ITC
irst

CENTRO PER LA RICERCA
SCIENTIFICA E TECNOLOGICA

38050 Povo (Trento), Italy
Tel.: +39 0461 314312
Fax: +39 0461 302040
e−mail: prdoc@itc.it − url: http://www.itc.it

AUTOMATIC VERIFICATION OF SECRECY PROPERTIES FOR LINEAR LOGIC SPECIFICATIONS OF CRYPTOGRAPHIC PROTOCOLS

Bozzano M., Delzanno G.

July 2003

Technical Report # 0307−11

Article Submitted to Journal of Symbolic Computation

Automatic Verification of Secrecy Properties for Linear Logic Specifications of Cryptographic Protocols

Marco Bozzano and Giorgio Delzanno

ITC - IRST, Via Sommarive 18, Povo, 38050 Trento, Italy,
bozzano@irst.itc.it
DISI, Università di Genova, via Dodecaneso 35, 16146 Genova, Italy,
giorgio@disi.unige.it

Abstract

In this paper we investigate the applicability of a *bottom-up evaluation strategy* for a first order fragment of linear logic we introduced in [Bozzano et al., 2002b] for the purposes of automated verification of *secrecy* in *cryptographic protocols*. Following [Cervesato et al., 1999], we use *multi-conclusion* clauses to represent the behaviour of agents in a protocol session, and we adopt the *Dolev-Yao* intruder model. In addition, universal quantification provides a formal and declarative way to express creation of *nonces*. Our approach is well suited to verify properties which can be specified by means of *minimal conditions*. 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 protocol. Furthermore, our approach can be used both to find attacks and to prove secrecy for a protocol. We apply our method to analyze several classical examples of authentication protocols. Among them we consider the *ffgg* protocol [Millen, 1999]. This protocol is a challenging case study in that it is free from *sequential attacks*, whereas it suffers from *parallel attacks* that occur only when at least two sessions are run in parallel. The other case studies are the Otway-Rees protocol and several formulation of the Needham-Schroeder protocol.

1. Introduction

Linear logic [Girard, 1987] provides a logical characterization of concepts and mechanisms peculiar to concurrency like *locality*, *recursion*, *non-determinism*, and *synchronization* [Andreoli and Pareschi, 1991, Miller, 1992, Kobayashi and Yonezawa, 1995]. Following the paradigm of *proofs as computations* proposed in [Andreoli, 1992, Miller, 1996], *provability* in fragments of linear logic can be used then as a formal tool to reason about behavioural aspects of *concurrent systems* (see e.g., [McDowell et al., 1996]). In other paradigms for concurrency like the theory of Petri Nets there exist however a number of consolidated *algorithmic techniques* for the validation of system properties. In [Bozzano et al., 2000, 2002a], we made a first attempt of relating these techniques with propositional fragments of linear logic, and, more precisely, with the linear logic programming language called LO [Andreoli and Pareschi, 1991]. LO was originally introduced as a theoretical foundation for extensions of *logic programming* languages. The appealing feature of this fragment, however, is that it can also be viewed as a rich *specification* language for protocols and concurrent systems. In fact, specification languages like Petri Nets and multiset rewriting over first order atomic formulas can be naturally embedded into *propositional* LO [Cervesato, 1995]. In [Bozzano et al., 2000], we established a connection between *provability* in LO and *reachability* of Petri Nets via the definition of an effective procedure to compute the set of linear logic *goals* (multisets of atomic formulas) that are consequences of a given propositional program. In other words we defined a *bottom-up** evaluation procedure for *propositional* programs. Our construction is based on the *backward reachability* algorithm of [Abdulla et al., 2000] used to decide the so called *control state reachability problem* of Petri Nets. The algorithm presented in [Bozzano et al., 2000] is defined, however, for the more general case of propositional LO specifications (i.e., with nested conjunctive and disjunctive goals).

A natural way of augmenting the expressivity of the specification language is to consider *first order fragments* of linear logic. First order formulas can be used, in fact, to *color* the internal state of processes with structured data [Andreoli and Pareschi, 1991, Miller, 1996]. The combination between first order formulas and linear connectives provides a well-founded interpretation of the dynamics in the evolution of the internal state of a process [Andreoli and Pareschi, 1991, Miller, 1992, 1996]. First order quantification in goal formulas has several interesting interpretations here: it can be viewed either as a sort of *hiding* operator in the

*According to the usual terminology in logic programming, *bottom-up* evaluation is intended to denote derivation of logical consequences of a program, starting from the axioms

style of π-calculus [Miller, 1992], or as a mechanism to generate *fresh names* as in [Cervesato et al., 1999].

In [Bozzano et al., 2002b, Bozzano, 2002] we defined a procedure for the *bottom-up* evaluation of first order LO *programs with universally quantified goals*. Via the connection between *provability* and *reachability* established in [Bozzano et al., 2000], we can view such an evaluation procedure as a validation technique for specifications of complex concurrent systems. The bottom-up evaluation procedure is based on an *effective* fixpoint operator and on a *symbolic* and *finite* representation of a potentially *infinite collection* of first order provable LO goals (multisets of atoms). The use of this symbolic representation is crucial when trying to prove properties of *parameterized* systems, i.e., systems in which the number of individual processes is left as a parameter of the specification like for *multi-agent protocols with multiple parallel sessions*.

New Contribution In this paper we investigate the applicability of the bottom-up evaluation strategy of [Bozzano et al., 2002b] for the purposes of automated validation of *authentication protocols*. The design and implementation of cryptographic protocols are difficult and error-prone. Authentication protocols should be reliable enough to be used in a potentially compromised environment. Exchanging *nonces*, i.e., fresh values, is a commonly used technique which is exploited in combination with cryptography to achieve authentication. Different approaches have been followed to specify and analyze protocols. An incomplete list include for instance using belief logics [Burrows et al., 1989], rewriting techniques [Denker et al., 1998, Cervesato et al., 1999, Cirstea, 2001], theorem proving [Paulson, 1998], logic programming [Meadows, 1996, Delzanno, 2001, Blanchet, 2001], and model-checking [Lowe, 1996, Marrero et al., 1997, Roscoe and Broadfoot, 1999]. Following [Cervesato et al., 1999], as specification language we will use *multi-conclusion* clauses to represent a given set of agents (called *principals*) executing *parallel* protocol sessions by exchanging messages over a network. We will use the *Dolev-Yao* intruder model and related message and cryptographic assumptions. Also, enriching linear logic specifications with universal quantification in goal formulas will provide a logical and clean way to express creation of *nonces*. In order to reason about security properties, we will apply our *general purpose* bottom-up evaluation scheme for first order linear logic. Our approach is well suited to verify properties which can be specified by means of *minimal 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). The resulting verification method has connections both with (symbolic) model checking [Abdulla et al., 2000] and with theorem proving [Andreoli, 1992]. 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).

We have built a prototype, written in standard ML, to implement the bottom up evaluation of LO programs (see [Bozzano, 2002]), which we have used to carry out some experiments using the approach previously described. In particular, in this paper we present and analyze well-known examples of authentication protocols taken from the literature on security, like the Needham-Schroeder protocol [Needham and Schroeder, 1978], a *corrected* version of Needham-Schroeder, and Otway-Rees [Otway and Rees, 1987]. In addition we report on the experiment with the *ffgg* protocol introduced by Millen in [Millen, 1999]. Millen's *ffgg* protocol is a challenging case study for the following reasons. First, the protocol is free from *sequential attacks*, whereas it suffers from *parallel attacks* that occur only when at least two sessions are run in parallel. Secondly, the scheme underlying *ffgg* can be generalized so as to obtain *higher order* attacks (i.e., attacks that need at least k sessions in parallel). Since we do not need to put a bound on the number of parallel sessions, the application of our method is sound for any instance of the protocol. Our experiments show that our methodology can be effective to analyze interesting aspects of authentication such as secrecy or confidentiality.

Related work A wide research area in security protocol analysis is related to rewriting. For instance, in [Cirstea, 2001] protocols are specified as rewriting theories which can be executed in the ELAN system. A similar approach is followed in [Denker et al., 1998], where the target executable language is instead Maude.

In [Jacquemard et al., 2000] the authors present an automatic compilation process from security protocol descriptions into rewrite rules. The resulting specifications are then executed using the daTac theorem prover. As a difference with [Denker et al., 1998], which is based on *matching*, the execution strategy of [Jacquemard et al., 2000] relies on *narrowing* and AC unification. Our approach, based on multiset unification, is clearly closer to the latter approach, although currently we do not support equational theories. All of the above approaches are limited to protocol debugging, therefore they can find attacks mounted on a given protocol, but they cannot be used to analyze correctness.

In [Genet and Klay, 2000] Genet and Klay use Term Rewriting Systems and tree

automata to build an over-approximation of the reachability set for an arbitrary number of protocol sessions. Tree automata are used to symbolically represent an arbitrary number of principals' local states. A rewriting system represents instead intruder and protocol rules. The over-approximation is computed via a completion procedure applied to automata and rewrite rules. The method can be used for secrecy but it might return false attacks. Furthermore, validity conditions on time-stamps cannot be modeled in this framework. Another approach which shares some similarity with ours is [Delzanno, 2001], 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, like us, and embedded implication to store the knowledge of agents. This approach is still limited to protocol debugging. Differently from our approach, all the above works are based on a *forward* search strategy, while effectiveness of our verification algorithm strongly relies on a *backward* search strategy.

An alternative approach to verifying security protocols is based on model checking. For instance, the FDR model checking tool was used by Lowe [Lowe, 1996] to analyze the Needham-Schroeder public-key protocol. Other works which fall into this class are [Marrero et al., 1997, Roscoe and Broadfoot, 1999]. All these approaches have in common the use of some kind of abstraction to transform the original problem into a *finite-state* model-checking problem, which is then studied by performing a *forward* reachability analysis. Using a finite-state approximation has the advantage of guaranteeing termination, however it only allows one to analyze a fixed number of concurrent protocol runs, an approach which is infeasible as this number increases. The restriction to finite-state models, however, is not always necessary. As shown in [Amadio and Lugiez, 2000, Boreale, 2001, Chevalier and Vigneron, 2002, Millen and Shmatikov, 2001, Rusinowitch and Turuani, 2001], attacks for a bounded number of sessions and Dolev-Yao intruders can algorithmically be discovered. In these approaches *constraints* relating the knowledge of the intruder and messages sent by honest principals are incrementally collected during a symbolic protocol execution and solved only after the session is completed. If a solution to the resulting constraint exists, then the intruder has a way to break the protocol. Lazy data structures can also be used to explore the infinite-state search space of an insecure network in a demand-driven manner [Basin, 1999]. As a difference from the above mentioned approaches, we use a *symbolic* representation for *infinite* sets of states and a *backward* reachability verification procedure, which avoids putting limitations on the number of parallel sessions.

Theorem proving techniques are used in [Paulson, 1998], where protocols are

inductively defined as sets of traces, and formally analyzed using the theorem prover Isabelle. Here, analysis is a *semi-automatic* process which can take several days. The NRL protocol analyzer [Meadows, 1996] provides a mixed approach. It is based on protocol specifications given via Prolog rules, and enriched via a limited form of term rewriting and narrowing to manage symbolic encryption equations. Similarly to us, verification is performed by means of a symbolic model-checker which relies on a *backward* evaluation procedure which takes as input a set of insecure states. The analyzer needs to be fed with some inductive lemma, in the same way theorem provers need to be guided during the proof search process.

