Ensuring Security through Model Checking in a Logical Environment
(Preliminary Results)

Marco Bozzano

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Via Dodecaneso 35
16146 Genova - Italy

bozzano@disi.unige.it

Abstract

A logical environment for specifying and proving authentication protocols correct is presented. Protocols are modeled as parametric, infinite-state systems, and specified using a multiset rewriting formalism, with universal quantification providing a way to choose new values like nonces. We show that secrecy and confidentiality properties can be verified using a technique based on symbolic model-checking and unification. Unlike previous approaches based on model-checking, we make use of a backward, bottom-up evaluation strategy which does not pose any restriction on the number of principals or the number of simultaneous sessions involved. We report on some preliminary experiments on the paradigmatic Needham-Schroeder public-key authentication protocol.

Keywords: multiset rewriting, model checking, authentication protocols.

1 Introduction

In a distributed environment, it is necessary to provide the agents with a way to ensure each other's identity. This is the role of authentication protocols. Authentication protocols should be reliable enough to be used in a potentially compromised environment, so to prevent a malicious intruder to cheat another agent about its own identity, or to get unauthorized information. While cryptographic primitives are a common means to achieve these goals, they are not sufficient to ensure authentication. Exchanging *nonces*, i.e. fresh values, is a commonly used technique which is exploited in combination with cryptography to achieve authentication, like in the well-known Needham-Schroeder public-key authentication protocol [NS78].

The design and implementation of cryptographic protocols are difficult and error-prone. The discovery of an *attack* to the above mentioned Needham-Schroeder protocol (seventeen years after its publication) by Lowe [Low95] is a meaningful example of this. That's why formal techniques to specify and analyze protocols have deserved great attention in the community. Different approaches have been followed for this purpose. An incomplete list include for instance using belief logics [BAN89], rewriting techniques [DMT98, CDL$^+$99, Cir01], theorem proving [Pau98], logic programming [Mea96, Del01, Bla01], and model-checking [MCJ97, RB99].

Formal verification of protocols, in our view is a process which is made up of two distinct, though related, phases:

i). choosing a suitable formalism to represent the environment and protocol execution (typically, in the form of a specification logic), and to specify security properties;

ii). enriching the logic with a (hopefully complete and automatic) tool to formally verify that a given specification is correct w.r.t. a given property.

Neither of the two phases should be underestimated. In particular, phase *i*) is responsible for fixing a suitable *abstraction* to represent protocol execution. Typically, simplifying assumptions on messages exchanged by agents and on cryptographic primitives are used in order to abstract from implementation-dependent details. For instance, according to the so-called *Dolev-Yao model* [DY83], messages are considered as indivisible abstract values, rather than sequences of bits, and encryption is modeled in an idealized way. Therefore, phase *ii*) alone is not sufficient to guarantee correctness. One should never forget that correctness results strongly rely on the abstraction chosen in phase *i*) (e.g. using the Dolev-Yao model we will not be able to prove that a protocol is resistant to implementation-dependent attacks due to the particular cryptoalgorithm used). Even most importantly, correctness results rely upon protocol specification given in phase *i*) being faithful (in a sense to be clarified) w.r.t. the given abstraction one has in mind.

In this paper we describe a comprehensive logical environment for the specification and analysis of protocols. Our position w.r.t. phase *i*) mentioned above is to use a specification logic which is as close as possible to the usual informal presentation it can be found in the literature on security protocols. In particular, following [CDL$^+$99] we will use a multiset rewriting formalism to represent a given set of agents (called *principals*) executing protocol sessions by exchanging messages over a network. We will use the Dolev-Yao intruder model and related message and cryptographic assumptions. Also, enriching multiset rewriting with quantification will provide a logical and clean way to express creation of new values like *nonces*. As far as phase *ii*) is concerned, we will describe a verification algorithm which can be (almost automatically) used to check some interesting security properties. In particular, our approach is well suited to verify properties which can be specified by means of *minimality conditions* (e.g., a given state is unsafe if there are *at least* two principals which have completed the execution of a protocol and a given shared secret has been unintentionally disclosed to a third malicious agent). Our verification algorithm has connections both with (symbolic) model checking and with theorem proving. Unlike traditional approaches based on model-checking, we can reason about *parametric*, *infinite-state* systems, thus we do not pose any limitation on the number of parallel runs of a given protocol (we also allow a principal to take part into different sessions at the same time, possibly with different roles). The verification algorithm relies upon a *backward* evaluation strategy which is based on a *bottom-up* semantics for a subset of linear logic, which we have studied in our previous work [BDM01b, Boz01].

In this paper, we present some preliminary experiments which we have carried out using the above approach. In particular, we will use the well-known public-key Needham-Schroeder protocol as a paradigmatic (though purely academic) example to describe our methodology. We have built a prototype, written in standard ML, to implement the verification algorithm. Although much work remains to be done, experiments show that our methodology can be effective to analyze interesting aspects of authentication such as secrecy or confidentiality.

Structure of the paper. After describing our specification logic and verification algorithm in Section 2, we will report on the experiments we have conducted on the Needham-Schroeder protocol in Section 3. In Section 4 we will discuss future work, and in Section 5 we will discuss related work and draw some conclusions.

2 A Logical Environment for Authentication

Security protocols are often presented by means of an informal notation. For instance, a simplified version of the public-key Needham-Schroeder authentication protocol (NS protocol hereafter), where the key distribution phase has been omitted [Low95], can be described by the following three lines:

$$\begin{array}{lllll} 1. & A \rightarrow B & : & \{N_a, a\}_{K_b} \\ 2. & B \rightarrow A & : & \{N_a, N_b\}_{K_a} \\ 3. & A \rightarrow B & : & \{N_b\}_{K_b} \end{array}$$

The notation $\{C\}_K$ indicates a message with content C and encrypted with a key K. Also, by convention $A, B, \ldots$ indicate *principal* identifiers, K_a denotes A's public key, and N_a is a *nonce*, i.e. a new value, generated by principal A. The protocol above has the following structure: a principal A initiates a new session by contacting another principal B and sending his/her own identity together with a nonce N_a, everything encrypted with B's public key. Upon receiving this message, principal B decrypts A's message, generates a new nonce N_b, and sends back to A both nonces, encrypted with A's key. Then, A checks the newly arrived message contains his/her previous nonce N_a, and, if so, sends N_b back to B, encrypted with B's key. Finally, B decrypts the last message and checks it contains the previously generated nonce N_b.

