} : c_i \equiv (Y \notin \mathcal{I})\}\end{array}$$

where the substitution θ_β is obtained as follows: there are several (possibly renamed) $d_{j_1}, \ldots, d_{j_m} \in \mathcal{C}$ of the form $(t_{j'} \not\approx s_{j'})$ with terms $t_1, \ldots, t_m, s_1, \ldots, s_m$. We then define $\theta_\beta(t_{j'}) = t$ for $j' \in \{1, \ldots, m\}$ and θ_β is the identity elsewhere.

Finally, for a set S of grammar rules, we define $\widehat{S}$ as follows

$$\widehat{S} = \{(c_1, \ldots, c_k \mapsto t \in L) \in S \mid \exists c_j, \exists Y \in \mathcal{X} : c_j \equiv (Y \in L)\}.$$

Example 6. Consider the following set $\mathcal{C}$ of new constraints (represented as a term of sort CtrSet)

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

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

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

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

$$\begin{array}{l}\mathit{newGrammar}_{S1}(G^0_{sd_3}, \mathcal{C}, \mathcal{H}) \\ \qquad = \alpha(G^0_{sd_3}, \mathcal{C}) \cup \alpha(\mathcal{H}, \mathcal{C}) \cup \widehat{G^0_{sd_3}} \cup \widehat{\mathcal{H}}\end{array}$$

where

$$\begin{array}{l}\alpha(G^0_{sd_3}, \mathcal{C}) \\ = \{Z \notin \mathcal{I},\ Z \not\approx (A; n(A,r)),\ Z \not\approx (Z'; n(B,r)) \mapsto pk(R,Z) \in L\}\end{array}$$

because we simply look for a rule that includes a constraint of the form $Y\notin\mathcal{I}$, namely $Z\notin\mathcal{I} \mapsto pk(R,Z)\in L$, 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

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

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 $(Y\in L)$. Finally we collect all the rules in $\mathcal{H}$ with a constraint of the form $(Y \in L)$

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

Thus, we finally obtain

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

■

5.1.3 *The Operator* $optimize(G)$

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

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

where $removeConstraints(G) = \{maximal_{\sqsubseteq}(\mathcal{C}) \mapsto (t\in L) \mid (\mathcal{C} \mapsto (t\in L)) \in G\}$ 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 $\sqsubseteq$ (defined in Section 4.3). And where $removeRules(G)$ is defined as pick a grammar rule $(\mathcal{C} \mapsto (t\in L)) \in G$, if we have $\langle G - \{\mathcal{C} \mapsto (t\in L)\}, \mathcal{C}\rangle \vdash (t\in L)$, then remove it from G, in any case repeat the process until no more grammar rules have been removed (fixpoint).

Example 7. Continuing Example 6, the optimization of the previous $newGrammar_{S1}(G^0_{sd_3},\mathcal{C},\mathcal{H})$ yields:

$$\begin{array}{l} optimize(newGrammar_{S1}(G_0,\mathcal{C},\mathcal{H}))\\ = \{Y \notin \mathcal{I},\ Y\not\preceq(Z';n(B,r)) \mapsto pk(A,n(A,r);Y)\in L\\ \quad Z \notin \mathcal{I},\ Z\not\preceq(Z';n(B,r)) \mapsto pk(R,Z)\in L,\\ \quad Z\in L \mapsto pk(c,Z)\in L,\\ \quad Z\in L \mapsto pk(A,n(A,r);sk(B,Z))\in L,\\ \quad Z\in L \mapsto sk(A,Z)\in L,\\ \quad Z\in L \mapsto Z;Y\in L,\\ \quad Z\in L \mapsto X;Z\in L\end{array}$$

since $(Y\not\preceq A;n(A,r)) \sqsubseteq (Y\not\preceq Z';n(B,r))$. If we had a rule $Y\in L \mapsto pk(c,n(A,r);sk(B,Y))\in L$, this would be removed, since it can be obtained from the other rules. ■