The Athena algorithm proposed in [Song, 1999] automatizes correctness proofs for models based on *Strand Spaces*. In Athena infinite sets of *bundles* (protocol executions), are symbolically represented via *semi-bundles*, a partially ordered event structure made parametric via the use of first order variables, and the *goal binding* relation, a symbolic representation of several possible 'useless' steps of the intruder that may occur between a send- and a receive-event. Semi-bundles are incrementally refined during backward search, until all receive-events are bound to some send-event. Similarly to our method and to NRL protocol analyzer, Athena can be used both for debugging and verification, and needs pruning techniques (e.g., the unreachability lemma of [Song, 1999] or restrictions on the capability of the intruder as in the example shown in the appendix of [Song, 1999]) to achieve termination. However, in Athena freshness of nonces is expressed at the *meta level* in the protocol model, and it is controlled during the analysis with special checks on the *origin* of messages. On the contrary, in our method the use of universal quantification allows us to reason about freshness inside the logic.

In [Blanchet, 2001], Blanchet proposes an optimized specification of security protocols based on an "attacker view" of protocol security, specified by means of Prolog rules, as in [Meadows, 1996]. The approach 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 protocol notation. Ensuring faithfulness between the intended semantics of a protocol and its specification is necessary to prove correctness. Also, with respect to [Blanchet, 2001], 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.

The use of Horn logic is also a characteristic of Cohen's TAPS verifier [Cohen, 2000]. Differently from Blanchet's method, the TAPS verifier generates however *state-dependent* invariants from the protocol model. These invariants are combines with lemmas provided by the user. The first order theorem prover SPASS is used then to discharge the proof obligations produced by a proof system used to simplify the invariants according to the capability of the intruder. While TAPS is very fast and effective, it does not produce readable counterexamples, a feature that might be very important when searching for justifications of potential attacks.

Concerning the process of translation from the usual informal notation for protocols, which we plan to study as future work, existing approaches include `Casper` [Lowe, 1998], a compiler from protocol specifications into the CSP process algebra, oriented towards verification in FDR, and `CAPSL` [Millen, 1997], a specification language which can be compiled into an intermediate language and used to feed tools like Maude [Denker et al., 1998] or the NRL analyzer [Meadows, 1996]. Finally, [Jacquemard et al., 2000] presents an automatic compilation process into rewriting rules which is able to manage infinite-state models.

Concerning the application of linear logic to verification, we would like to mention [Fages et al., 2001], where phase semantics is used to prove properties of specification of concurrent constraint programs. The phase semantics for LO proposed by Andreoli could be the possible connection between the manual 'semantic-driven' method of [Fages et al., 2001] and our automated 'syntactic-driven' method that could be interesting to investigate.

This paper extends the preliminary results discussed in [Bozzano and Delzanno, 2002] where the focus was on the *ffgg* protocol. The technical details of the bottom-up evaluation strategy for $\mathrm{LO}_\forall$ programs is described in [Bozzano et al., 2002b] (where, as practical example, we have studied a parameterized *mutual exclusion* protocol). The first author's PhD thesis [Bozzano, 2002] also contains a detailed presentation and proofs for the results presented in this paper. Some preliminary results (e.g. Needham-Schroeder protocol) were also discussed in [Bozzano, 2001].

Structure of the paper In Section 2 we present some background material on authentication. In Section 3 we introduce the language LO with universally quantified goals. In Section 4 we illustrate our encoding of authentication protocols in linear logic, and we exemplify it in Section 5, where the *ffgg* protocol is presented. In Section 6 we discuss the adequacy of the protocol encoding in linear logic. In Section 7 we discuss the application of our bottom-up evaluation algorithm for the verification of security properties of authentication protocols. Section 8 collects

some examples of authentication protocols, and discusses their specification and verification. Finally, in Section 9 we draw some conclusions.

2. Background on Authentication

In this section we briefly discuss some background on authentication protocols and we fix some notations which we will use in the rest of the paper.

Authentication protocols are used to coordinate the activity of different parties (e.g., users, hosts, or processes) operating over a network. These parties are usually referred to as *principals* in security literature. An authentication protocol generally consists of a *sequence* of messages exchanged between two or more principals (e.g., two users and a coordinating entity acting as a server). The form and number of exchanged messages is usually fixed in advance and must conform to a specific format. In general, a given principal can take part into a given protocol run in different ways, e.g., as the *initiator* of the protocol or the *responder* (it is usually said that a principal can have different *roles*). Often, a principal is allowed to take part into different protocol runs simultaneously, possibly with different roles.

The design of authentication protocols must take into account the possibility of messages to be intercepted, and the presence of malicious agents which can impersonate the role of an honest principal. One of the key issues in authentication is to ensure *confidentiality*, i.e., to avoid private information being disclosed to unauthorized clients. Another issue is to prevent malicious principals from cheating by impersonating other principals. A principal should have enough information to ensure that every message received has been created *recently* (as part of the current protocol run) and by the principal who claims to have sent it (replaying of old messages should be detected). Authentication protocols must be designed in such a way to be resistant to every possible form of *attack*. In particular, interception of messages can prevent completion of a protocol run, but should never cause a leak of information or compromise security.

Cryptographic primitives are a fundamental, though not sufficient, ingredient of authentication protocols. A message to be transmitted over a network is usually referred to as *plaintext*. The task of a cryptographic algorithm is to convert the given message to a form which is unintelligible to anyone else except the intended receiver. The conversion phase is called *encryption* and usually depends on an additional parameter known as *encryption key*, whereas the encoded message is referred to as *ciphertext*. The reverse phase of decoding is called *decryption*, and usually requires possession of the corresponding *decryption key*.

Authentication protocols (see [Clark and Jacob, 1997] for a survey) are usually

classified depending on the cryptographic approach taken, e.g., symmetric key or public key protocols. Furthermore, a distinction is also made between protocols which use one or more *trusted third parties* (e.g., a central distribution key server) and protocols which do not. In *symmetric key cryptography*, security of communication requires the keys to be kept secret between the relevant principals, whereas in *public key cryptography*, each principal A has a pair of keys, the first one being the *public* key, and the other the *private* key. The public key is made available and can be used to encrypt messages for principal A, whereas the private key is only known to A, who can use it to decrypt incoming messages. Some public key algorithm allow the private key to be used for encryption and the public key to be used for decryption. This mechanism is used to guarantee *authenticity* (rather than *confidentiality*) of messages.

Authentication protocols can be compromised by different forms of *attacks* (see also [Clark and Jacob, 1997]). For instance, *freshness attacks* typically take place when a principal is induced to accept, as part of the current protocol run, an old message which is currently being replayed by a malicious intruder; *type flaw attacks* take place when a principal is induced to erroneously interpret the structure of the current message; *parallel session attacks* are usually carried out by a malicious intruder which forms messages for a given protocol run using messages coming another legitimate session which is executed concurrently; *implementation dependent attacks* are very subtle and can depend on a number of ways a given protocol is implemented.

2.1. The Dolev-Yao Intruder Model

Most formal approaches to protocol specification and analysis, including ours, are based on a set of simplifying assumptions, which is known as the *Dolev-Yao* intruder model. This model has been developed on the basis of some assumptions described in [Needham and Schroeder, 1978] and [Dolev and Yao, 1983]. According to this model, messages are considered as indivisible abstract values, instead of sequences of bits. Furthermore, the details of the particular crypto-algorithm used are abstracted away, giving rise to a *black-box* model of encryption (*perfect encryption*). This set of assumptions simplifies protocol analysis, although it has the drawback of preventing the discovery of implementation dependent attacks.

The Dolev-Yao intruder model consists of a set *conservative* assumptions on the potentialities of any possible attacker. Typically, 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 modifying them. The intruder works by *decomposing* messages (provided he/she knows the key they are encrypted with), and *composing* new messages. In general, the intruder knows the identity and, in the case of public-key encryption, the public keys of the other principals. The intruder is supposed *not* to know the *private* keys of the other principals, unless they have been disclosed in some way.

It has been proved that the Dolev-Yao intruder is, so to say, the *most powerful attacker* [Cervesato, 2001a] for the given model under consideration (e.g., perfect encryption), in the sense that it can simulate the activity of any other possible attacker. Furthermore, in [Syverson et al., 2000] it has been proved that it is not restrictive to consider a single Dolev-Yao intruder instead of multiple ones.

2.2. An Informal Protocol Notation

In the literature on security, protocols are usually presented by means of an informal notation. We explain this notation illustrating the so-called Needham-Schroeder public-key authentication protocol [Needham and Schroeder, 1978] (see also Section 8.2). The protocol, in the usual notation, is as follows.

$$
\begin{array}{lrcl}
1. & A \rightarrow S & : & A, B \\
2. & S \rightarrow A & : & \{K_b, B\}_{K_s^{-1}} \\
3. & A \rightarrow B & : & \{N_a, A\}_{K_b} \\
4. & B \rightarrow S & : & B, A \\
5. & S \rightarrow B & : & \{K_a, A\}_{K_s^{-1}} \\
6. & B \rightarrow A & : & \{N_a, N_b\}_{K_a} \\
7. & A \rightarrow B & : & \{N_b\}_{K_b}
\end{array}
$$

The protocol is run to achieve authentication between two principals A and B. A central server S is in charge of distributing the public keys of principals. Messages 3, 6, and 7 are the core of the protocol, while the purpose of messages 1, 2, 4 and 5 is to get public keys from the central authority. The notation $\{M\}_K$ indicates a message with content M and encrypted with a key K. Also, by convention $A, B, \ldots$ indicate principal identifiers, K_a and K_b denote, respectively, A's and B's public keys, and K_s^{-1} is the private key of server S (encryption with the private key is used to ensure authenticity).

Now, the protocol has the following structure. A given principal A acts as *initiator* of the protocol, and asks the central authority for B's public key (message 1). The central authority sends back to A the required key (message 2: B's identity is included in the message to prevent attacks based on diverting key deliveries).

Principal A creates a *nonce* (i.e., a newly generated value), called N_a, and sends it to B together with its own identity (message 3), encrypting the message with B's public key. Upon receiving this message, principal B decrypts the message, and in turn asks the central authority for the public key of A (message 4). After getting the server's reply (message 5), principal B generates a new nonce N_b, and sends both nonces, N_a and N_b, to A, encrypting the message with A's key (message 6). When A gets this message, a check is made that it contains the previously generated nonce N_a, and, if so, a new message, encrypted with B's key and including the last nonce N_b, is sent to B (message 7). The protocol is successfully completed provided B gets 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. The use of *nonces* is ubiquitous in authentication protocols. Intuitively, a nonce should be considered as some sort of *random* and *unguessable* value, whose purpose is to prevent a malicious intruder from attempting to break a given protocol by sending messages and pretending they have been generated by someone else. To exemplify, an attacker could possibly pretend to be principal B, intercept message 6 and replace it with a different message. However, the message created by the attacker will never be accepted as a legitimate message by A, unless it contains the nonce N_a. Unfortunately (for the attacker) nonce N_a is not known except to B (and A, of course), because only B can decrypt message 3. 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 principals. However, under certain conditions, this protocol fails to achieve authentication [Lowe, 1995]. We will discuss this point in Section 8.2.

3. The Specification Language $\mathbf{LO}_\forall$

LO [Andreoli and Pareschi, 1991] is a logic programming language based on a fragment of LinLog [Andreoli, 1992]. Its mathematical foundations lie on a proof-theoretical presentation of a fragment of linear logic defined over the linear connectives $\multimap$ (*linear implication*, we use the reversed notation $H \circ\!\!- G$ for $G \multimap H$), $\&$ (*additive conjunction*), $\parr$ (*multiplicative disjunction*), and the constant $\top$ (*additive identity*). In this section we present the proof-theoretical semantics, corresponding to the usual *top-down* operational semantics for traditional logic programming languages, for an extension of LO. First of all, we consider a slight extension of LO which admits the constant $\bot$ in goals and clause heads.

More importantly, we allow the universal quantifier to appear, possibly nested, in goals. This extension is inspired by *multiset rewriting with universal quantification* [Cervesato et al., 1999]. The resulting language will be called $\mathrm{LO}_\forall$ hereafter.