Completion of the protocol should convince A about B's identity (and vice versa) and also provide A and B with two shared values (N_a and N_b) which they could use afterwards for authentication purposes. Intuitively, assuming A and B behave honestly and their private keys are not known to anyone else, and assuming nonces are not guessable, the protocol should prevent a malicious intruder to impersonate one of the two agents. For instance, only B can decrypt message $\{N_a, a\}_{K_b}$, therefore only B can reply with an appropriate message $\{N_a, N_b\}_{K_a}$. However, under certain conditions, this protocol fails to achieve authentication [Low95].

2.1 A Logic for Protocol Specification

Following [CDL+99], we will present a logic for protocol specification based on multiset rewriting. Formally, this logic can be regarded as as an extension of multiset rewriting with universal quantification or equivalently as a fragment of Girard's linear logic [Gir87] [1], and its operational semantics can be given through a *goal-driven* sequent system, in the style of [Mil96]. In the rest of this section, we will use the NS protocol presented before to give an informal description of this logic.

Term Representation. First of all, we will need a term representation for the entities involved, such as *principals*, *keys*, and *messages*. The notation `pr(id,s)` will denote a principal with identifier `id` and internal state `s`. The internal state `s` can store information about an ongoing execution of the protocol (for instance, the identifier of the other principal, which step of the protocol has been executed, and so on). Messages sent over the network can be represented by terms like `n(pubk(id),m)` which stands for a message containing `m`, and encrypted with the public key of principal `id`. At any given instant of time, we can think of the current *state* as a *multiset* of terms representing both principals and messages currently on the network. For instance, the following multiset (where we have used "|", which is assumed to be commutative and associative, as multiset constructor): `pr(alice,init) | pr(bob,init)`, can describe a state in which there are two principals `alice` and `bob` in their initial state, and no messages currently over the network. The multiset

$$\mathtt{pr(alice,step1(na,bob))} \mid \mathtt{pr(bob,init)} \mid \mathtt{n(pubk(bob),mess(na,alice))}$$

can instead represent the current state after execution of step 1. of the protocol: `alice` sends a message to `bob` containing her identity and a new nonce `na`, and keeps trace of `na` and `bob`'s identity in her internal state. In general we allow a term to appear more than once in a given state, e.g. we can have two copies of the same message. We also allow multiple occurrences of terms representing principals, like `pr(alice,init) | pr(alice,init)`: in this way we allow a principal to take part into multiple protocol sessions, possibly with different roles.

Rules. Now, let us explain how protocol rules are specified. As said previously, we need to consider a restricted form of quantification to deal with nonce creation. Rules specify state transitions and have the syntactic format

$$\mathtt{F_1} \mid \ldots \mid \mathtt{F_n} \;:\!-\; \forall \mathtt{X_1} \ldots \forall \mathtt{X_k}\ (\mathtt{G_1} \mid \ldots \mid \mathtt{G_m}),$$

where $\mathtt{F_i}$, $\mathtt{G_i}$ are atomic formulas of the form $\mathtt{p(t_1,\ldots,t_l)}$ and $\mathtt{X_i}$ are variables. Intuitively, the standard semantics for universal quantification requires new values to be chosen before application of a rule. Rules should be read from left to right, and have the following intuitive

[1] To be precise, it is a subset of the language LO [AP91] enriched with universal quantification, see also [Boz01]. An equivalent *dual* fragment with *existential* quantification is instead considered in [CDL+99]

meaning: given a state containing the terms $\mathtt{F_1}, \dots, \mathtt{F_n}$, I can replace them with $\mathtt{G_1}, \dots, \mathtt{G_m}$, where new (distinct) constants $\mathtt{c_1}, \dots, \mathtt{c_k}$ have been substituted for the variables $\mathtt{X_1} \dots \mathtt{X_k}$. Free variables inside a rule are implicitly universally quantified throughout the rule, in other words they can be instantiated with any term before rule application. We immediately show an example. Step 2. of the NS protocol (see Figure 1) can be represented by the following rule:

$$\mathtt{pr(B, init)} \mid \mathtt{n(pubk(B), mess(NA, A))} \;\; :- \; \forall \mathtt{NB} \; (\mathtt{pr(B, step2(NA, NB, A))} \mid \mathtt{n(pubk(A), mess(NA, NB))}).$$

Principal `B` first decrypts the incoming message `n(pubk(B),mess(NA,A))`, stores the relevant information in his/her internal state, and replies to principal `A` with message `n(pubk(A),mess(NA,NB))`, where `NB` is a new nonce. We can apply rule above in the following way. Consider the state `pr(alice,step1(na,bob)) | pr(bob,init) | n(pubk(bob),mess(na,alice))`. We *instantiate* `A` with `alice`, `B` with `bob`, `NA` with `na`, and we substitute a new value `nb` for `NB`. Now, the head of the rule matches a submultiset of the current state, which can be replaced by the body, yielding the new state

$$\mathtt{pr(alice, step1(na, bob))} \mid \mathtt{pr(bob, step2(na, nb, alice))} \mid \mathtt{n(pubk(alice), mess(na, nb))}.$$

Rule theories. Following [CDL+99], we represent the environment in which protocol execution takes place by means of: a *protocol theory*, which includes rules for every protocol role (typically, one rule for every step of the protocol), and an *intruder theory*, which formalizes the set of possible actions of a malicious intruder which tries to break the protocol. In addition, it is possible to have additional rules for the environment, for instance we can allow the creation of new principals or new roles of a given principal during the execution of a protocol. We shall see an example in Section 3. We will use some simplifying assumptions, as mentioned in the introduction. In particular we will use an idealized view of cryptography, as in the examples given above, and we will use the so called "Dolev-Yao model" for the intruder. This model tries to depict a worst-case scenario, in which there is an intruder who has complete control of the network, so that he/she can intercept messages, block further transmission and/or replay them at any time, possibly modified. The intruder can also decompose messages, provided he/she knows the key they are encrypted with, and compose new messages with the information in his/her possession. The intruder can also use an internal memory to store information, create new nonces and in general he/she knows the identity and the public keys of the other principals. We will use terms such as `m(inf)` to represent the internal memory of the intruder, i.e. `m(inf)` holds if the intruder is in possession of the information `inf` (e.g. a nonce). By default, we assume the intruder *does not* know the *private* keys of the other principals, unless they have been disclosed to him in some way. Of course, without this latter assumption *any* protocol is breakable in general. We also assume that nonces are not guessable (we get this latter assumption for free, thanks to the treatment of nonces via universal quantification which we have chosen). Finally, we assume to have a set of principals (called `alice`, `bob`, ... in the following) who behave *honestly*, i.e. will adhere to the protocol rules and will not disclose secret information to a malicious intruder *on purpose*. In [SMC00], it has been proved that it is not restrictive to consider a single Dolev-Yao intruder instead of multiple ones. We postpone the presentation of the intruder theory, in the case of NS protocol, to Section 3.