5.1.4 *The Relation* $\langle\mathcal{C},\mathcal{H},G\rangle \Rrightarrow_{\mathcal{P},\mathcal{G}_i,G^k_j,s} \langle\mathcal{C}',\mathcal{H}',G''\rangle$

Given a set of grammar rules G, a set of constraints $\mathcal{C}$, and a set of grammar rules $\mathcal{H}$, we define the transformation relation $\Rrightarrow_{\mathcal{P},\mathcal{G}_i,G^k_j,s}$ 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 $g \in G$. The auxiliary relation $\leadsto_{\sigma,\mathcal{P},\mathcal{G}_i,G^k_j}$, which produces a backwards narrowing step using $R^{-1}_{\mathcal{P}}$ under some extra conditions, is defined in Section 5.1.5 below. The auxiliary function $heuristics_{G^k_j,s}(g,\sigma,g')$, which applies a set of heuristics to produce a new constraint or new rule from a backwards narrowing step, is defined in Section 5.1.6 below. The relation $\Rrightarrow_{\mathcal{P},\mathcal{G}_i,G^k_j,s}$ is defined as follows:

$$\langle\mathcal{C},\mathcal{H},G\cup\{g\}\rangle \Rrightarrow_{\mathcal{P},\mathcal{G}_i,G^k_j,s} \langle\mathcal{C}\cup\mathcal{C}_g,\mathcal{H}\cup\mathcal{H}_g,G\rangle$$

where the set of new constraints is $\mathcal{C}_g = \cup\{\mathcal{C}_{g\leadsto_\sigma g'}\}$, the set of new grammar rules is $\mathcal{H}_g = \cup\{\mathcal{H}_{g\leadsto_\sigma g'}\}$, and for each backwards narrowing step $g \leadsto_{\sigma,\mathcal{P},\mathcal{G}_i,G^k_j} g'$, $\mathcal{C}_{g\leadsto_\sigma g'}$ is by definition the first component of the heuristic application $heuristics_{G^k_j,s}(g,\sigma,g') = \langle\mathcal{C}',\mathcal{H}'\rangle$, and $\mathcal{H}_{g\leadsto_\sigma g'}$ is by definition the second component.

Example 8. For the transformation step

$$\begin{array}{l}\mathcal{G}_5 = \langle G^!_{sd_1}, G^!_{sd_2}, G^0_{sd_3}, G^0_{sd_4}\rangle\\ \quad\Rrightarrow_{\mathcal{P},\mathcal{G}_5,S1}\ \mathcal{G}_6 = \langle G^!_{sd_1}, G^!_{sd_2}, G^1_{sd_3}, G^0_{sd_4}\rangle\end{array}$$

of Example 6, we obtain a set of backwards narrowing steps $sd_3 \leadsto_{\sigma,\mathcal{P},\mathcal{G}_5,G^0_{sd_3}} g'$ from the only rule sd_3 in $G^0_{sd_3}$ and apply $heuristics_{G^0_{sd_3},S1}(sd_3,\sigma,g')$ to each one of them. Then, the sets $\mathcal{H}_{sd_3} = \cup\{\mathcal{H}_{sd_3\leadsto_\sigma g'}\}$ and $\mathcal{C}_{sd_3} = \cup\{\mathcal{C}_{sd_3\leadsto_\sigma g'}\}$ are the sets $\mathcal{H}$ and $\mathcal{C}$ shown in Example 7. ■

5.1.5 *The Backwards Narrowing Relation* $g \leadsto_{\sigma,\mathcal{P},\mathcal{G}_i,G^k_j} g'$