Following [Andreoli and Pareschi, 1991], we give the following definitions. Let Σ be a signature with predicates including a set of constant and function symbols $\mathcal{L}$ and a set of predicate symbols $\mathcal{P}$, and let $\mathcal{V}$ be a denumerable set of variables. An atomic formula over Σ and $\mathcal{V}$ has the form $p(t_1, \ldots, t_n)$ (with $n \geq 0$), where $p \in \mathcal{P}$ and $t_1, \ldots, t_n$ are (non-ground) terms in $T_\Sigma^\mathcal{V}$. We denote the set of such atomic formulas as $A_\Sigma^\mathcal{V}$, and the set of *ground* (i.e., without variables) atomic formulas as A_Σ. Finally, given a formula F, $FV(F)$ is the set of free variables of F.

The classes of **G**-formulas (goal formulas), and **D**-formulas (multi-headed clauses) over Σ and $\mathcal{V}$ are defined by the following grammar:

$$\begin{aligned}
\mathbf{G} &::= \mathbf{G} \parr \mathbf{G} \mid \mathbf{G} \;\&\; \mathbf{G} \mid \forall x.\mathbf{G} \mid \mathbf{A} \mid \top \mid \bot \\
\mathbf{H} &::= \mathbf{A} \parr \ldots \parr \mathbf{A} \mid \bot \\
\mathbf{D} &::= \forall(\mathbf{H} \circ\!\!- \mathbf{G})
\end{aligned}$$

where **A** stands for an atomic formula over Σ and $\mathcal{V}$, and $\forall(H \circ\!\!- G)$ stands for $\forall x_1 \ldots x_k.(H \circ\!\!- G)$, with $\{x_1, \ldots, x_k\} = FV(H \circ\!\!- G)$.

An $\mathrm{LO}_\forall$ program over Σ and $\mathcal{V}$ is a set of **D**-formulas over Σ and $\mathcal{V}$. A multiset of goal formulas will be called a *context* hereafter. In the following we usually omit the universal quantifier in **D**-formulas, i.e., we consider free variables as being *implicitly* universally quantified. Let Σ_P be a signature with predicates and $\mathcal{V}$ a denumerable set of variables. An $\mathrm{LO}_\forall$ sequent has the form

$$P \vdash_\Sigma G_1, \ldots, G_k,$$

where P is an $\mathrm{LO}_\forall$ program over Σ_P and $\mathcal{V}$, $G_1, \ldots, G_k$ is a context (i.e., a multiset of goals) over Σ and $\mathcal{V}$, and Σ is a signature such that $\Sigma_P \subseteq \Sigma$. We will use Sig_P to denote the set of all possible extensions of Σ_P (as explained in Section 3.1, the top-down construction of LO proofs, and in particular the rule dealing with the universal quantifier, may require to extend the original signature).

3.1. Top-down Provability

We now define provability in $\mathrm{LO}_\forall$. Let Σ be a signature with predicates and $\mathcal{V}$ a denumerable set of variables. Given an $\mathrm{LO}_\forall$ program P over Σ_P and $\mathcal{V}$, and a signature Σ such that $\Sigma_P \subseteq \Sigma$, the set of ground instances of P, denoted $Gnd(P)$, is defined as follows:

$$Gnd(P) = \{\ (H \circ\!\!- G)\,\theta \mid \forall(H \circ\!\!- G) \in P\ \},$$

$$\frac{}{P \vdash_\Sigma \top, \Delta}\ \top_r \qquad \frac{P \vdash_\Sigma G_1, G_2, \Delta}{P \vdash_\Sigma G_1 \parr G_2, \Delta}\ \parr_r \qquad \frac{P \vdash_\Sigma G_1, \Delta \quad P \vdash_\Sigma G_2, \Delta}{P \vdash_\Sigma G_1 \,\&\, G_2, \Delta}\ \&_r$$

$$\frac{P \vdash_\Sigma \Delta}{P \vdash_\Sigma \bot, \Delta}\ \bot_r \qquad \frac{P \vdash_{\Sigma,c} G[c/x], \Delta}{P \vdash_\Sigma \forall x.G, \Delta}\ \forall_r\ (c \notin \Sigma) \qquad \frac{P \vdash_\Sigma G, \mathcal{A}}{P \vdash_\Sigma \widehat{H}, \mathcal{A}}\ bc\ \ (H \multimap G\ \in Gnd(P))$$

Figure 1: A proof system for $\mathrm{LO}_\forall$

where θ is a grounding substitution for $H \multimap G$ (i.e., it maps free variables in $FV(H \multimap G)$ to ground terms in T_Σ). The execution of a multiset of **G**-formulas $G_1, \ldots, G_k$ over Σ in P corresponds to a *goal-driven* proof for the $\mathrm{LO}_\forall$ sequent $P \vdash_\Sigma G_1, \ldots, G_k$.

The operational semantics of $\mathrm{LO}_\forall$ is given via the *uniform* (*focusing*) [Andreoli, 1992] proof system presented in Figure 1, where P is a set of clauses, $\mathcal{A}$ is a multiset of atomic formulas, and Δ is a multiset of **G**-formulas. We have used the notation $\widehat{H}$, where H is a linear disjunction of atomic formulas $a_1 \parr \ldots \parr a_n$, to denote the multiset $a_1, \ldots, a_n$ (by convention, $\widehat{\bot} = \epsilon$, where ϵ is the empty multiset). We say that G is provable from P if there exists a proof tree, built over the proof system of Figure 1, with root $P \vdash_\Sigma G$, and such that every branch is terminated with an instance of the $\top_r$ axiom. The proof system of Figure 1 is a specialization of more general uniform proof systems for linear logic like Andreoli's focusing proofs [Andreoli, 1992] and Forum [Miller, 1996]. Rule *bc* is analogous to a backchaining (resolution) step in traditional logic programming languages. Note that according to the concept of resolution explained above, *bc* can be executed only if the right-hand side of the current LO sequent consists of atomic formulas. As an instance of rule *bc*, we get the following proof fragment, which deals with the case of clauses with empty head:

$$\frac{\vdots}{\dfrac{P \vdash_\Sigma \mathcal{A}, G}{P \vdash_\Sigma \mathcal{A}}\ bc}$$

$$provided\ \bot \multimap G \in Gnd(P)$$

Given that clauses with empty head are always applicable in atomic *contexts*, the degree of non-determinism they introduce in proof search is usually considered unacceptable [Miller, 1996] and in particular they are forbidden in the original

presentation of LO [Andreoli and Pareschi, 1991]. However, our computational model, i.e., bottom-up evaluation, does not suffer this drawback. Clauses with empty head often allow more flexible specifications (see, e.g., Section 4).

LO clauses having the form $H \circ\!\!- \top$ play the same role as axioms (i.e., unit clauses) for Horn programs. In fact, when a backchaining step over such a clause is possible, we get a *successful* (branch of) computation, independently of the current *context* $\mathcal{A}$, as shown in the following proof scheme:

$$\dfrac{\dfrac{}{P \vdash_\Sigma \top, \mathcal{A}}\ \top_r}{P \vdash_\Sigma \widehat{H}, \mathcal{A}}\ bc$$

$$\textit{provided } H \circ\!\!- \top \in \mathit{Gnd}(P)$$

The previous observation can be stated as follows. Let $\preccurlyeq$ denote the multiset inclusion relation. Given an $\mathrm{LO}_\forall$ program P and two multisets of goals Δ, Δ' such that $\Delta \preccurlyeq \Delta'$, if $P \vdash_\Sigma \Delta$ then $P \vdash_\Sigma \Delta'$ (we refer to this property as *admissibility* of the weakening rule [Bozzano et al., 2002b]).

Finally, rule $\forall_r$ can be used to *dynamically* introduce new *names*. The initial signature Σ must contain at least the constant, function, and predicate symbols of a given program P, and it can dynamically grow thanks to rule $\forall_r$. Every time rule $\forall_r$ is fired, a new constant c is added to the current signature, and the resulting goal is proved in the new one. The idea is that all terms appearing on the right-hand side of a sequent are implicitly assumed to range over the relevant signature. This behaviour is standard in logic programming languages [Miller et al., 1991].

EXAMPLE 3.1: *Let Σ be a signature with a constant symbol a, a function symbol f and predicate symbols p, q, r, s. Let P be the program consisting of the clauses*

1. $r(w) \circ\!\!- q(f(w)) \parr s(w)$
2. $s(z) \circ\!\!- \forall x.p(f(x))$
3. $\bot \circ\!\!- q(u) \,\&\, r(v)$
4. $p(x) \parr q(x) \circ\!\!- \top$

The goal $s(a)$ is provable from P. The corresponding proof is shown in Figure 2 (where $bc^{(i)}$ denotes backchaining rule over clause number i of P). Notice that the notion of ground instance *is relative to the current signature. For instance, backchaining over clause 3 is possible because the corresponding signature contains the constant c, and therefore $\bot \circ\!\!- q(f(c)) \,\&\, r(c)$ is a valid instance of clause 3.*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash_{\Sigma,c} \top}\,\top_r}{P \vdash_{\Sigma,c} p(f(c)), q(f(c))}\,bc^{(4)} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash_{\Sigma,c} \top, s(c)}\,\top_r}{P \vdash_{\Sigma,c} p(f(c)), q(f(c)), s(c)}\,bc^{(4)}}{P \vdash_{\Sigma,c} p(f(c)), q(f(c)) \parr s(c)}\,\parr_r}{P \vdash_{\Sigma,c} p(f(c)), r(c)}\,bc^{(1)}}{P \vdash_{\Sigma,c} p(f(c)), q(f(c)) \,\&\, r(c)}\,\&_r}{P \vdash_{\Sigma,c} p(f(c))}\,bc^{(3)}}{P \vdash_{\Sigma} \forall x.p(f(x))}\,\forall_r}{P \vdash_{\Sigma} s(a)}\,bc^{(2)}$$

Figure 2: An example of $\mathrm{LO}_\forall$ proof

In the rest of the paper we will focus our attention on an *observational semantics* that captures the provability of a restricted form of $\mathrm{LO}_\forall$ goals, namely goals consisting of a multiset of ground atomic formulas. Specifically, given a program P we define its *top-down* operational semantics as follows.

Definition (Operational Semantics): Given an $\mathrm{LO}_\forall$ program P, its operational semantics, denoted $\mathcal{O}(P)$, is given by

$$\mathcal{O}(P) = \{\ \mathcal{A}\ |\mathcal{A} \text{ multiset of ground atoms in } A_{\Sigma_P},\ P \vdash_{\Sigma_P} \mathcal{A}\ \}.$$

Notice that a multiset $\mathcal{A} = A_1, \ldots, A_k$ (in the r.h.s. of a sequent) is logically equivalent to the multiplicative disjunction $A_1 \parr \ldots \parr A_k$.

4. Specifying Authentication Protocols in $\mathrm{LO}_\forall$

In this section we introduce, rather informally, some generalities about our encoding of protocols in linear logic. Our specification language has a natural correspondence with the multiset rewriting formalism MSR proposed in [Cervesato et al., 1999]. This connection will be formally discussed in Section 6.

First of all, we need a representation for the entities (e.g., principals and messages) involved. In particular, we will use a notation like

$$pr(id, s)$$

to denote a principal with identifier id and internal state s. The internal state s can store information about an ongoing execution of a protocol (e.g.,, the identifier of another principal, which step of the protocol has been executed, the *role* of the principal, and so on). Typically, the state s will be a term like $init$ (indicating the initial state of a principal, before protocol execution), or a term like

$$step_i(data),$$

where the constructor $step_i$ denotes which is the last step executed and $data$ represents the internal data of a principal. We allow more than one atom $pr(id, \_)$ inside a given configuration. In this way, we can model the possibility of a given principal to take part into different protocol runs, possibly with different *roles*. Messages sent over a given network can in turn be represented by terms like

$$n(mess\_content),$$

where $mess\_content$ is the content of the message. Depending on the particular protocol under consideration, we can fix a specific format for messages. For instance, a message encrypted with the public key of a principal a could be represented as the term $enc(pubk(a), mess\_content)$.