Security properties. We conclude by discussing how it is possible to express the security properties we are interested in verifying. To be precise, as we will explain in Section

2.2, we express security properties by means of their negation, i.e. using formulas representing their *violation*. We will focus on properties which can be expressed by means of *minimality conditions*. Let us see an example. The following multiset represents a possible violation of the NS protocol (for every possible instantiation of the variables `NA` and `NB`): `pr(bob,step4(NA,NB,alice)) | m(NB) | m(NA)`. Intuitively, we are expressing that `bob` has completed the last step of the protocol with `alice`, but the intruder knows the nonces that `alice` and `bob` have exchanged (remember that we are assuming `alice` and `bob` to behave honestly). In other words authentication between `alice` and `bob` has caused leakage of information. Any multiset which strictly contains the one above still represents a security violation. For instance, the above multiset augmented with the atomic formula `pr(charles,step2(NC,ND,david))` still represents a security violation. A multiset such as the one above should be read "every state in which there is *at least* a principal `alice` who has completed the protocol with `bob` and the intruder is *at least* in possession of nonces `NA` and `NB`". In other words, multisets are implicitly considered *upward-closed*, i.e. if $\mathcal{M}$ represents a violation, $\mathcal{M} \cup \mathcal{M}'$ also does for every $\mathcal{M}'$.

2.2 Overview of the Verification Algorithm

The procedure we propose is inspired by a general method for verification of parameterized systems described in [ACJT96]. The methodology of [ACJT96] combines a *backward* evaluation strategy with a symbolic representation for *infinite* sets of states.

Backward Evaluation. In short, suppose S_{good} and S_{bad} represent the (disjoint) sets of good (safe) and bad (unsafe) states of a given (parametric) system S, and assume that I is a set representing the possible (legal) initial states of S. Proving the system correct means proving that all states which can be reached starting from a state in I belong to S_{good} (i.e. there is no trace leading from an initial state to a state in S_{bad}). Let *Post* and *Pre* be usual next state and previous state operators associated to the (transition) system S, and $Post^*$, Pre^* their reflexive and transitive closures. Now, the *forward* approach to verification consists in computing the set $Post^*(I)$and in checking that $Post^*(I) \subseteq S_{good}$. On the contrary, using the *backward* approach one tries to prove that $Pre^*(S_{bad}) \cap I = \emptyset$. In other words, one tries to verify that the set of states which can *eventually* lead to an unsafe state does not contain any legal initial state.

Despite these two approaches are apparently similar, it turns out that for many applications backward analysis is effective while forward analysis is not. The most interesting feature of backward analysis is that *violations* of safety (security) properties (i.e. the set S_{bad}) can often be expressed as *upward-closed* sets of states, as discussed in Section 2.1, while in general this is not the case for the set I of initial states. This is a desirable property, because upward-closed sets of states can be used to give a *finite* (symbolic) representation for *infinite* sets of states. As a result, the set $Pre^*(S_{bad})$ is often effectively computable, while the set $Post^*(I)$ is not. Note that computation of $Pre^*(S_{bad})$ can be done independently of the set I of initial states.

Verification Algorithm. The verification algorithm follows the directions outlined in the previous paragraph. Namely, a given protocol specification, as informally described in Section 2.1, can be seen as a set of rules defining a transition system, and the algorithm works by computing the set $Pre^*(S_{bad})$, where S_{bad} is the set of states representing security properties violations. Formally, the evaluation strategy can be seen as a kind of *bottom-up* evaluation for a fragment of linear logic, as described in our previous work [BDM01b, Boz01]. In this view, rules correspond to program clauses and security violations to axioms. As usual in logic programming [Llo87], bottom-up evaluation is a *goal-independent*, *fixpoint* computation which starts from the axioms and, step by step, accumulates goals which are provable from the goals already proved. In short, the following sequence is computed: $S_0 = S_{bad}$, $S_1 = S_0 \cup Pre(S_0)$, $S_2 = S_1 \cup Pre(S_1)$, $\ldots$. The algorithm terminates if a fixpoint is reached. The computation can be carried out symbolically thanks to the fact that the set S_{bad} is upward-closed and thanks to the following property: the set of predecessors of an upward-closed set of states is upward-closed. The algorithm makes use of an extended notion of multiset unification to compute the set of predecessors of a given set of states w.r.t. a given set of rules. Instead of trying to condense in a few lines a description of the bottom-up evaluation algorithm, we will present just a simple example. We refer the reader to [Boz01] for details. A summary of the soundness and completeness results for the fixpoint, bottom-up semantics for the logic under consideration is also presented in Appendix B. Suppose we have the following multiset representing a violation of security for the NS protocol: $\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{NA}', \mathtt{NB}', \mathtt{bob})) \mid \mathtt{m}(\mathtt{NB}')$, and consider the rule p_3) representing step 3. of the protocol (see Figure 1). We can compute the set of predecessors of the multiset w.r.t. the above rule in the following way. First of all, we need to unify the multiset with the rule body. This latter is composed of two atoms, `pr(A,step3(NA,NB,B))` and `n(pubk(B),mess(NB))`, only the first of which is contained (modulo unification) in the first multiset. Now, remember that violations are expressed as upward-closed sets. Therefore, the first multiset also symbolically represents all the multisets of the form

$$\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{NA}', \mathtt{NB}', \mathtt{bob})) \mid \mathtt{m}(\mathtt{NB}') \mid \mathtt{n}(\mathtt{pubk}(\mathtt{B}''), \mathtt{mess}(\mathtt{NB}'')).$$

Now, unification is performed between the rule body and a *submultiset* of the above multiset, namely $\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{NA}', \mathtt{NB}', \mathtt{bob})) \mid \mathtt{n}(\mathtt{pubk}(\mathtt{B}''), \mathtt{mess}(\mathtt{NB}''))$. This is done by substituting `alice` for `A`, $\mathtt{NA}'$ for `NA`, $\mathtt{NB}'$ for `NB` and $\mathtt{NB}''$, `bob` for `B` and $\mathtt{B}''$. By applying this substitution to the union of the clause head with what has been left out in the unification process, i.e. $\mathtt{m}(\mathtt{NB}')$, we get $\mathtt{pr}(\mathtt{alice}, \mathtt{step1}(\mathtt{NA}', \mathtt{bob})) \mid \mathtt{n}(\mathtt{pubk}(\mathtt{alice}), \mathtt{mess}(\mathtt{NA}', \mathtt{NB}')) \mid \mathtt{m}(\mathtt{NB}')$. In general, there may be more than one way to unify two given multisets, therefore we get a *set* of predecessors. This algorithm is repeated for every rule and every multiset in S_i, and results are joined together to compute the set S_{i+1}.