Given a grammar rule g of a grammar G^k_j in $\mathcal{G}_i$, 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^{k+1}_j. We denote such backwards narrowing relation by the arrow $\leadsto_{\sigma,\mathcal{P},\mathcal{G}_i,G^k_j}$, where σ is the computed unifier, $\mathcal{R}_{\mathcal{P}}$ is the set of rules to be used for narrowing modulo $E_{\mathcal{P}}$, and $\mathcal{G}_i$ and G^k_j are used to further narrow the grammar rule and check some properties that the resulting pre-grammar rule must satisfy. Note that rules are always renamed to avoid variable name clashes. Note also that we assume that $E_{\mathcal{P}}$ has a finitary unification algorithm (see Section 2). Actually, in the NPA $E_{\mathcal{P}}$ consists of the usual equations for encryption and decryption and a complete and finite unification algorithm exists for such equational theory $E_{\mathcal{P}}$ (by orienting the equational theory into a rewrite theory; see Section 2). The relation $\leadsto_{\sigma,\mathcal{P},\mathcal{G}_i,G^k_j}$ is defined as follows:

$$\begin{array}{l} g \leadsto_{\sigma,\mathcal{P},\mathcal{G}_i,G^k_j} g''\\ \text{if } g \leadsto^{\bullet}_{\sigma,R^{-1}_{\mathcal{P}},E_{\mathcal{P}}} g',\ g' \leadsto^{!}_{\mathcal{R}^{-1}_{G^k_j}} g'', \text{ and } g'' \text{ is } \mathcal{G}_i\textit{-expandable}\end{array}$$

where the narrowing relation $\leadsto^{\bullet}_{\sigma,R^{-1}_{\mathcal{P}},E_{\mathcal{P}}}$ used in the NPA is slightly different from the ordinary narrowing relation $\leadsto_{\sigma,R^{-1}_{\mathcal{P}},E_{\mathcal{P}}}$ (see Appendix A for details). And where we say that a pre-grammar rule $g'' \equiv c_1,\ldots,c_k \mapsto (t_1,\ldots,t_n)\in L$ is $\mathcal{G}_i$*-expandable* iff

(i) there is no c_i and t_j such that $c_i \equiv (t_j \notin \mathcal{I})$,

(ii) for each c_i of the form $(u \not\preceq t)$, $\exists\theta : \theta(u) \equiv \theta(t)$, and

(iii) for each t_j, $\langle \mathcal{G}_i, (c_1, \ldots, c_k)\rangle \not\vdash (t_i \in L)$.

If a pre-grammar rule is $\mathcal{G}_i$-expandable, we apply the heuristics (Section 5.1.6) 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.

Recall from Sections 4.1 and 4.2 that symbol $\_\mapsto\_$ has its first argument frozen in $R_{\mathcal{P}}$, whereas it has instead its second argument frozen in R_G, i.e., given a pre-grammar rule $c_1, \ldots, c_k \mapsto (t_1, \ldots, t_n) \in L$, the relation $\leadsto^{\bullet}_{\sigma, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}}$ narrows only the right part, whereas the relation $\leadsto_{\mathcal{R}^{-1}_{G^k_j}}$ narrows only the left part.

EXAMPLE 9. Consider the following backwards narrowing steps. In the transformation step $\mathcal{G}_5 \Rrightarrow_{\mathcal{P},\mathcal{G}_5,S1} \mathcal{G}_6$ from $G^0_{sd_3}$ to $G^1_{sd_3}$ of the global scheme of Example 5, we have the following backwards narrowing step using the protocol rule $(p2) \equiv pk(A, n(A, r); Z') \to pk(B, Z')$:

$$\begin{array}{l} Z \notin \mathcal{I} \mapsto pk(R, Z) \in L \\ \quad \leadsto^{\bullet}_{id, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} \; Z \notin \mathcal{I} \mapsto pk(A, n(A, r); sk(B, pk(R, Z))) \in L \end{array}$$

where $Z \notin \mathcal{I} \mapsto pk(A, n(A, r); sk(B, pk(R, Z))) \in L$ is a $G^0_{sd_3}$-expandable grammar rule and we have solved the equational unification problem $pk(B, Z') =_{E_{\mathcal{P}}} pk(R, Z)$ using the unifier id and the equation $eq1 \equiv sk(Y, pk(Y, Z)) = Z$. For the grammar $G^0_{sd_3}$, we have the following backwards narrowing step using the protocol rule $(p10) \equiv M \to pk(Y, M)$:

$$Z \notin \mathcal{I} \mapsto pk(R, Z) \in L \;\; \leadsto^{\bullet}_{id, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} \;\; Z \notin \mathcal{I} \mapsto Z \in L$$

but this step is not $\mathcal{G}_5$-expandable, because of condition (i) in the previous definition. Again for the grammar $G^0_{sd_3}$, we have the following backwards narrowing step using $(p1) \equiv \emptyset \to pk(B, A; n(A, r))$:

$$\begin{array}{l} Z \notin \mathcal{I},\; Z \not\preceq (Z'; n(B, r)) \mapsto pk(R, Z) \in L \\ \quad \leadsto^{\bullet}_{[Z/(A; n(A,r))], R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} \\ (A; n(A, r) \notin \mathcal{I},\; (A; n(A, r) \not\preceq (Z'; n(B, r))) \mapsto \emptyset \end{array}$$

but this step is not $\mathcal{G}_5$-expandable, because of condition (ii) in the previous definition, i.e., $(A; n(A, r) \preceq (Z'; n(B, r))$ as shown in Example 4. And for the transformation step $\mathcal{G}_6 \Rrightarrow_{\mathcal{P},\mathcal{G}_6,S1} \mathcal{G}_7$ from $G^1_{sd_3}$ to $G^!_{sd_3}$, we have the following backwards narrowing step using $(p7) \equiv M_1; M_2 \to M_1$:

$$\begin{array}{l} Y \in L \mapsto pk(c, Y) \in L \\ \quad \leadsto^{\bullet}_{id, R_{\mathcal{P}}^{-1}, E_{\mathcal{P}}} \; Y \in L \mapsto pk(c, Y); M_2 \in L \end{array}$$

but this step is not $\mathcal{G}_6$-expandable because of condition (iii) in the previous definition, i.e.,

$$\langle\; G^0_{sd_3},\; Y \in L \;\rangle \vdash (pk(c, Y); M_2) \in L,$$

as shown in Example 4. ■

5.1.6 *The Operator* $heuristics_{G^k_j, s}(g, \sigma, g')$

Here, we use the result of a backwards narrowing step $\leadsto_{\sigma, \mathcal{P}, \mathcal{G}_i, G^k_j}$ to decide which grammar rule or constraint should be generated in order to refine the language 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). The following inference rules define the four heuristics, where s is a global strategy parameter that can be instantiated to $S1$ or $S2$ (see below), the variable Y is a fresh new variable, $g = \mathcal{C} \mapsto t \in L$, and $g' = \mathcal{D} \mapsto (s_1, \ldots, s_n) \in L$:

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

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

$$H2b \quad \frac{\sigma(t) \not\equiv t \quad \forall c_i \in \mathcal{C} : c_i \not\equiv (X \in L)}{heuristics_{G^k_j, s}(g, \sigma, g') = \langle \{t \not\preceq \sigma(t)\}, \emptyset\rangle}$$

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

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

- $S1$. Apply heuristics in the following order: try 1, if it fails try $2a$, if it fails try 3, otherwise $heuristics_{G^k_j, s}(g, \sigma, g') = \langle \emptyset, \emptyset\rangle$.
- $S2$. Like $S1$, but try $2b$ instead of $2a$.