Finally, we will use the Dolev-Yao *intruder* model (see [Cervesato et al., 1999]) and the associated assumptions. In particular, we use terms such as

$$m(inf)$$

to represent the information in possession of the intruder (m stands for the internal *memory* of the intruder). At any given instant of time, we can think of the current *state* of a given system as a *multiset* of atoms representing principals and messages currently on the network, and the intruder knowledge.

Following [Cervesato et al., 1999], 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 who tries to break the protocol. In addition, it is possible to have additional rules for the environment. Rules assume the general format

$$F_1 \parr \ldots \parr F_n \multimap \forall x_1 \ldots \forall x_k.\ G_1 \parr \ldots \parr G_m$$

where F_i, G_i are atomic formulas (representing e.g., principals or messages), x_i are variables, and all free variables are implicitly universally quantified. As explained in Section 3, the standard semantics for the universal quantifier requires new

values to be chosen before application of a rule. We use this behaviour to encode nonce generation during protocol runs. As a result, we get *for free* the assumption (required by the Dolev-Yao model) that nonces are not *guessable*.

We allow a *partial* specification of the set of *initial states*. This strategy is more flexible in that it may help us to find additional hypotheses under which a given attack might take place. As a general rule, the partial specification of the initial states we have chosen requires every principal to be in his/her initial state (represented by the term *init*) at the beginning of protocol execution.

We conclude this section by collecting together some rules which are common to *all* the examples presented in the rest of the paper. We will use these conventions: free variables inside a rule are always implicitly universally quantified, and variables are written as *upper-case* identifiers. We have two rules for the environment:

$$\begin{array}{ll} e_1) & \bot \circ\!\!- \forall ID.(pr(ID, init)) \\ e_2) & pr(Z,S) \circ\!\!- pr(Z,S) \parr pr(Z, init) \end{array}$$

The first rule allows the *non-deterministic* creation of new principals (we use the universal quantifier to generate new identifiers for them), whereas the second one allows creation of a new instance of a given principal (this allows a principal to start another execution of a given protocol with a new and possibly different *role*). Both rules can be fired at run-time, i.e., during the execution of a given protocol. Thus, we will always work in an *open* environment with multiple sessions running in parallel between several agents. We use the term *init* to denote the initial state of any given principal. We also have the following two rules for the intruder theory:

$$\begin{array}{ll} t_1) & pr(Z,S) \circ\!\!- pr(Z,S) \parr m(Z) \\ t_2) & \bot \circ\!\!- \forall N.(m(N)) \end{array}$$

The first rule allows the intruder to store a principal identifier, whereas the second one formalizes the capability of the intruder to generate new values (e.g., nonces).

To exemplify the protocol encoding introduced in this section, in the following section we will present the *ffgg* protocol. Other examples of protocols and their analysis will be discussed in Section 8.

5. An Example: The Millen's *ffgg* protocol

Although an artificial protocol, Millen's *ffgg* protocol [Millen, 1999] provides an example of a *parallel session* attack, which requires running at least two processes for the same role. It has been proved [Millen, 1999] that no *serial* attacks exist,

$p_1)$ $pr(A, init) \parr pr(B, init) \circ\!\!- pr(A, step1(B)) \parr pr(B, init) \parr n(plain(A))$

$p_2)$ $pr(B, init) \parr n(plain(A)) \circ\!\!- \forall N1.\forall N2.(pr(B, step2(A, N1)) \parr n(plain(N1, N2)))$

$p_3)$ $pr(A, step1(B)) \parr n(plain(N1, N2)) \circ\!\!- \forall S.(pr(A, step3(B, S)) \parr n(enc(pubk(B), N1, N2, S)))$

$p_4)$ $pr(B, step2(A, N1)) \parr n(enc(pubk(B), N1, X, Y)) \circ\!\!- pr(B, step4(A)) \parr n(plain(N1, X), enc(pubk(B), X, Y, N1))$

Figure 3: Specification of the *ffgg* protocol

i.e., the protocol is secure if processes are serialized. The protocol is as follows.

$$\begin{array}{lcl} 1. & A \rightarrow B & : \quad A \\ 2. & B \rightarrow A & : \quad N_1, N_2 \\ 3. & A \rightarrow B & : \quad \{N_1, N_2, S\}_{K_b} \% \{N_1, X, Y\}_{K_b} \\ 4. & B \rightarrow A & : \quad N_1, X, \{X, Y, N_1\}_{K_b} \end{array}$$

N_1 and N_2 stand for nonces, created by principal B and included in message 2. The $m\%m'$ notation, introduced in [Lowe, 1998], used in message 3 represents a message which has been created by the sender according to format m, but is interpreted as m' by the receiver. In this case, the intuition is that upon receiving message 3, B checks that the first component does correspond to the first of the two nonces previously created, while no check at all is performed on the second component of the message. In message 3, S stands for a secret, of the same length as a nonce, which is in possession of A. The security property one is interested to analyze is whether the secret S can be disclosed to a malicious intruder. We have implemented the *ffgg* protocol through the specification shown in Figure 3, while the intruder theory is presented in Figure 4. The specification consists of a set of protocol rules (rules p_1 through p_4 in Figure 3) and an intruder theory (rules i_1 through i_8 in Figure 4). We remind the reader that the four rules e_1, e_2, t_1 and t_2 discussed in Section 4 are *in addition* to the present rules. Protocol rules directly correspond to the informal description of the *ffgg* protocol previously presented. We have followed the conventions outlined in Section 4 to model the internal state of principals. In particular, we have a term *init* denoting the initial state of a principal, and the constructors *step*1, *step*2, *step*3 and *step*4 to model the different steps of a protocol run. At every step, each principal needs to remember the identifier of the other principal he/she is executing the protocol with. In addition, at step 2 the responder stores the first nonce created (in order to be

$i_1)$ $n(plain(X)) \multimap m(plain(X))$

$i_2)$ $n(plain(X,Y)) \multimap m(plain(X)) \parr m(plain(Y))$

$i_3)$ $n(enc(X,Y,Z,W)) \multimap m(enc(X,Y,Z,W))$

$i_4)$ $n(plain(X,Y), enc(U,V,W,Z)) \multimap m(plain(X)) \parr m(plain(Y)) \parr m(enc(U,V,W,Z))$

$i_5)$ $m(plain(X)) \multimap m(plain(X)) \parr n(plain(X))$

$i_6)$ $m(plain(X)) \parr m(plain(Y)) \multimap m(plain(X)) \parr m(plain(Y)) \parr n(plain(X,Y))$

$i_7)$ $m(enc(X,Y,Z,W)) \multimap m(enc(X,Y,Z,W)) \parr n(enc(X,Y,Z,W))$

$i_8)$ $m(plain(X)) \parr m(plain(Y)) \parr m(enc(U,V,W,Z)) \multimap m(plain(X)) \parr m(plain(Y)) \parr m(enc(U,V,W,Z)) \parr n(plain(X,Y), enc(U,V,W,Z))$

Figure 4: Intruder theory for the *ffgg* protocol

able to perform the required check, see rule p_4), and at step 3 the initiator of the protocol remembers the secret S. We have modeled the secret S using the universal quantifier, as for nonces. In this way, we can get for free the requirement that the secret initially is only known to the principal who possesses it. Finally, we have term constructors $plain(\ldots)$ and $enc(\ldots)$ (to be precise, we should say a *family* of term constructors, as we find it convenient to overload the same symbol with different *arities*) to distinguish *plain* messages from *encrypted* messages.

The intruder theory is made up of rules i_1 through i_8 in Figure 4. Let us discuss it in more detail. Rules i_1 through i_4 are *decomposition* rules, whereas rules i_5 through i_8 are *composition* rules. We have four rules for each of the two different kinds (composition and decomposition) of messages, dealing with the different formats of messages used in the *ffgg* protocol. For instance, rule i_1 deals with decomposition of plain messages with one component, whereas rule i_4 deals with decomposition of messages with two plain components and one encrypted component, and so on. Clearly, the intruder cannot furtherly decompose encrypted components, which are stored exactly as they are, whereas plain messages are decomposed into their atomic constituents. The intruder theory we have presented is an instance of the general Dolev-Yao intruder theory, in that intruder rules have been tailored to the particular form of messages used in the specific protocol under consideration, an optimization often taken by verification methods [Jacquemard et al., 2000]. We refer to Section 9 for a discussion on how this approach could be generalized. In the case of *ffgg* protocol, we have not given the intruder

any capability to decrypt messages. This hypothesis can be relaxed (as we did for the analysis of the Needham-Schroeder protocol, see Section 8.2). The present specification is sufficient for our purposes.

After presenting the encoding of protocols in linear logic and the *ffgg* protocol as an example, in the following section we will discuss the relationships between derivations in the $\mathrm{LO}_\forall$ theory and MSR [Cervesato et al., 1999], a consolidated interleaving computational model for cryptographic protocols.

6. Adequacy of $\mathrm{LO}_\forall$-based Protocol Specifications

To prove the adequacy of our encoding of authentication protocols, we take as reference model the MSR language proposed in [Cervesato et al., 1999]. This language is based on multiset rewriting over first order atomic formulas and has been used as reference model in several papers in the literature on security protocols.

The language MSR is strictly related to a fragment of linear logic which turns out to be *dual* with respect to ours. Specifically, an MSR rule can be interpreted as a linear logic formula defined as $A \multimap \exists x.B$, meaning that A evolves into B by creating a new name for x. A and B are multiplicative disjunctions of atomic formulas. In LO with universal quantification the same effect is obtained via the clause $A \circ\!\!- \forall x.B$. In fact, in the goal driven proof system of LO a computation step is obtaining by resolution (i.e., by reducing the conclusion of a clause to its premise). In the framework of [Cervesato et al., 1999] the rules are applied by rewriting the premise into the conclusion. It is easy to see that the effect of quantification is the same in the two cases. Let us illustrate this idea with the help of an example. Let Σ be a signature with two constant symbols a and b, one function symbol f and two predicate symbols p, q. Let $\mathcal{V}$ be a denumerable set of variables and $w, x, y \in \mathcal{V}$. Let P consists of the LO formula

$$\forall x, y, z.p(x) \parr q(f(y)) \circ\!\!- p(f(x)) \parr q(y) \parr q(f(x))$$

and $G = \{p(a) \parr p(b) \parr q(f(b))\}$. Figure 5 shows one possible sequence of applications of LO proof rules that start from the sequent $P \vdash_\Sigma G$. It is important to note that every *bc* rule basically rewrites the head of a ground instance of a rule in P (whose atoms are underlined in the previous figure) into its body. This property allows us to represent computations via $\mathrm{LO}_\forall$ derivations, i.e., sequences of multiset rewriting steps defined over (ground) atomic formulas. In the rest of this section we will formalize the connection between MSR and $\mathrm{LO}_\forall$.

$$
\begin{array}{c}
\vdots \\
P \vdash_\Sigma p(a), q(b), p(f(f(b))), q(b), q(f(f(b))) \\
\vdots \quad \parr_r \\
\dfrac{P \vdash_\Sigma p(a), q(b), p(f(f(b))) \parr q(b) \parr q(f(f(b)))}{P \vdash_\Sigma p(a), \underline{p(f(b))}, q(b), \underline{q(f(b))}}\ bc \\
\vdots \quad \parr_r \\
\dfrac{P \vdash_\Sigma p(a), p(f(b)) \parr q(b) \parr q(f(b))}{P \vdash_\Sigma p(a), \underline{p(b), q(f(b))}}\ bc \\
\vdots \quad \parr_r \\
P \vdash_\Sigma p(a) \parr p(b) \parr q(f(b))
\end{array}
$$

Figure 5: A fragment of $\mathrm{LO}_\forall$ proof

6.1. MSR

An MSR specification consists of a set of multiset rewriting rules over first order atomic formulas with quantification on the right-hand side, namely formulas like

$$A_1, \ldots, A_n \ \rightarrow \exists x_1 \ldots . \exists x_k .\ B_1, \ldots, B_m$$

where all free variables are considered as universally quantified. An MSR configuration $\mathcal{M}$ is a multiset of *ground* atomic formulas. The operational semantics of an MSR specification S is defined via the following one-step rewriting relation defined over pairs $\langle \mathcal{M}, \Sigma \rangle$ consisting of an MSR configuration $\mathcal{M}$ and of a *signature* Σ containing the function and constant symbols occurring in S and $\mathcal{M}$.