If a rule body contains universally quantified variables, new constants are substituted for them before applying the algorithm informally described above, and in the end a static check is made in order to ensure that the predecessor does not contain the constants which have been introduced. More details can be found in [Boz01].

Subsumption and Invariant Strengthening. In order to avoid state explosion, at every step a *subsumption* check is made in order to eliminate redundant elements in the sets S_i. A multiset $\mathcal{M}$ is subsumed by $\mathcal{M}'$ if there exist a substitution θ and a multiset $\mathcal{M}''$ s.t. $\mathcal{M} = \mathcal{M}'\theta + \mathcal{M}''$. In particular, this holds if $\mathcal{M}$ is an instance of $\mathcal{M}'$ (as usual in logic programming) and also if $\mathcal{M}'$ is a submultiset of $\mathcal{M}$ (multisets are always considered *upward-closed*).

In order to accelerate convergence of the fixpoint computation, it may be useful to apply a technique which we might call *invariant strengthening*. We will see some examples in Section 3. This technique consists in adding new axioms (or rules) to the theory under analysis. Specifically, the idea which we will discuss in Section 3 is that of enriching a theory by adding axioms representing further security violations w.r.t. the ones under analysis. Two comments are in order. First of all, this technique is perfectly *sound*. In other words, if no attacks exists in the augmented theory, no attack exists in the original one. In fact, adding axioms (or rules) has the effect of increasing the set $Pre^*(S_{bad})$, therefore if $Pre^*(S_{bad}) \cap I = \emptyset$ holds in the augmented theory, it also holds in the original theory. Second, the reason why invariant strengthening may be useful is intimately linked to subsumption. The idea is that a careful addition of new axioms can allow one to eliminate a lot of redundant elements at an earlier stage of computation. In other words, we can use *fewer* elements to *symbolically* represent a *larger* set of states. Invariant strengthening is not *complete* in general, in the sense that it may introduce *false attacks*. However, it should be noted that whenever adding an axiom introduces a false attack, this actually means that the axiom is *not* an invariant, i.e. it represents a security violation for which a trace exists.

Applications. A simple prototype which implements the verification procedure has been implemented in SML. At present we have used it to verify safety properties for *parametric* systems, like, e.g., mutual exclusion for access control protocols. In Section 3 we present our experiments on the NS public-key authentication protocol. We have performed the experiments on a Pentium II 233MhZ under Linux 2.0.32, running Standard ML of New Jersey, Version 110.0.7.

3 The Needham-Schroeder Authentication Protocol

In this section we discuss some experiments on the NS protocol. We will focus on the simplified version given in [Low95], where the key distribution phase has been omitted. Given that keys are not exchanged during protocol execution, we can make a *secrecy* assumption, i.e. we can assume that private keys are not known except to the relevant principals and will not be disclosed to the intruder. Accordingly, for illustration purposes it is convenient to make a simplification in the intruder theory. Namely, we assume that the intruder is able to decrypt only those messages which are encrypted with his/her own public key. More complex protocols and/or intruder theories can be managed as done in [CDL$^+$99].

3.1 Incorrect Needham-Schroeder

The overall theory for the first version of NS protocol corresponding to [NS78], together with a specification of the initial states and the security violations we want to study, is shown in Figure 1. Let us discuss it in more detail.

Protocol rules. We have one rule for every step of the protocol, i.e. rules p_1) through p_4) (we need one step more w.r.t. the informal notation in order to conclude the protocol). The coding is exactly as discussed in Section 2.1, i.e. we have synchronization between principals and messages on the network. In the body of rule p_1), the first term has been brought outside the scope of the quantifier for convenience. The internal states of principals have been enriched with the information on the current session they may need. Universal quantification is used to create nonces. Step 3 and step 4 conclude the protocol from the point of view of, respectively, the principal who began the session and the other one.

Protocol rules

$p_1)$ `pr(A, init) | pr(B, init) :- pr(B, init) | ∀NA (pr(A, step1(NA, B)) | n(pubk(B), mess(NA, A))).`
$p_2)$ `pr(B, init) | n(pubk(B), mess(NA, A)) :- ∀NB (pr(B, step2(NA, NB, A)) | n(pubk(A), mess(NA, NB))).`
$p_3)$ `pr(A, step1(NA, B)) | n(pubk(A), mess(NA, NB)) :- pr(A, step3(NA, NB, B)) | n(pubk(B), mess(NB)).`
$p_4)$ `pr(B, step2(NA, NB, A)) | n(pubk(B), mess(NB)) :- pr(B, step4(NA, NB, A)).`

Role creation

$r_1)$ `⊥ :- ∀ID (pr(ID, init)).`
$r_2)$ `pr(Z, S) :- pr(Z, S) | pr(Z, init).`

Unsafe states

`pr(alice, step3(NA, NB, bob)) | m(NA).`
`pr(alice, step3(NA, NB, bob)) | m(NB).`
`pr(bob, step4(NA, NB, alice)) | m(NA).`
`pr(bob, step4(NA, NB, alice)) | m(NB).`

Initial states

`Exclude:`
`pr(A, step1(NA, B)).  pr(A, step2(NA, NB, B)).`
`pr(A, step3(NA, NB, B)).  pr(A, step4(NA, NB, B)).`

Intruder theory

$i_1)$ `n(pubk(A), M) :- intercept(n(pubk(A), M)).`
$i_2)$ `intercept(n(pubk(A), M)) :-`
`intercept(n(pubk(A), M)) |`
`n(pubk(A), M).`
$i_3)$ `n(pubk(intruder), M) :- dec(M).`
$i_4)$ `dec(mess(M)) :- m(M).`
$i_5)$ `dec(mess(M, N)) :- m(M) | m(N).`
$i_6)$ `m(M) :- m(M) | comp(mess(M)).`
$i_7)$ `m(M) | m(N) :- m(M) | m(N) | comp(mess(M, N)).`
$i_8)$ `comp(C) :- n(pubk(Z), C).`
$i_9)$ `pr(Z, S) :- pr(Z, S) | m(Z).`
$i_{10})$ `⊥ :- ∀N (m(N)).`

Figure 1: First version of Needham-Schroeder protocol

Role creation. Rule r_1), where $\perp$ matches the empty multiset (therefore the rule is applicable in any state) allows creation of new principals at execution time (using universal quantification over their identifiers ensures creation of principals with new names). Rule r_2) allows a principal in the current state to make a copy of itself (note that the internal state is reset); in this way a principal can take part into different sessions at the same time, possibly in different roles.