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

EXAMPLE 10. Consider the transformation step $\mathcal{G}_5 \Rrightarrow_{\mathcal{P},\mathcal{G}_5,S1} \mathcal{G}_6$ from $G^0_{sd_3}$ to $G^1_{sd_3}$ of the global scheme of Example 5. For the backwards narrowing step using protocol rule (p7)

$$\begin{array}{l} g \equiv Z \notin \mathcal{I} \mapsto pk(R, Z) \in L \\ \quad \leadsto_{id, \mathcal{P}, \mathcal{G}_5, G^0_{sd_3}} \; g' \equiv Z \notin \mathcal{I} \mapsto pk(R, Z); M_2 \in L \end{array}$$

we can apply heuristic 1 yielding

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

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

$$\begin{array}{l} g \equiv Z \notin \mathcal{I} \mapsto pk(R, Z) \in L \\ \quad \leadsto_{[Z/A; n(A,r)], \mathcal{P}, \mathcal{G}_5, G^0_{sd_3}} \; g' \equiv A; n(A, r) \notin \mathcal{I} \mapsto \emptyset \end{array}$$

we can apply heuristic $2a$ yielding

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

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)

$$\begin{array}{l} g \equiv Z \notin \mathcal{I} \mapsto pk(R, Z) \in L \\ \quad \leadsto_{id, \mathcal{P}, \mathcal{G}_5, G^0_{sd_3}} \; g' \equiv Z \notin \mathcal{I} \mapsto pk(A, n(A, r); Z) \in L \end{array}$$

we can apply heuristic 3 yielding

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

since the subterm Z is already in an original constraint of the form $\_ \notin \mathcal{I}$ in g. ■

$$\begin{tikzcd}
 & \langle t, w\rangle \arrow[dl] \arrow[r, phantom, "\rightsquigarrow_{\mathcal{P},\mathcal{G}}"] & \langle t, w'\rangle \arrow[d] \\
t \rightsquigarrow^{\bullet}_{\sigma,R_{\mathcal{P}}^{-1},E_{\mathcal{P}}} t' \arrow[rr] & & \langle \mathcal{G}, \mathcal{D}\rangle \vdash \mathcal{C} \arrow[d] \\
 & & c \sqsubseteq c'
\end{tikzcd}$$

Figure 2: Dependencies between relations and operators in the Reachability phase

5.2 Reachability Analysis by Narrowing

The key requirement for any inference system that can make use of the grammar generation technique is that it offers some notion of an intruder who knows some terms in the present and past, and will learn more terms in the future. The inference system offered by the NRL Protocol Analyzer, which we are currently in the process of formalizing, has such properties. But we can also show how grammars are used using a *simpler* relation $\rightsquigarrow_{\mathcal{P},\mathcal{G}}$, based on the inference rules that we developed in this paper. In this section we use this simpler relation to illustrate how one might use grammars to cut down the search space.

The auxiliary relations of $\rightsquigarrow_{\mathcal{P},\mathcal{G}}$ are depicted in Figure 2. The relation $\rightsquigarrow_{\mathcal{P},\mathcal{G}}$ is defined as follows, where t_{bad} is a term of sort $MsgSet$ and w is a sequence $s_1 \,.\, s_2 \,.\, \cdots \,.\, s_{k-1} \,.\, s_k$ of previously found terms with ϵ the empty sequence:

$$\langle t, w\rangle \rightsquigarrow_{\mathcal{P},\mathcal{G}} \langle t', \sigma(t|_p \,.\, w)\rangle$$

if $t \overset{p}{\rightsquigarrow}{}^{\bullet}_{\sigma,R_{\mathcal{P}}^{-1},E_{\mathcal{P}}} t'$ and for each $s_i \in t'$,
$\langle \mathcal{G}, ctr(w)\rangle \not\vdash (s_i \notin \mathcal{I})$ and $\langle \mathcal{G}, ctr(w)\rangle \not\vdash (s_i \in L)$;
where $ctr(u_1 \,.\, \cdots \,.\, u_n) = (u_1 \notin \mathcal{I}, \ldots, u_n \notin \mathcal{I})$.