Let $\mathcal{M}_1$ be an MSR configuration. Suppose that there exists a ground instance of an MSR rule $\mathcal{N}_1 \ \rightarrow \exists x_1 \ldots . \exists x_k .\ \mathcal{N}_2$ in S such that $\mathcal{M}_1 = \mathcal{N}_1 + Q$ for some configuration Q (where '+' denotes multiset union). Then, $\langle \mathcal{M}_1, \Sigma \rangle \Rightarrow \langle \mathcal{N}_2[c_1/x_1, \ldots, c_k/x_k] + Q, \Sigma' \rangle$ where $c_1, \ldots, c_k \notin \Sigma$, and $\Sigma' = \Sigma \cup \{c_1, \ldots, c_k\}$. In the following we will use $\stackrel{*}{\Rightarrow}$ to denote the reflexive and transitive closure of $\Rightarrow$. An MSR configuration $\mathcal{M}$ is reachable from $\mathcal{M}_0$ if $\langle \mathcal{M}_0, \Sigma_0 \rangle \stackrel{*}{\Rightarrow} \langle \mathcal{M}_1, \Sigma_1 \rangle$ for some Σ_1, where Σ_0 is the signature associated with S and $\mathcal{M}_0$.

We now define an embedding of MSR into $\mathrm{LO}_\forall$. Given an MSR rule R

$$A_1, \ldots, A_n \ \rightarrow \exists x_1 \ldots . \exists x_k .\ B_1, \ldots, B_m$$

we define $lo(R)$ as the (universally quantified) $\mathrm{LO}_\forall$ clause

$$\forall(A_1 \parr \ldots \parr A_n \ \circ\!\!- \forall x_1 \ldots . \forall x_k .\ B_1 \parr \ldots \parr B_m)$$

Given an MSR specification S consisting of the MSR rules $R_1, \ldots, R_q$, we naturally extend the mapping lo to S as follows: $lo(S) = \{lo(R_1), \ldots, lo(R_q)\}$. Let us now use the notation $Seq \rhd Seq'$ to denote a partial $\mathrm{LO}_\forall$-derivation of sequent Seq' from sequent Seq having the form of a single branch. Then, the following results holds.

PROPOSITION 6.1 (ADEQUACY OF THE $\mathrm{LO}_\forall$ ENCODING OF MSR): *Let S be an MSR specification, $\mathcal{M}_0$ and $\mathcal{M}_1$ be two MSR configurations. If $\langle \mathcal{M}_0, \Sigma_0 \rangle \stackrel{*}{\Rightarrow} \langle \mathcal{M}_1, \Sigma_1 \rangle$ in S, then $lo(S) \vdash_{\Sigma_1} \mathcal{M}_1 \rhd lo(S) \vdash_{\Sigma_0} \mathcal{M}_0$.*

Proof: The proof is by induction on the number of rewriting steps K needed to go from $\langle \mathcal{M}_0, \Sigma_0 \rangle$ to $\langle \mathcal{M}_1, \Sigma_1 \rangle$.

- The thesis immediately holds for $K = 0$ (i.e., $\mathcal{M}_0 = \mathcal{M}_1$).
- Now, suppose $K > 0$, i.e., $\langle \mathcal{M}_0, \Sigma_0 \rangle \stackrel{*}{\Rightarrow} \langle \mathcal{M}, \Sigma \rangle \Rightarrow \langle \mathcal{M}_1, \Sigma_1 \rangle$.
  Then, by inductive hypothesis we assume $lo(S) \vdash_\Sigma \mathcal{M} \rhd lo(S) \vdash_{\Sigma_0} \mathcal{M}_0$.
  Furthermore, since $\langle \mathcal{M}, \Sigma \rangle \Rightarrow \langle \mathcal{M}_1, \Sigma_1 \rangle$, there exists a ground instance of an MSR rule $\mathcal{N}_1 \ \rightarrow \exists x_1 \ldots . \exists x_k .\ \mathcal{N}_2$ in S such that $\mathcal{M} = \mathcal{N}_1 + Q$ for some configuration Q, $\mathcal{M}_1 = \mathcal{N}_2[c_1/x_1, \ldots, c_k/x_k] + Q$ and $\Sigma_1 = \Sigma \cup \{c_1, \ldots, c_k\}$ for $c_1, \ldots, c_k \notin \Sigma$.
  By definition of the mapping lo, it is easy to check that $\overline{\mathcal{N}_1} \ \rightarrow \forall x_1 \ldots . \forall x_k . \overline{\mathcal{N}_2}$ is a ground instance of $lo(R)$, where $\overline{A_1, \ldots, A_n} = A_1 \parr \ldots \parr A_n$.
  Since $\widehat{\overline{\mathcal{N}}} = \mathcal{N}$, we can apply an instance of the bc rule to sequent $Seq_1 = lo(S) \vdash_\Sigma \mathcal{M}$ obtaining the sequent $Seq_2 = lo(S) \vdash_\Sigma \forall x_1 \ldots . \forall x_k . \overline{\mathcal{N}_2}, Q$. By applying k times the $\forall_r$ rule, we get $Seq_3 = lo(S) \vdash_{\Sigma_1} \overline{\mathcal{N}_2[c_1/x_1, \ldots, c_k/x_k]}, Q$. Finally, assuming that $\overline{\mathcal{N}_2[c_1/x_1, \ldots, c_k/x_k]} = B_1 \parr \ldots \parr B_m$, by applying m times the $\parr_r$ rule we obtain $Seq = lo(S) \vdash_{\Sigma_1} \mathcal{N}_2[c_1/x_1, \ldots, c_k/x_k], Q$. Since $\mathcal{M}_1 = \mathcal{N}_2[c_1/x_1, \ldots, c_k/x_k], Q$ we have proved the thesis.

□

Let us now call *simple* an $\mathrm{LO}_\forall$ clause R having the following form:

$$\forall(A_1 \parr \ldots \parr A_n \ \circ\!\!- \forall x_1 \ldots . \forall x_k .\ B_1 \parr \ldots \parr B_m)$$

The corresponding MSR rule $msr(R)$ is defined as

$$A_1, \ldots, A_n \ \rightarrow \exists x_1 \ldots . \exists x_k .\ B_1, \ldots, B_m$$

A simple $\mathrm{LO}_\forall$ specification consists of simple $\mathrm{LO}_\forall$ clauses only. The above mapping extends from rules to specifications in a straightforward manner. Then, the following proposition holds.

PROPOSITION 6.2 (ADEQUACY OF THE MSR ENCODING OF $\mathrm{LO}_\forall$): *Let T be a simple $LO_\forall$ specification, $\mathcal{M}_0$ and $\mathcal{M}_1$ be two MSR configurations. If $T \vdash_{\Sigma_1} \mathcal{M}_1 \rhd T \vdash_{\Sigma_0} \mathcal{M}_0$ then $\langle \mathcal{M}_0, \Sigma_0 \rangle \stackrel{*}{\Rightarrow} \langle \mathcal{M}_1, \Sigma_1 \rangle$ in $msr(T)$.*

Proof: We first note that partial derivations for simple $\mathrm{LO}_\forall$ theories have only a single proof branch (clauses have no additive conjunction in the body). The proof is then by induction on the length of the single branch in the derivation $T \vdash_{\Sigma_1} \mathcal{M}_1 \rhd T \vdash_{\Sigma_0} \mathcal{M}_0$. The proof is carried out quite similarly to the one of the previous proposition, the only interesting case being the inductive step. Since proofs in $\mathrm{LO}_\forall$ are goal driven we are forced to apply in sequence an instance of the *bc* rule with respect to a clause R, a finite number of times the $\forall_r$ rule, and, finally, a finite number of times the $\parr_r$ rule until the current goal consists of atomic formulas only. As shown in the previous proposition, this proof scheme precisely mimics an application of the MSR rule $msr(R)$. □

Thanks to the previous property and from the observation that we have adopted the same style for the specification of protocols proposed in [Cervesato et al., 1999], we can state the following result.

PROPOSITION 6.3 (ADEQUACY OF THE ENCODING): *Every interleaving execution of a protocol S specified in MSR (with or without intruder) is represented by a partial $LO_\forall$ derivation in $lo(S)$. Vice versa, every partial $LO_\forall$ derivation of a protocol T specified in simple $LO_\forall$ represents an interleaving execution of the protocol $msr(T)$.*

7. Verification of Security Properties

Several practical examples of *safety properties* present the following interesting feature: their negation can be represented by means of the upward closure of a collection of *minimal violations*, e.g., as for mutual exclusion properties of communication protocols [Abdulla et al., 2000]. Thanks to this property, it often becomes

$$u)\ \ pr(alice, step3(bob, S))\ \ ⅋\ \ pr(bob, step4(alice))\ \ ⅋\ \ m(plain(S))\ \ \circ\!\!-\ \ \top$$

Figure 6: A logical representation of an infinite set of unsafe configuration for the *ffgg* protocol

possible to *finitely* represent infinite collections of unsafe configurations. Symbolic procedures can be applied then in order to saturate the set of predecessor states (by iteratively applying a transition relation *backwards*[†]). Using this method and assuming that a fixpoint is eventually reached, it is possible then to establish which initial states lead to violations of the property.

This observation can be applied in our setting in order to specify interesting *security properties*. As an example, in the *ffgg* protocol we consider a configuration *unsafe* if there exist *at least* two honest principals, say *alice* and *bob*, who have run the protocol to completion (i.e., they have completed, respectively, step 4 and step 3) and the secret S has been disclosed to the intruder (i.e., it is eventually stored in the intruder's internal memory). In our setting a configuration is represented as a multiset of atomic formulas. In order to symbolically represent *all possible configurations* in which a violation might occur, we can use then the $\mathrm{LO}_\forall$ clause in Figure 6. Every *top-down* derivation leading from an initial goal (state) to an instance of the axiom $\top_r$ obtained by applying the rule u will represent a possible *attack* to the protocol security. It is important to note that, thanks to the admissibility of the weakening rule (see Section 3.1), the previous $\mathrm{LO}_\forall$ rule can be used to represent unsafe configurations for *any number of principals* involved in sessions running in parallel with the session carried over by *alice* and *bob*. Exploring all possible *top-down* derivations however corresponds to an exhaustive search of the state space, and it would force us to fix a given initial configuration.