Intruder theory. The intruder can: arbitrarily intercept any message (rule i_1)) and block further transmission; replay any intercepted message at any later time, possibly more than once (rule i_2)); decompose any message encrypted with his/her own public key (rule i_3)). Rules i_4) and i_5) formalize the decomposition phase; if the message contains just one component it is directly stored in the intruder memory (rule i_4)), otherwise the two components are stored. Decomposition rules have been optimized for the form of messages used in the protocol theory, we will discuss further this point in Section 4. Rules i_6) and i_7) model the composition phase. The intruder can choose at random pieces of information stored it his/her memory and use them to compose new messages. Note that the | operator is commutative and associative, therefore no particular order is imposed between the different pieces of information. Messages can then be sent (rule i_8)) to any principal by encrypting them with the relevant public key. Composition rules have been optimized similarly to decomposition ones. Note that our language is untyped, therefore we allow the intruder to use, e.g., principal identifiers as nonces. This permits to consider a limited form of *type flaw* attacks. Finally, the intruder can: store principal identifiers (rule i_9)), and create and store new values (rule i_{10})).

Initial States. Currently, our framework allows one to give a *partial* specification of the set of the initial states of a given system. It is possible to require a certain atom (multiset of atoms, in general) to belong to every initial state and/or impose an atom (multiset of atoms) not to belong to any initial state. For instance, in Figure 1 we simply require all principals to be initially in their initial state (we exclude principals in step 1, 2, 3, or 4), i.e. no session has been established yet. The verification algorithm signals it has *possibly* found an attack every time it finds a state which is compatible with the partial specification and the user must take care of checking if it is really an attack or not. This kind of partial specification is more flexible than specifying exactly the set of initial states, because it may help us to find additional hypothesis under which an attack might take place. For instance, the algorithm could inform us that it has found a state which complies with the partial specification but contains a term like `m(inf)`. This can be considered an attack if it is reasonable, under some circumstances, that the intruder initially possesses information `inf`.

Automatic verification. As a first experiment, we run our verification algorithm with the security properties (to be precise their violations) specified in Figure 1 as "unsafe states". We consider a state to be unsafe whenever there exist *at least* two principals `alice` and `bob` such that either `alice` believes to have completed a session with `bob` or `bob` believes to have completed a session with `alice`, and, in addition, the intruder has got *at least* one of two nonces (either the one created by `alice` or the one created by `bob`). As anticipated, the protocol fails to satisfy this (apparently rather obvious) security property, as first observed by Lowe [Low95]. Specifically, the algorithm terminates, in about 1 minute, finding an attack. Appendix A shows the trace of this attack, together with the corresponding transition sequence in our specification.

The attack takes place because `alice` decides to contact the intruder, *without knowing he/she is cheating*. Thus, the intruder can easily impersonate `alice` and cheat `bob`. Note that as a result the protocol has been broken from the point of view of `bob` (`bob` thinks he has got authentication with *alice* and that provided `alice` is honest, the nonces have not been disclosed to anyone else, which is false), whereas from the point of view of *alice*, she correctly thinks to have established authentication with the intruder principal (the nonces have been disclosed to `bob`, but only because the intruder is cheating and `alice` does not know that). With this in mind, we can now try the following stronger security violation:

$$\mathtt{pr(alice, step3(NA, NB, bob))} \mid \mathtt{pr(bob, step4(NA, NB, alice))} \mid \mathtt{m(NA)}.$$
$$\mathtt{pr(alice, step3(NA, NB, bob))} \mid \mathtt{pr(bob, step4(NA, NB, alice))} \mid \mathtt{m(NB)}.$$

Namely, we try to ascertain whether it is possible that two honest principals `alice` and `bob` both believe to have completed the protocol with each other, and still *at least* one of the two nonces has been disclosed to the intruder. As the reader can easily verify, this is not the case for the trace of the previous attack. Actually, this time the verification algorithm terminates, after about 10 minutes, yielding a negative result. NS protocol is safe with respect to this property. We conclude this section by showing how the methodology of *invariant strengthening*, discussed in Section 2.2, can dramatically improve the verification algorithm performance. Namely, we augment the theory with this further rule (violation): `pr(alice,step1(NA,bob)) | m(NA).` Intuitively, this violation is never met. In fact, the nonce `NA` is created by `alice` during step 1 and sent to `bob`. If `bob` is honest, he will never send it to the intruder. Adding this rule has the effect of accelerating convergence of the fixpoint computation. This time, the verification algorithm terminates in just 1 minute, which is a dramatic improvement. Note that the following invariant is instead violated (the attack follows the same scheme of the one given previously, see Appendix A): `pr(bob,step2(NA,NB,alice)) | m(NB).` We remind that adding rules (including axioms) to the theory is always sound, in the sense that if no attack is found in the augmented theory, no attack is found in the original one.

3.2 Corrected Needham-Schroeder

As observed by Lowe [Low95], the NS protocol can be fixed with a small modification. The problem with the original protocol is that the second message exchanged does not contain the identity of the principal who is responding. Therefore, the intruder can cheat the protocol initiator by simply forwarding the message created for him/her by the responder. Adding the responder's identity to the message prevents the intruder from replaying it, because now the initiator is expecting a message from the intruder. Also, the message is encrypted with the initiator's public key, and therefore it cannot be modified by the intruder. The corrected version of NS protocol is

$$\begin{array}{lrcl} 1. & A \rightarrow B & : & \{N_a, a\}_{K_b} \\ 2. & B \rightarrow A & : & \{B, N_a, N_b\}_{K_a} \\ 3. & A \rightarrow B & : & \{N_b\}_{K_b} \end{array}$$

We need to make minor modifications to the theory presented in Figure 1. Namely, we modify rules p_2) and p_3) by adding `B` to the message, and we add two rules for composition and decomposition of messages with three components:

p_2) $\mathtt{pr(B,init)} \mid \mathtt{n(pubk(B),mess(NA,A))} :\!- \forall \mathtt{NB}\ (\mathtt{pr(B,step2(NA,NB,A))} \mid \mathtt{n(pubk(A),mess(B,NA,NB))})$.
p_3) $\mathtt{pr(A,step1(NA,B))} \mid \mathtt{n(pubk(A),mess(B,NA,NB))} :\!- \mathtt{pr(A,step3(NA,NB,B))} \mid \mathtt{n(pubk(B),mess(NB))}$.

i_{11}) $\mathtt{dec(mess(M,N,O))} :\!- \mathtt{m(M)} \mid \mathtt{m(N)} \mid \mathtt{m(O)}$.
i_{12}) $\mathtt{m(M)} \mid \mathtt{m(N)} \mid \mathtt{m(O)} :\!- \mathtt{m(M)} \mid \mathtt{m(N)} \mid \mathtt{m(O)} \mid \mathtt{comp(mess(M,N,O))}$.