Then, given a set of terms t_{bad} that define the final state of the protocol we are looking for:

(i) if we can reach by narrowing an attack state of the form $\langle \emptyset, w\rangle$, i.e., $\langle t_{bad}, \epsilon\rangle \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle \emptyset, w\rangle$ such that $t_{\mathcal{I}} \sqsubseteq ctr(w)$, then this implies that the final state t_{bad} is reachable and w contains the sequence of protocol steps that constitute the attack;

(ii) if we do not find any attack state and obtain a finite narrowing tree using $\rightsquigarrow_{\mathcal{P},\mathcal{G}}$, i.e., there is no infinite sequence $\langle t_{bad}, \emptyset\rangle \rightsquigarrow_{\mathcal{P},\mathcal{G}} \cdots \langle t_i, w_i\rangle \rightsquigarrow_{\mathcal{P},\mathcal{G}} \cdots$, then the final state t_{bad} cannot be reached by the protocol, i.e., the protocol is secure; or

(iii) if the narrowing tree using $\rightsquigarrow_{\mathcal{P},\mathcal{G}}$ is infinite and we don't find any attack state, then the NPA cannot decide whether the protocol is secure or not.

Example 11. Continuing Example 5, the (backwards narrowing-based) reachability analysis detects an attack from the term $t_{bad} = final(B, A; n(B, r))$ in nine steps, i.e.,

$$\begin{array}{l}
\langle final(B, A; n(B, r)), \epsilon\rangle \\
\quad \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle (pk(B, A; Z), pk(B, n(B, r))), w_1\rangle \\
\quad \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle pk(B, n(B, r))), w_2\rangle \\
\quad \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle n(B, r), w_3\rangle \ \ \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle pk(c, n(B, r)), w_4\rangle \\
\quad \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle pk(A, n(A, r'); n(B, r)), w_5\rangle \\
\quad \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle pk(B, A; n(A, r')), w_6\rangle \ \ \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle A; n(A, r'), w_7\rangle \\
\quad \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle pk(c, A; n(A, r')), w_8\rangle \ \ \rightsquigarrow^{*}_{\mathcal{P},\mathcal{G}} \langle \emptyset, w_9\rangle
\end{array}$$

where the attack trace w_9 is just $w_9 = pk(c, A; n(A, r')) \,.\, A; n(A, r') \,.\, pk(B, A; n(A, r')) \,.\, pk(A, n(A, r'); n(B, r)) \,.\, pk(c, n(B, r)) \,.\, n(B, r) \,.\, pk(B, n(B, r))) \,.\, pk(B, A; Z) \,.\, final(B, A; n(B, r))$. This attack corresponds to the following message exchange sequence according to the informal description of the protocol in [18]:

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

2. $I_A \hookrightarrow B : \{K_A, A\}_{K_B}$
   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 previous nonce N_A or not. If it finds N_A, it assumes 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 see how the grammars can be used in this reachability analysis, consider the above state $\langle n(B, r), w_3\rangle$, where $w3 = pk(B, n(B, r)) \,.\, pk(B, A; Z) \,.\, final(B, A; n(B, r))$. Suppose that we apply the rule $(p10) \equiv M \to pk(U, M)$ to the state $\langle n(B, r), w_3\rangle$. Using backwards narrowing via this rule, we obtain the state

$$\langle\ sk(U, n(B, r)),\ n(B, r) \,.\, pk(B, n(B, r)) \,.\, pk(B, A, Z)\ \rangle.$$

We can then use grammar rule $(g4.5) \equiv Z \notin \mathcal{I} \mapsto sk(A, Z) \in L$ from Example 5 to conclude that this state is unreachable. Thus this state is not explored any further. ■

6. CONCLUDING REMARKS

We have given a precise rewriting-based formalization of the NPA language generation mechanisms. And we have illustrated its use by means of a well-known protocol. We have implemented the grammar generation inference system based on rewriting and narrowing in the Maude rewriting logic language [4], and this prototype has been used to check all the grammar-generation examples in the paper.

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 the grammar-based reachability analysis used in the NPA tool. We have taken the first steps towards such a formalization in this paper via the relation $\rightsquigarrow_{\mathcal{P},\mathcal{G}}$, but in order to increase its efficiency and expressiveness we will need to capture some notion

of local state of each principal involved. We will also integrate this reachability analysis into the present implementation.