A possible way to circumvent the problems due to forward exploration (top-down provability) is to use an alternative evaluation strategy for $\mathrm{LO}_\forall$ specifications. The approach we propose in this paper is based on the bottom-up evaluation strategy of $\mathrm{LO}_\forall$ program we introduced in [Bozzano et al., 2002b]. The idea is as follows. Starting from a set of *facts* of the form $H \circ\!\!-\ \top$, we compute all possible logical consequences with respect to a given $\mathrm{LO}_\forall$ program. Computationally, this evaluation strategy can be viewed as a *backward* exploration of the state space of the system specified by the $\mathrm{LO}_\forall$ theory. As explained before, the set of facts

[†]Given that sets of unsafe configurations are encoded via logical axioms, computing the *backward* reachability set of a transition relation amounts to evaluating the corresponding logic program *bottom-up*

can be used to symbolically express infinite sets of violations of a given safety property. The bottom-up evaluation amounts to a fixpoint computation defined over a transformation of sets of facts into sets of facts (as the classical T_P operator used in the fixpoint semantics of logic programs).

For instance, in the context of security protocols by evaluating *bottom-up* the $\mathrm{LO}_\forall$ program obtained by merging the protocol and intruder theory with the symbolic representation of unsafe states like the clause u, we obtain the same effect using *backward reachability* for a complex specification (with quantification and so on) carried over in a completely open environment. Furthermore, if a fixpoint is reached (this is not guaranteed in general) we can derive *conditions* on the initial states under which unsafe configurations will not be reached.

7.1. Bottom-up Evaluation for $\mathrm{LO}_\forall$ Specifications

In this section we introduce the basic ideas underlying the *bottom-up* evaluation scheme of $\mathrm{LO}_\forall$ programs. For more details, the reader may refer to [Bozzano et al., 2002b, Bozzano, 2002]. As mentioned in the previous section, we are interested in observing the set of disjunctive atomic goals that are provable in a given program P. By admissibility of weakening, we observe that if $\mathcal{A} \in \mathcal{O}(P)$ then $\mathcal{A}+\mathcal{C} \in \mathcal{O}(P)$ (where '+' denotes multiset union) for any multiset $\mathcal{C}$ (of atomic formulas). In other words, $\mathcal{O}(P)$ is *upward closed* w.r.t. multiset inclusion. We can exploit this property in order to "finitely" represent sets of provable goals by using the following idea. We will consider interpretations consisting of multisets of *non-ground* formulas and we will lift their denotation to the upward closure of their *ground* instances. Formally, we give the following definition. Given an $\mathrm{LO}_\forall$ program P, let $HB(P)$ be the (non-ground) *Herbrand base* of P, i.e., the *set of multisets* built over the signature Σ_P associated to the program P. Furthermore, let Sig_P be the set of all extensions of Σ_P with new constant symbols.

Definition (Interpretation): Given an $\mathrm{LO}_\forall$ program P, an interpretation I is any subset of $HB(P)$. The denotation of an interpretation I, denoted $[\![I]\!]$, is the family of *ground* interpretations $\{[\![I]\!]_\Sigma\}_{\Sigma \in Sig_P}$ defined as follows:

$$[\![I]\!]_\Sigma = Up_\Sigma(Inst_\Sigma(I)),$$

where $Inst_\Sigma$ and Up_Σ are defined by $Inst_\Sigma(I) = \{\mathcal{A}\theta \mid \mathcal{A} \in I\}$, $Up_\Sigma(I) = \{\mathcal{A}+\mathcal{C} \mid \mathcal{A} \in I\}$, θ being a substitution over Σ and $\mathcal{C}$ a multiset over Σ.

Notice that, in the definition of $[\![\cdot]\!]_\Sigma$ the operations of instantiation and upward-closure are performed for every possible signature $\Sigma \in Sig_P$, i.e., for every possible

extension of the program signature Σ_P. Furthermore, we assume the substitution θ and the multiset $\mathcal{C}$ to be defined over Σ (i.e., they may refer/contain only terms over Σ). We say that an interpretation I is *ground* whenever all multisets $\mathcal{A} \in I$ consist of ground atomic formulas.

Given an $\mathrm{LO}_\forall$ goal G, we need to define a notion of satisfiability w.r.t. to our definition of interpretation. For this purpose, we introduce the following satisfiability judgment

$$I \Vdash_\Sigma \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta,$$

where I is an interpretation, Δ is a multiset of goal formulas (a *context*), $\mathcal{C}$ is an output multiset of atomic formulas, and θ is an output substitution. We give the following definition (for simplicity, we present only the formal definition of the judgment for goals without conjunction, see [Bozzano et al., 2002b] for the complete definition). Below, $\mathcal{A} \setminus \mathcal{B}$ denotes the multiset difference between $\mathcal{A}$ and $\mathcal{B}$, $|\mathcal{A}|$ denotes the *cardinality* of $\mathcal{A}$, $FV(\mathcal{A}, \mathcal{C})$ denotes the set of free variables in $\mathcal{A} + \mathcal{C}$, and $\preccurlyeq$ denotes *multiset inclusion*.

Definition (Satisfiability Judgment): Let P be an $\mathrm{LO}_\forall$ program, I an interpretation, and $\Sigma \in Sig_P$. The satisfiability judgment $\Vdash_\Sigma$ is defined as follows:

axiom :	$I \Vdash_\Sigma \top, \Delta \blacktriangleright \epsilon \blacktriangleright nil$;
anti :	$I \Vdash_\Sigma \bot, \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$, if $I \Vdash_\Sigma \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$;
par :	$I \Vdash_\Sigma G_1 \parr G_2, \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$, if $I \Vdash_\Sigma G_1, G_2, \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$;
forall :	$I \Vdash_\Sigma \forall x.G, \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$, if $I \Vdash_{\Sigma,c} G[c/x], \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$, with $c \notin \Sigma$;
atomic multiset :	$I \Vdash_\Sigma \mathcal{A} \blacktriangleright \mathcal{C} \blacktriangleright \theta$, if there exist $\mathcal{B} \in I$ (variant), $\mathcal{B}' \preccurlyeq \mathcal{B}$, $\mathcal{A}' \preccurlyeq \mathcal{A}$, $\lvert\mathcal{B}'\rvert = \lvert\mathcal{A}'\rvert$, $\mathcal{C} = \mathcal{B} \setminus \mathcal{B}'$, and $\theta = m.g.u.(\mathcal{B}', \mathcal{A}')_{\lvert FV(\mathcal{A},\mathcal{C})}$

The judgment is used to compute the set of *resources* $\mathcal{C}$ and the corresponding *variable bindings* that are needed for Δ to be provable in I (in the previous definition, I is intended to denote its upward closure, according to Definition 7.1). Intuitively, if $I \Vdash_{\Sigma_P} \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$ holds then the sequent $P, P' \vdash_{\Sigma_P} \Delta\theta\gamma, \mathcal{C}\theta\gamma, \gamma$ being a grounding substitution, is provable by augmenting P with the program P' consisting of clauses like $\mathcal{A} \multimap \top$ for any $\mathcal{A} \in [\![I]\!]_{\Sigma_P}$. Technically, the idea behind the definition is that the output multiset $\mathcal{C}$ and the output substitution θ are *minimal* (in a sense to be clarified) so that they can be computed effectively given a program P, an interpretation I, and a signature Σ. The output substitution θ is needed in order to deal with clause instantiation, and its minimality is ensured by using most general unifiers in the definition. In general, two multisets may have more than one (not necessarily equivalent) most general unifier (see [Bozzano

et al., 2002b]). By using the notation $m.g.u.(\mathcal{B}', \mathcal{A}')$, we mean any unifier which is *non-deterministically* picked from the set of most general unifiers of $\mathcal{B}'$ and $\mathcal{A}'$.

Remark: The notation $I \Vdash_\Sigma \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$ requires that Δ, $\mathcal{C}$, and θ are defined over Σ. As a consequence, the newly introduced constant c in the $\forall$-case of the $\Vdash_\Sigma$ definition below *cannot be exported* through the output parameters $\mathcal{C}$ or θ. This way, universal quantification is always resolved *locally*. For simplicity, in Definition 7.2 we present only the formal definition of the judgment for goals without conjunction (see [Bozzano et al., 2002b] for the complete definition).

We are now ready to define the symbolic fixpoint operator S_P working on our notion of interpretation. Below, $Vrn(P)$ denotes the set of clauses that are variant (i.e., renamed with fresh variables) of clauses in P. Furthermore, we remind that, given $H = A_1 \parr \ldots \parr A_k$, $\widehat{H}$ is the multiset $A_1, \ldots, A_k$.

Definition: Given an $\mathrm{LO}_\forall$ program P and an interpretation I, the symbolic fixpoint operator S_P is defined as follows:

$$S_P(I) = \{\ (\widehat{H} + \mathcal{C})\,\theta \mid\ \ (H \multimap G) \in Vrn(P), I \Vdash_{\Sigma_P} G \blacktriangleright \mathcal{C} \blacktriangleright \theta\ \}.$$

EXAMPLE 7.4: *Let Σ_P be a signature with a function symbol f and predicate symbols p, q, r, s. Let I be the interpretation consisting of the multiset $\{p(x), q(x)\}$ (for simplicity, hereafter we omit brackets in multiset notation), and P the program*

1. $r(w) \multimap q(f(w))$
2. $s(z) \multimap \forall x.p(f(x))$

Let's consider (a renaming of) the body of the first clause, $q(f(w'))$, and (a renaming of) the element in I, $p(x'), q(x')$. Using the atomic *clause for the $\Vdash_{\Sigma_P}$ judgment, with $\mathcal{A} = \mathcal{A}' = q(f(w'))$, $\mathcal{B} = p(x'), q(x')$, $\mathcal{B}' = q(x')$, we get*

$$I \Vdash_{\Sigma_P} q(f(w')) \blacktriangleright p(x') \blacktriangleright [x' \mapsto f(w')].$$

Thus, the multiset $p(f(w')), r(w') \in S_P(I)$ (in fact, any of its instances is provable in P enriched with $p(x) \parr q(x) \multimap \top$). This is not the only possible result of applying S_P. In fact we can apply the first clause to I by choosing $\mathcal{A}' = \mathcal{B}' = \epsilon$ in the atomic *case of $\Vdash_{\Sigma_P}$. Thus, the multiset $\mathcal{A} = p(x'), q(x'), r(w)$ belongs to $\in S_P(I)$, too. Notice that the latter multiset denotes* redundant *information w.r.t. the denotations of $\mathcal{B} = p(x'), q(x')$. In fact $[\![\{\mathcal{A}\}]\!] \subseteq [\![\{\mathcal{B}\}]\!]$.*

Let's consider now (a renaming of) the body of the second clause, $\forall x.p(f(x))$, and another renaming of the single element in I, $p(x''), q(x'')$. From the $\forall$-case of $\Vdash_{\Sigma_P}$

definition, $I \Vdash_{\Sigma_P} \forall x.p(f(x)) \blacktriangleright \mathcal{C} \blacktriangleright \theta$ *if* $I \Vdash_{\Sigma_P,c} p(f(c)) \blacktriangleright \mathcal{C} \blacktriangleright \theta$*, with* $c \notin \Sigma_P$*.*

Now, we can apply the atomic *clause for* $\Vdash_{\Sigma_P,c}$*. Unfortunately, we can't choose* $\mathcal{A}'$ *to be* $p(f(c))$ *and* $\mathcal{B}'$ *to be* $p(x'')$*. In fact, by unifying* $p(f(c))$ *with* $p(x'')$*, we should get the substitution* $\theta = [x'' \mapsto f(c)]$ *and the output multiset* $q(x'')$ *(notice that* x'' *is a free variable in the output multiset) and this is not allowed because the substitution* θ *must be defined on* Σ_P*, in order for* $I \Vdash_{\Sigma_P} \forall x.p(f(x)) \blacktriangleright \mathcal{C} \blacktriangleright \theta$ *to be meaningful. It turns out that the only way to use the second clause for* $\Vdash_{\Sigma_P,c}$ *is to choose* $\mathcal{A}' = \mathcal{B}' = \epsilon$*. In fact, notice that goals of the form* $p(c_1), r(c_2)$ *are not provable in* P *enriched with the axiom* $p(x) \parr q(x) \multimap \top$*.*

Notice that the S_P operator is defined using the judgment $\Vdash_{\Sigma_P}$. This corresponds to the idea that we are interested in observing only provable goals that are *visible* outside the scope of programs with universal quantification. The constants that are introduced during a derivation, in fact, cannot be exported outside the scope of the corresponding subderivation. The operator S_P is monotonic and continuous over the set of interpretations ordered with respect to inclusion of their denotations [Bozzano et al., 2002b]. The fixpoint semantics $\mathcal{F}(P)$ of an $\mathrm{LO}_\forall$ program P is defined then as the *least fixpoint* of the operator S_P. Furthermore, the following property (proved in [Bozzano et al., 2002b]) holds.