We can now use our algorithm to automatically verify the protocol correct w.r.t. the weaker security violations given in Figure 1. As in Section 3.1, we can accelerate convergence by means of invariant strengthening. We can use the two invariants discussed in Section 3.1 (the modified protocol should now satisfy also the second invariant, if our intuition is correct), adding

$$\mathtt{pr(alice,step1(NA,bob))} \mid \mathtt{m(NA)}. \quad \mathtt{pr(bob,step2(NA,NB,alice))} \mid \mathtt{m(NB)}.$$

The verification algorithm terminates, as expected, proving that the modified version of NS protocol is resistant w.r.t. the security violations in Figure 1. The fixpoint computation converges in a few seconds, whereas it takes about 20 minutes without invariant strengthening.

4 Future Work

In Section 3 we have given an encoding of NS public-key authentication protocol, which we have used to show a methodology for proving secrecy and confidentiality properties. Much work is still to be done, both from the point of view of the implementation and on the theoretical side.

The current implementation is a simple prototype written in Standard ML. It takes as input the theory to analyze, written in a syntax very close to the one presented in this paper, and run the verification algorithm until a fixpoint is reached (non-termination is possible). The user can give a partial specification of the set of initial states, as discussed in Section 3.1 (an exact specification is also possible), and the security violations to analyze, possibly enriched using invariant strengthening. Every possible attack found is reported to the user. Unification is implemented directly in SML. In particular, the critical operation is multiset unification, which requires to consider (classical) unification between sub-multisets of terms of the original multisets. The most expensive phase of the verification algorithm is by far the *subsumption* check, which makes heavy use of multiset unification. Subsumption is indeed the basis of the verification algorithm's power. Future work includes implementing some kind of *indexing* to order terms inside multisets, so to make unification faster, and optimizing the subsumption phase. We also need to optimize the term representation, which is currently very naive. *Cutting* the search space is also currently supported in the framework. The user can instruct the program to cut a part of the search space which is known to be unreachable. This strategy in general is *not* sound as invariant strengthening, therefore it is responsibility of the user to ensure that cut is performed correctly. We also plan to make further experiments on the several protocols studied in literature [CJ97]. Further work has to be done on the theoretical side. We discuss this point below.

Protocol Translation. We want to study a (possibly automatic) translation between the usual informal description of protocols and our representation. As shown in the case of NS protocol, a one-to-one translation (one rule for every step) could be enough, provided we have a way to store the information about the internal state of principals;

Theory Optimization. For efficiency reason, it could be worth making some optimizations, in particular to the intruder theory. Specifically, the general intruder theory could be specialized for every given protocol. For instance, in Section 3.2 we optimized (by hand) the rules for composition and decomposition. We plan to use techniques like *folding/unfolding* to automatize the process. A similar technique is used for instance in [Bla01];

Invariant Strengthening and Cut. We plan to enrich the framework with support for (automatic or semi-automatic) generation of invariants and cut conditions. Criteria for searching invariants on Petri Nets [Rei85] could be useful for this purpose. As shown by the experiments, invariant strengthening is crucial to improve efficiency. We also plan to consider defining widening operators in the style of [CC77].

We plan to enrich our framework with support for constraints, along the lines of [BDM01a]. We have carried out some preliminary experiments which show that our framework in combination with a constraint solver can be useful to prove safety of mutual-exclusion protocols using integer-valued variables. In the field of authentication protocols, a constraint solver can be exploited to prove correctness of protocols which make use of *timestamps*.

Another topic we would like to investigate is *typed multiset rewriting* [Cer01], which extends multiset rewriting with a typing theory based on dependent types with subsorting. Dependent types can be used to enforce dependency between an encryption key and its owner. The paper [Cer01] also presents some extensions which increase the flexibility of multiset rewriting specifications, e.g. using memory predicates to remember information across role executions.

Finally, an open question is the problem of non-termination. In the few examples we have presented, our algorithm is always terminating, even without invariant strengthening. Although secrecy has been proved to be undecidable, even for finite-length protocols with data of bounded complexity [CDL$^+$99], one may ask if a more restricted subclass of protocols exists, for which the verification algorithm presented here is always terminating.

5 Related Work and Conclusions

In this paper we have presented a framework for specification and analysis of authentication protocols. Protocols and security properties are specified using a uniform logic based on multiset rewriting, enhanced with nonce management via universal quantification. Analysis is based on a bottom-up evaluation algorithm which performs a *backward* search starting from a protocol/intruder theory and a set of axioms representing security *violations*. The verification procedure is tailored to study security violations which can be specified by means of *minimality conditions*. While this may rule out interesting properties, e.g. questions of *belief* [BAN89], the proposed approach can be used to study secrecy and confidentiality properties. No artificial limit is imposed on the number of simultaneous sessions we are able to analyze. Preliminary experiments show that the methodology we propose can be effective, and that using optimizations such as *invariant strengthening*, computation time is quite reasonable. As we expect computation time to increase for more complicated protocol and intruder theories, we think an optimized implementation should be developed. Also, some work has to be done to automatize some phases of the verification process. We discuss below some related work in more detail.

Most of the approaches which make use of rewriting techniques to study security protocols, e.g. [DMT98, Cir01] are limited to protocol debugging, i.e. they can find attacks mounted on a given protocol, but they cannot be used to analyze correctness. In [DMT98], an approach based on narrowing and symbolic execution is only presented as a future development. Also, the authors propose a *forward* breadth-first-search strategy, while effectiveness of our verification algorithm strongly relies on a *backward* search strategy. Another approach which shares some similarity with ours is [Del01], where a specification for security protocols based on rewriting and encoded in a subset of intuitionistic logic is presented. The author uses universal quantification to generate nonces and embedded implication to store the agents' knowledge. This approach is still limited to protocol debugging.

Traditional approaches based on model-checking, e.g. [MCJ97, RB99] make use of some kind of abstraction to transform the original problem into a *finite-state* model-checking problem. While this has the advantage of guaranteeing termination, on the other hand it only allows one to analyze a fixed number of concurrent protocol runs, an approach which is infeasible as this number increases. On the contrary, we use a symbolic representation which avoids putting limitations on the number of parallel sessions. While our approach is restricted to secrecy and confidentiality properties, some security properties other than secrecy are considered in [MCJ97]. [RB99] considers only one session at a time, does not allow type confusion, as we do, and does not support timestamps.