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.

7. REFERENCES

[1] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, June 2005. Published online December 2004.

[2] Y. Chevalier, R. Kusters, M. Rusinowitch, and M. Turuani. 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*, volume 2914 of *Lecture Notes in Computer Science*, pages 124–135, 2003.

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

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

[5] H. Comon-Lundh and V. Shmatikov. 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)*, pages 271–280, 2003.

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

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

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

[9] C. Meadows. Applying formal methods to the analysis of a keymanagement protocol. *Journal of Computer Security*, 1(1), January 1992.

[10] C. Meadows. Language generation and verification in the NRL Protocol Analyzer. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 48–61. IEEE Computer Society Press, June 1996.

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

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

[13] C. Meadows, I. Cervesato, and P. Syverson. Specification of the group domain of interpretation protocol using npatrl and the NRL protocol analyzer. *Journal of Computer Security*, 12(6):893–932, 2004.

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

[15] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, pages 18–61. Springer LNCS 1376, 1998.

[16] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to the verification of cryptographic protocols. In N. Martí-Oliet, editor, *Proc. 5th. Intl. Workshop on Rewriting Logic and its Applications*. ENTCS, Elsevier, 2004.

[17] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In 8^{th} *ACM Conference on Computer and Communications Security (CCS '01)*, pages 166–175, 2001.

[18] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[19] S. Stubblebine and C. Meadows. Formal characterization and automated analysis of known-pair and chosen-text attacks. *IEEE Journal on Selected Areas in Communications*, 18(4):571–581, April 2000.

[20] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.

APPENDIX

A. THE NARROWING RELATION $\leadsto^{\bullet}_{\sigma,R,E}$

The narrowing relation $\leadsto^{\bullet}_{\sigma,R,E}$ is entirely similar to the narrowing modulo E relation $\leadsto_{\sigma,R,E}$, except that the unification algorithm modulo E and its corresponding set of unifiers are modified as follows.

> $t \overset{p}{\leadsto}{}^{\bullet}_{\sigma,R,E}\, t'$ if there is a nonfrozen position $p \in \mathcal{P}os_{\Sigma}(t)$, a (possibly renamed) rule $l \to r$ in R where we assume $\mathcal{V}ar(t) \cap Var(l) = \emptyset$, and $\sigma \in CSU^{\bullet}(t|_p =_E l, V)$ for a set of variables V containing $\mathcal{V}ar(t)$ and $\mathcal{V}ar(l)$, such that $t' = \sigma(t[r]_p)$

where the set of unifiers $CSU^{\bullet}(u =_E v, V)$ is defined by $\sigma \in CSU^{\bullet}(u =_E v, V)$ iff $u \overset{\bullet}{\simeq} v \leadsto^{!}_{\sigma, {}^{\bullet}E}$ *True*, where we assume that the equations E can be viewed as terminating and confluent rewrite rules $\vec{E}$, and where ${}^{\bullet}E$ is the set of rewrite rules ${}^{\bullet}E = \vec{E} \cup \{x \overset{\bullet}{\simeq} x \to True\}$ and we add the frozenness requirement $\phi(\overset{\bullet}{\simeq}) = \{1\}$.

In other words, in this modified procedure when unifying two terms u and v modulo the equations E by narrowing with the rules $\vec{E}$, we begin with the expression $u \overset{\bullet}{\simeq} v$ and try to reach the term *True*, but the frozenness of $\overset{\bullet}{\simeq}$ 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 $x \overset{\bullet}{\simeq} x \to True$, is a final unification of u with the narrowed right term performed.

The reason for using this modified narrowing relation $\leadsto^{\bullet}_{\sigma,R,E}$ in 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 in irreducible form w.r.t. the encryption/decryption equations in E, whereas the output terms $s_1, \ldots, s_k$ are not necessarily irreducible. This is because the terms $t_1, \ldots, t_m$ (understood as system messages) 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.