Theorem 7.1 (Soundness and Completeness [Bozzano et al., 2002b]): *Let* P *be an* $LO_\forall$ *program. Then,* $O(P) = [\![\mathcal{F}(P)]\!]_{\Sigma_P}$*.*

Finally, we can define an effective *subsumption* test between multisets of non-ground goals, in accordance with the notion of rich denotations of Definition 7.1. Intuitively, $\mathcal{B}$ *entails* $\mathcal{A}$ if there exists $\mathcal{A}' \preccurlyeq \mathcal{A}$ such that $\mathcal{B}$ is *more general* than a permutation of $\mathcal{A}'$ (where multisets are viewed as lists of terms). Formally, we have the following proposition.

Proposition 7.1 ([Bozzano et al., 2002b]): *Given two interpretations* I *and* J*,* $[\![I]\!] \subseteq [\![J]\!]$ *if and only if for every* $\mathcal{A} \in I$*, there exist* $\mathcal{B} \in J$*, a substitution* θ *and a fact* $\mathcal{C}$ *(defined over* Σ_P*) s.t.* $\mathcal{A} = \mathcal{B}\theta + \mathcal{C}$*.*

This effective test can be used as a symbolic termination test for the least fixpoint computation built on top of the operator S_P. The resulting machinery represents then the *core* of our bottom-up evaluation procedure for $\mathrm{LO}_\forall$ programs. Sufficient conditions for termination are discussed in detail in [Bozzano et al., 2002b].

7.2. Verification as Deduction

On the basis of the notions introduced in the previous sections, we can establish the following connection between *bottom-up* evaluation (i.e., the semantics $\mathcal{F}(P)$ defined in Section 7.1) of an $\mathrm{LO}_\forall$ specification of an authentication protocol and its corresponding security properties. Let $Init$ be a collection of multisets of ground atomic formulas (the initial states of a protocol), T_p be the $\mathrm{LO}_\forall$ theory encoding a protocol $\mathcal{P}$, T_i the $\mathrm{LO}_\forall$ intruder theory, and let U be a collection of $\mathrm{LO}_\forall$ clauses $\mathcal{A}_1 \multimap \top, \ldots, \mathcal{A}_k \multimap \top$ (corresponding to the minimal violations of a secrecy property $\mathcal{S}$). Furthermore, let $T = T_p \cup T_i \cup U$. Then, we have the following properties.

PROPOSITION 7.2 (ENSURING SECRECY): *The protocol $\mathcal{P}$ is secure w.r.t. the intruder with capabilities $\mathcal{T}_i$, initial configurations $Init$, and the secrecy property $\mathcal{S}$, and an interleaving computational model for the agents behaviour, if and only if $Init \cap [\![\mathcal{F}(T)]\!] = \emptyset$.*

Proof: This result follows from Proposition 6.3 and Theorem 7.1. □

We remark that an *if and only if* condition holds in the previous proposition, i.e., the bottom-up evaluation algorithm is correct and complete. As a corollary, we get the following property (*only if* direction in the previous proposition), useful for debugging purposes.

COROLLARY 7.1 (PROVING INSECURITY): *If there exists $\mathcal{A}$ such that $\mathcal{A} \in Init \cap [\![\mathcal{F}(T)]\!]$, then in the interleaving computational model for the agents behaviour there exists an* attack *that leads from the initial configuration $\mathcal{A}$ to an unsafe configuration $\mathcal{B} \in [\![U]\!]$.*

8. Practical Results

This section discusses the specification and analysis of some authentication protocols, taken from the literature on security. Specifically, in addition to the *ffgg* protocol (which has been presented in Section 5), we will discuss the Needham-Schroeder protocol (see Section 2.2) in Section 8.2, a corrected version of the same protocol in Section 8.3, and the Otway-Rees protocol [Otway and Rees, 1987] in Section 8.4. We will follow the guidelines illustrated in Section 4. Experimental results for all the examples presented in this paper, obtained using the tool presented in [Bozzano, 2002], are summarized in Table 14 (see Section 8.5).

8.1. Analysis of Millen's *ffgg* protocol

Running our bottom-up evaluation algorithm on the *ffgg* specification (see Section 5), we automatically find a violation to the security property of Figure 6.

As in traditional model checking, counterexamples traces can be automatically generated whenever a violation is found. In particular, the trace corresponding to the above attack is shown in Figure 7 (we only post-processed the output of our verification tool to show the trace in a more human-readable form). The trace in Figure 7 corresponds to an $\mathrm{LO}_\forall$ derivation which leads from an initial state to a state violating the security property of Figure 6. The attack is exactly the *parallel session* one described in [Millen, 1999]. This attack is also an example of a *type flaw* attack, in that it relies on the secret S be passed as a nonce (under the hypothesis that the lengths of the respective fields are the same). In order to let the reader better understand the connection between *bottom-up* evaluation and the *top-down* derivation shown in Figure 7, we present below some of the steps performed by the bottom-up evaluation algorithm. In the following we follow the same syntactical notations as in Figure 7. Bottom-up evaluation starts from axiom u, i.e., we assert the following provable multiset:

$$m_1) \quad al^3_{bob,S}, bob^4_{al}, m(S),$$

where S is a *free* variable. Different clauses are applicable at this point. Among them, decomposition rule i_4. We can apply a variant of i_4, let it be

$$n(X', Y', \{V', W', Z'\}_{K_U}) \multimap m(X') \parr m(Y') \parr m(\{V', W', Z'\}_{K_U})$$

to m_1, in the following manner: unify S with Y' (hence unifying $m(S)$ in the multiset with $m(Y')$ in the clause body) and consider the other two atoms in the body (i.e., $m(X')$ and $m(\{V', W', Z'\}_{K_U})$) as being implicitly contained in m_1 (remember that interpretations are to be considered *upward-closed*). By applying the resulting clause *backwards* (i.e., the body is replaced by the head) we get

$$m_2) \quad al^3_{bob,S}, bob^4_{al}, n(X', S, \{V', W', Z'\}_{K_U}).$$

Multiset m_2 is accumulated into the current set of provable goals (other multisets can be obtained by applying the remaining program clauses). Now, consider the application of a variant of protocol rule p_4, let it be

$$(B'')^2_{A'',N1''} \parr n(\{N1'', X'', Y''\}_{K_{B''}}) \multimap (B'')^4_{A''} \parr n(N1'', X'', \{X'', Y'', N1''\}_{K_{B''}})$$

to m_2, in the following way: unify $N1''$ with X' and Z', X'' with S and V', Y'' with W', and B'' with U (thus unifying the atoms $n(N1'', X'', \{X'', Y'', N1''\}_{K_{B''}})$ and

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash_{\Sigma_3} \top, bob^4_{al}, m(al,n1,n2,n3,n4,\{n3,s,n1\}_{K_{bob}},\{s,n1,n3\}_{K_{bob}})}\top_r}{P \vdash_{\Sigma_3} al^3_{bob,s}, bob^4_{al}, bob^4_{al}, m(al,n1,n2,n3,n4,s,\{n3,s,n1\}_{K_{bob}},\{s,n1,n3\}_{K_{bob}})}bc^{(u)}}{P \vdash_{\Sigma_3} al^3_{bob,s}, bob^4_{al}, bob^4_{al}, m(al,n1,n2,n3,n4,\{n3,s,n1\}_{K_{bob}}), n(n3,s,\{s,n1,n3\}_{K_{bob}})}bc^{(i_4)}}{P \vdash_{\Sigma_3} al^3_{bob,s}, bob^4_{al}, bob^2_{al,n3}, m(al,n1,n2,n3,n4,\{n3,s,n1\}_{K_{bob}}), n(\{n3,s,n1\}_{K_{bob}})}bc^{(p_4)}}{P \vdash_{\Sigma_3} al^3_{bob,s}, bob^4_{al}, bob^2_{al,n3}, m(al,n1,n2,n3,n4,\{n3,s,n1\}_{K_{bob}})}bc^{(i_7)}}{P \vdash_{\Sigma_3} al^3_{bob,s}, bob^4_{al}, bob^2_{al,n3}, m(al,n1,n2,n3,n4), n(n1,n3,\{n3,s,n1\}_{K_{bob}})}bc^{(i_4)}}{P \vdash_{\Sigma_3} al^3_{bob,s}, bob^2_{al,n1}, bob^2_{al,n3}, m(al,n1,n2,n3,n4), n(\{n1,n3,s\}_{K_{bob}})}bc^{(p_4)}}{P \vdash_{\Sigma_2} al^1_{bob}, bob^2_{al,n1}, bob^2_{al,n3}, m(al,n1,n2,n3,n4), n(n1,n3)}bc^{(p_3)}}{P \vdash_{\Sigma_2} al^1_{bob}, bob^2_{al,n1}, bob^2_{al,n3}, m(al,n1,n2,n3,n4)}bc^{(i_6)}}{P \vdash_{\Sigma_2} al^1_{bob}, bob^2_{al,n1}, bob^2_{al,n3}, m(al,n1,n2), n(n3,n4)}bc^{(i_2)}}{P \vdash_{\Sigma_1} al^1_{bob}, bob^2_{al,n1}, bob^{init}, m(al,n1,n2), n(al)}bc^{(p_2)}}{P \vdash_{\Sigma_1} al^1_{bob}, bob^2_{al,n1}, bob^{init}, m(al,n1,n2)}bc^{(i_5)}}{P \vdash_{\Sigma_1} al^1_{bob}, bob^2_{al,n1}, bob^{init}, m(al), n(n1,n2)}bc^{(i_2)}}{P \vdash_{\Sigma} al^1_{bob}, bob^{init}, bob^{init}, m(al), n(al)}bc^{(p_2)}}{P \vdash_{\Sigma} al^{init}, bob^{init}, bob^{init}, m(al)}bc^{(p_1)}}{P \vdash_{\Sigma} al^{init}, bob^{init}, bob^{init}}bc^{(t_1)}}{P \vdash_{\Sigma} al^{init}, bob^{init}}bc^{(e_2)}$$

Figure 7: A parallel session attack to the *ffgg* protocol: al=$alice$; p^i_d is principal p after execution of step i with internal data d; p^{init} is principal p in its initial state; we have omitted the *plain* term constructor for plain messages; we have noted encrypted messages using the usual protocol notation; $\mathcal{M}(x,y,\ldots)$ stands for the multiset $m(x), m(y), \ldots$; finally, $\Sigma_1 = \Sigma, n1, n2$, $\Sigma_2 = \Sigma, n1, n2, n3, n4$, and $\Sigma_3 = \Sigma, n1, n2, n3, n4, s$