Our verification algorithm has something in common with theorem proving techniques. For instance, the NRL protocol analyzer [Mea96] specifies security protocols using Prolog rules, and, similarly to us, uses a bottom-up evaluation procedure which takes as input a set of insecure states. While state-space search can be automatically performed, the analyzer usually needs to be fed with some inductive lemmas by the user. Another approach based on theorem proving is

[Pau98]. Here, properties are proved by induction on traces, and analysis is a semi-automatic process which can take several days. Finally, in [Bla01], the author presents an optimized specification of security protocols based on an "attacker view" of protocol security, specified by means of Prolog rules, as in [Mea96]. The approach is effective, and has been applied to prove correctness of a number of real protocols. The verification algorithm performs a "backward depth-first" search, which seems to be closely related to our evaluation strategy, and uses an intermediate code optimization using a technique similar to *unfolding*, which we plan to study as future work. On the other hand, we think that the multiset rewriting formalism which we use is more amenable to an automatic translation from the usual informal notation used in the literature on security. As we stressed in the introduction, ensuring faithfulness between the informal description of a protocol and its specification is necessary to prove correctness. Also, with respect to [Bla01], we use a cleaner treatment for nonces, and we don't have to use approximations (which may introduce false attacks) except for *invariant strengthening*, which can be controlled by the user.

Acknowledgments

Part of this work has been carried out while the author was visiting Naval Research Laboratory and Carnegie Mellon University, and funded by ONR under grant N00014-01-1-0432. I would like to thank Cathy Meadows, Iliano Cervesato and Frank Pfenning for providing an exciting environment and for the several fruitful discussions I had with them. Finally, some parts of the SML code for the verification algorithm are adapted from the source code for a Lolli [HM94] interpreter, originally written by Joshua Hodas.

References

[ACJT96] P. A. Abdulla, K. Cerāns, B. Jonsson, and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proceedings of the 11th Annual IEEE Int. Symposium on Logic in Computer Science (LICS'96)*, pages 313–321. IEEE Computer Society Press, 1996.

[AP91] J.-M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with Built-In Inheritance. *New Generation Computing*, 9(3-4):445–473, 1991.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. Technical Report 39, Digital Equipment Corporation Systems Research Center, 1989.

[BDM01a] M. Bozzano, G. Delzanno, and M. Martelli. An Effective Bottom-Up Semantics for First Order Linear Logic Programs. In *Proceedings of the 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, pages 138–152, 2001.

[BDM01b] M. Bozzano, G. Delzanno, and M. Martelli. An Effective Fixpoint Semantics for Linear Logic Programs, 2001. To appear in *Theory and Practice of Logic Programming*.

[Bla01] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the 14th Computer Security Foundation Workshop (CSFW'01)*, 2001.

[Boz01] M. Bozzano. Fixpoint Semantics for a Fragment of First-Order Linear Logic. Technical Report CMU-CS-01-129, School of Computer Science, Carnegie Mellon University, 2001.

[CC77] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fix-Points. In *Proceedings of* *the 4th Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.

[CDL+99] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A Meta-notation for Protocol Analysis. In R. Gorrieri, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, pages 55–69, 1999.

[Cer01] I. Cervesato. Typed Multiset Rewriting Specifications of Security Protocols. In A. Seda, editor, *Proceedings of the First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT'00)*. Elsevier ENTCS, 2001. To appear.

[Cir01] H. Cirstea. Specifying Authentication Protocols Using Rewriting and Strategies. In I. Ramakrishnan, editor, *Proceedings of Conference on Practical Aspects of Declarative Languages (PADL'01)*, volume 1990 of *LNCS*, pages 138–152. Springer-Verlag, 2001.

[CJ97] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature, 1997. Web Draft Version 1.0 available from `http://www-users.cs.york.ac.uk/~jac/`.

[Del01] G. Delzanno. Specifying and Debugging Security Protocols via Hereditary Harrop Formulas and λProlog - A Case-study -. In *Proceedings of the 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, pages 123–137, 2001.

[DMT98] G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In N. Heintze and J. Wing, editors, *Proceedings of Workshop on Formal Methods and Security Protocols*, 1998.

[DY83] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1:1–102, 1987.

[HM94] J. Hodas and D. Miller. Logic Programming in a Fragment of Intuitionistic Linear Logic. *Information and Computation*, 110(2):327–365, 1994.

[Llo87] J. .W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[Low95] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56:131–133, 1995.

[MCJ97] W. Marrero, E. Clarke, and S. Jha. Model Checking for Security Protocols. Technical Report CMU-CS-97-139, School of Computer Science, Carnegie Mellon University, 1997.

[Mea96] C. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[Mil96] D. Miller. Forum: A Multiple-Conclusion Specification Logic. *Theoretical Computer Science*, 165(1):201–232, 1996.

[NS78] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[Pau98] L.C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[RB99] A.W. Roscoe and P.J. Broadfoot. Proving Security Protocols with Model Checkers by Data Independence Techniques. *Journal of Computer Security*, 7(2-3):147–190, 1999.

[Rei85] W. Reisig. *Petri Nets, an introduction*. EATCS, Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[SMC00] P. Syverson, C. Meadows, and I. Cervesato. Dolev-Yao is no better than Machiavelli. In P. Degano, editor, *Proceedings of the First Workshop on Issues in the Theory of Security (WITS'00)*, pages 87–92, 2000.

A An Attack to the Needham-Schroeder Protocol

$$
\begin{array}{lllll}
1. & alice \rightarrow intruder & : & \{N_a, alice\}_{K_{intruder}} \\
2. & intruder \rightarrow bob & : & \{N_a, alice\}_{K_{bob}} \\
3. & bob \rightarrow alice & : & \{N_a, N_b\}_{K_{alice}} \\
4. & alice \rightarrow intruder & : & \{N_b\}_{K_{intruder}} \\
5. & intruder \rightarrow bob & : & \{N_b\}_{K_{bob}}
\end{array}
$$

$$
\begin{array}{l}
\mathtt{pr}(\mathtt{alice}, \mathtt{init}) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{init}) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \xrightarrow{\mathtt{p_1})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step1}(\mathtt{na}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{init}) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{n}(\mathtt{pubk}(\mathtt{intruder}), \mathtt{mess}(\mathtt{na}, \mathtt{alice})) \xrightarrow{\mathtt{i_3})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step1}(\mathtt{na}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{init}) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{dec}(\mathtt{mess}(\mathtt{na}, \mathtt{alice})) \xrightarrow{\mathtt{i_5})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step1}(\mathtt{na}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{init}) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \xrightarrow{\mathtt{i_7})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step1}(\mathtt{na}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{init}) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{comp}(\mathtt{mess}(\mathtt{na}, \mathtt{alice}) \xrightarrow{\mathtt{i_8})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step1}(\mathtt{na}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{init}) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{n}(\mathtt{pubk}(\mathtt{bob}), \mathtt{mess}(\mathtt{na}, \mathtt{alice}) \xrightarrow{\mathtt{p_2})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step1}(\mathtt{na}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{step2}(\mathtt{na}, \mathtt{nb}, \mathtt{alice})) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{n}(\mathtt{pubk}(\mathtt{alice}), \mathtt{mess}(\mathtt{na}, \mathtt{nb}) \xrightarrow{\mathtt{p_3})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{na}, \mathtt{nb}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{step2}(\mathtt{na}, \mathtt{nb}, \mathtt{alice})) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{n}(\mathtt{pubk}(\mathtt{intruder}), \mathtt{mess}(\mathtt{nb})) \xrightarrow{\mathtt{i_3})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{na}, \mathtt{nb}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{step2}(\mathtt{na}, \mathtt{nb}, \mathtt{alice})) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{dec}(\mathtt{mess}(\mathtt{nb})) \xrightarrow{\mathtt{i_4})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{na}, \mathtt{nb}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{step2}(\mathtt{na}, \mathtt{nb}, \mathtt{alice})) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{m}(\mathtt{nb}) \xrightarrow{\mathtt{i_6})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{na}, \mathtt{nb}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{step2}(\mathtt{na}, \mathtt{nb}, \mathtt{alice})) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{m}(\mathtt{nb}) \mid \mathtt{comp}(\mathtt{mess}(\mathtt{nb})) \xrightarrow{\mathtt{i_8})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{na}, \mathtt{nb}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{step2}(\mathtt{na}, \mathtt{nb}, \mathtt{alice})) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{m}(\mathtt{nb}) \mid \mathtt{n}(\mathtt{pubk}(\mathtt{bob}), \mathtt{mess}(\mathtt{nb})) \xrightarrow{\mathtt{p_4})} \\
\mathtt{pr}(\mathtt{alice}, \mathtt{step3}(\mathtt{na}, \mathtt{nb}, \mathtt{intruder})) \mid \mathtt{pr}(\mathtt{bob}, \mathtt{step4}(\mathtt{na}, \mathtt{nb}, \mathtt{alice})) \mid \mathtt{pr}(\mathtt{intruder}, \mathtt{init}) \mid \\
\quad \mathtt{m}(\mathtt{na}) \mid \mathtt{m}(\mathtt{alice}) \mid \mathtt{m}(\mathtt{nb})
\end{array}
$$

B Soundness and Completeness of the Symbolic Fixpoint Semantics

We briefly recall some definitions and state a soundness and completeness result on the multiset rewriting formalism informally described in this paper, hereafter called **MSLO**. Details and proofs can be found in [Boz01].

Definition B.1 (The MSLO logic) *An* **MSLO** *theory is made up of clauses of the form* $F_1 \mid \ldots \mid F_n \; :- \; \forall X_1 \ldots \forall X_k \; (G_1 \mid \ldots \mid G_m)$ *and facts of the form* $F_1 \mid \ldots \mid F_n$*, where* F_i*,* G_i *are atomic formulas of the form* $p(t_1, \ldots, t_l)$ *and* X_i *are variables.*

The top-down (proof-theoretical) semantics for **MSLO** is defined through a sequent system [Boz01]. The turnstyle $\vdash_\Sigma$ denotes the corresponding derivability relation, which is the counterpart of the (forward) reachability relation of the **MSLO** specification of a transition system.

Definition B.2 (Operational Semantics) *Given an* **MSLO** *theory* T *on a signature* Σ*, its operational semantics, denoted* $O(T)$*, is given by*

$$O(T) = \{\mathcal{A} \mid \mathcal{A} \textit{ is a multiset of (non ground) atomic formulas and } T \vdash_\Sigma \mathcal{A}\}.$$

Definition B.3 (Interpretation) *Given an* **MSLO** *theory on a signature* Σ*, the Herbrand base is the set of all multisets of non-ground atomic formulas over* Σ*. An interpretation is any subset of the Herbrand base.*

Definition B.4 (Instance and Upward Closure Operators) *Given an interpretation* I *and a signature* Σ*, we define the operators* $Inst_\Sigma$ *and* Up_Σ *as follows:*

$$Inst_\Sigma(I) = \{\mathcal{A}\theta \mid \mathcal{A} \in I, \; \theta \textit{ substitution}\}, \quad Up_\Sigma(I) = \{\mathcal{A} + \mathcal{C} \mid \mathcal{A} \in I\}.$$

Given an **MSLO** theory T on a signature Σ, we denote by Sig_T the set of signatures which comprise at least the symbols in Σ (and possibly new constants introduced by universal quantification).

Definition B.5 *Given an interpretation* I*, its denotation* $[\![I]\!]$ *is the family* $([\![I]\!]_\Sigma)_{\Sigma \in Sig_T}$ *defined as follows:* $[\![I]\!]_\Sigma = Inst_\Sigma(Up_\Sigma(I))$ *(or, equivalently,* $[\![I]\!]_\Sigma = Up_\Sigma(Inst_\Sigma(I))$*).*

Definition B.6 (Symbolic Fixpoint Operator) *Given an* **MSLO** *theory* T *on a signature* Σ *and an interpretation* I*, the symbolic fixpoint operator* S_T *is defined as follows:*

$$S_T(I) = \{(\widehat{H} + \mathcal{C})\theta \mid \textit{there exists } (H \; :- \; G) \in T \textit{ (variant) s.t. } I \Vdash_\Sigma G \blacktriangleright \mathcal{C} \blacktriangleright \theta\}.$$

The judgment $\Vdash_\Sigma$ is the *abstract satisfiability judgment* defined in [Boz01], $\mathcal{C}$ and θ are an output multiset and an output substitution, $+$ denotes multiset union, and $\mathcal{M}\theta$ denotes the application of a substitution θ to a multiset $\mathcal{M}$. Given $H = A_1 \mid \ldots \mid A_n$, the notation $\widehat{H}$ stands for the multiset $\{A_1, \ldots, A_n\}$.

We have the following main theorem, which states soundness and completeness of the fixpoint (bottom-up) semantics our verification algorithm is based on, w.r.t. the operational (top-down) semantics for **MSLO**.

Theorem B.7 (Soundness and Completeness) *For every* **MSLO** *theory* T *on a signature* Σ*,* $O(T) = [\![\mathit{lfp}(S_T)]\!]_\Sigma$*.*