$n(X', S, \{V', W', Z'\}_{K_U})$). Furthermore, assume the atom $(B'')^4_{A''}$ to be implicitly

contained in m_2. We get the multiset

$$m_3) \quad al^3_{bob,S}, bob^4_{al}, (B'')^2_{A'',N1''}, n(\{N1'', S, Y''\}_{K_{B''}})$$

which is in turn accumulated as a provable goal. We invite the reader to observe the correspondence between the *bottom-up* construction we are sketching and the *top-down* construction illustrated in Figure 7. Notice that the sequence of rules we are applying is the same but in the reversed order, i.e., axiom u, then rules i_4 and p_4, and so on (clearly, we are illustrating only one among the possible bottom-up derivations). Furthermore, every atom that we described as *implicitly contained* in the current multiset corresponds to one of the atoms in the top sequent of Fig. 7. In other words, the bottom-up computation starts from a multiset representing the *minimal* violations of the security property under consideration (i.e., axiom u), whereas any additional atom that turns out to be involved in the proof (see top-sequent in Fig. 7) is (implicitly) added, so to say, in a *lazy* manner as the bottom-up construction proceeds. Variable bindings can also be (implicitly) enforced during the bottom-up construction. For instance, the atom $(B'')^4_{A''}$ (which we assumed to be implicitly contained in m_2) corresponds to the atom bob^4_{al} in the top sequent of Fig. 7. Eventually, variable B'' (which is contained, e.g., in m_3) will be unified with *bob*, and similarly, A'' will be unified with al. We conclude with an example of application of a clause involving universal quantification. Proceeding as above, eventually we get (a variant) of the multiset

$$m_6) \quad al^3_{bob,S}, bob^2_{al,N'}, bob^2_{A,N''}, n(\{N', N'', S\}_{K_{bob}}).$$

Now, we can apply a variant of protocol rule p_3 (see the corresponding inference in Fig. 7), let it be

$$(A')^1_{B'} \ \parr \ n(N1', N2') \ \circ\!\!- \ \forall S'.\ ((A')^3_{B',S'} \parr n(\{N1', N2', S'\}_{K_{B'}}))$$

to m_6, by unifying A' with al, B' with *bob*, S' with S, N' with $N1'$ and N'' with $N2''$. We get the following multiset:

$$m_7) \quad al^1_{bob}, bob^2_{al,N'}, bob^2_{A,N''}, n(N', N').$$

Notice that the bottom-up inference requires a new constant, let it be c, to be introduced in place of the universally quantified variable S' in the body of the above clause. According to rule $forall$ for the satisfiability judgment (see Definition 7.2) a static check must be performed in order to ensure that the output multiset and unifier do not contain the constant c. This check is successfully passed, thus the above inference is perfectly legal. The bottom-up construction

goes on in this way until the multiset al^{init}, bob^{init} (corresponding to the bottom sequent in Figure 7) is reached.

We conclude by mentioning that we have also performed some further experiments regarding Millen's *ffgg* protocol, which we don't discuss in detail. In particular, we wanted to ascertain the role of the two nonces N_1 and N_2 in the *ffgg* protocol. According to the informal notation for the protocol introduced at the beginning of this section, principal B only checks that the first component of message T is the nonce N_1, whereas no check is performed for the second component. We have verified that imposing the check on the second component, the *ffgg* protocol is safe w.r.t. the security property and the intruder theory we have presented, while removing all checks, as expected, introduces *serial* attacks.

We think that this example is a good illustration of the capabilities of our general framework. In fact, using the *backward* evaluation strategy championed in this paper, we are able to automatically find a parallel session attack, *without enforcing any particular search strategy* of our evaluation algorithm (i.e., the same algorithm can be used to find serial or parallel attacks). Furthermore, according to [Millen, 1999] the *ffgg* protocol can be generalized to protocols which only admit *higher-order* parallel attacks (i.e., attacks which take place only in presence of *three or more* concurrent roles for the same principal). Using the same algorithm, and the same protocol and intruder theories as before, we can automatically find such attacks, if any exists. This distinguishes our methodology from most approaches based on model-checking, which operate on a finite-state abstraction of a given protocol, and require the number of principals and roles to be fixed *in advance*.

Another advantage of using backward reasoning is related to the generation of nonces. Forward exploration needs to explicitly manage generation of fresh names. The backward application of $\mathrm{LO}_\forall$ rules allows us instead to observe only formulas defined over the signature of the original program. In fact, suppose that a rule body contains universally quantified variables that have to be matched with an interpretation computed during the bottom-up evaluation of a program. By the definition of the satisfiability judgment (see Definition 7.2) and of the S_P operator, we can restrict ourselves to a *local* top-down derivation in which we simplify the body of the clause (see rule *forall* and *par* of Definition 7.2) and then we match the resulting multiset of formulas against the current interpretation (see rule *atomic multiset* of Definition 7.2). In the end a *static check* is made in order to ensure that the *output multiset* and the *resulting unfier* do not contain the constants which have been introduced. In other words, the effect of quantification in the body of a clause is simply that of *restricting* the set of possible *predecessor* configurations.

$p_1)$ $pr(A, init) \parr pr(B, init) \circ\!\!- pr(B, init) \parr \forall NA.(pr(A, step1(NA, B)) \parr n(pubk(B), mess(NA, A)))$

$p_2)$ $pr(B, init) \parr n(pubk(B), mess(NA, A)) \circ\!\!- \forall NB.(pr(B, step2(NA, NB, A)) \parr n(pubk(A), mess(NA, NB)))$

$p_3)$ $pr(A, step1(NA, B)) \parr n(pubk(A), mess(NA, NB)) \circ\!\!- pr(A, step3(NA, NB, B)) \parr n(pubk(B), mess(NB))$

$p_4)$ $pr(B, step2(NA, NB, A)) \parr n(pubk(B), mess(NB)) \circ\!\!- pr(B, step4(NA, NB, A))$

Figure 8: Specification of the Needham-Schroeder protocol

On the contrary, using top-down evaluation, the current signature is enriched by adding a new constant every time the rule for the universal quantifier is used.

8.2. The Needham-Schroeder Protocol

In this section we analyze the Needham-Schroeder public-key authentication protocol [Needham and Schroeder, 1978], previously introduced in Section 2.2. For the sake of precision, we restrict our attention the fragment of the Needham-Schroeder protocol where the key distribution phase (i.e., the messages exchanged with the trusted server) has been omitted. The resulting protocol, corresponding to messages 3, 6, and 7 of Section 2.2, is as follows:

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

and has been implemented using the specification illustrated in Figure 8 and the intruder theory in Figure 9. Let us discuss the rules in more detail.

The protocol rules have been encoded in the same way as for the *ffgg* protocol of Section 5, and directly correspond to the informal notation previously introduced. This time, all messages sent over the network are encrypted, therefore we have modeled them using atomic formulas like $n(pubk(id), mess\_content)$, where id is a principal identifier, whereas $mess\_content$ is a term of the form $mess(\ldots)$. Internal states of principals have been enriched in order to express the security violations we will present in the following. The intruder theory is shown in Figure 9. We consider here a more extended intruder theory w.r.t. the one for the *ffgg* protocol in Figure 4. Namely, we give the intruder the capability to decrypt messages addressed to himself/herself. Given that private keys are not exchanged,

$$
\begin{array}{ll}
i_1) & n(pubk(A), M) \multimap intercept(n(pubk(A), M)) \\
i_2) & intercept(n(pubk(A), M)) \multimap intercept(n(pubk(A), M)) \parr n(pubk(A), M) \\
i_3) & n(pubk(intruder), M) \multimap dec(M) \\
i_4) & dec(mess(M)) \multimap m(M) \\
i_5) & dec(mess(M, N)) \multimap m(M) \parr m(N) \\
i_6) & m(M) \multimap m(M) \parr comp(mess(M)) \\
i_7) & m(M) \parr m(N) \multimap m(M) \parr m(N) \parr comp(mess(M, N)) \\
i_8) & comp(C) \multimap n(pubk(Z), C)
\end{array}
$$

Figure 9: Intruder theory for the Needham-Schroeder protocol

the intruder will never be able to know the private keys of some other principal, therefore we still assume that the intruder is not able to decrypt messages other than the ones intended for himself/herself. Rules i_1 and i_2 deal with *interception*: the intruder can intercept any message and replay it (possibly more than once) at any given time in the future. Rule i_3 states that the intruder can *decompose* any message addressed to himself/herself. Rules i_4 and i_5 are the usual rules for *decomposition* of messages (with one or two components). Rules i_6 and i_7 are the corresponding rules for *composition* of messages (with one or two components). Finally, using rule i_8 the intruder can send a message to *any* principal.

Now, the specification of unsafe states is as follows:

$$
\begin{array}{ll}
u_1) & pr(alice, step3(NA, NB, bob)) \parr m(NA) \multimap \top \\
u_2) & pr(alice, step3(NA, NB, bob)) \parr m(NB) \multimap \top \\
u_3) & pr(bob, step4(NA, NB, alice)) \parr m(NA) \multimap \top \\
u_4) & pr(bob, step4(NA, NB, alice)) \parr m(NB) \multimap \top
\end{array}
$$

Namely, a state is unsafe if there exist two principals, say *alice* and *bob*, such that either *alice* has run the protocol to completion with *bob*, or *bob* with *alice*, and *at least* one of the two nonces has been disclosed to the intruder. We call the security property specified by the above rules *strong correctness*.

We can now run our verification tool on the resulting specification. As observed by Lowe [Lowe, 1995], Needham-Schroeder protocol is not safe w.r.t. the above security properties. In fact, we find the attack shown in Figure 10, corresponding to the one presented in [Lowe, 1995] (we have followed the same conventions as in Figure 7). The attack takes place because *alice* decides to contact the intruder, *without knowing he/she is cheating*. Thus, the intruder is able to impersonate

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash_{\Sigma_2} \top, al^3_{na,nb,i}, i^{init}, m(al,nb)}\top_r}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^4_{na,nb,al}, i^{init}, m(na,al,nb)}bc^{(u_3)}}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, m(na,al,nb), n(\{mess(nb)\}_{K_{bob}})}bc^{(p_4)}}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, m(na,al,nb), comp(mess(nb))}bc^{(i_8)}}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, m(na,al,nb)}bc^{(i_6)}}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, m(na,al), dec(mess(nb))}bc^{(i_4)}}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, m(na,al), n(\{mess(nb)\}_{K_i})}bc^{(i_3)}}{P \vdash_{\Sigma_2} al^1_{na,i}, bob^2_{na,nb,al}, i^{init}, m(na,al), n(\{mess(na,nb)\}_{K_{al}})}bc^{(p_3)}}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, m(na,al), n(\{mess(na,al)\}_{K_{bob}})}bc^{(p_2)}}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, m(na,al), comp(mess(na,al))}bc^{(i_8)}}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, m(na,al)}bc^{(i_7)}}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, dec(mess(na,al))}bc^{(i_5)}}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, n(\{mess(na,al)\}_{K_i})}bc^{(i_3)}}{P \vdash_{\Sigma} al^{init}, bob^{init}, i^{init}}bc^{(p_1)}$$

Figure 10: An attack to the Needham-Schroeder protocol: $\Sigma_1 = \Sigma, na$ and $\Sigma_2 = \Sigma, na, nb$

alice and cheat *bob*. Note that as a result the protocol has been broken from the point of view of *bob*. In fact, *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 (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 violations (we call the corresponding security property *weak correctness*).

$u'_1)$ $\quad pr(alice, step3(NA,NB,bob)) \parr pr(bob, step4(NA,NB,alice)) \parr m(NA) \circ\!\!- \top$

$u'_2)$ $\quad pr(alice, step3(NA,NB,bob)) \parr pr(bob, step4(NA,NB,alice)) \parr m(NB) \circ\!\!- \top$

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). This time the verification algorithm terminates, proving that Needham-Schroeder protocol is safe with respect to this property.

We conclude this section by showing how the methodology of *invariant strengthening* can improve the verification algorithm performance. Invariant strengthening is a *conservative* technique that can speed up the fixpoint computation, and consists in enriching the set of unsafe states with the negation of other invariants. Namely, we augment the set of security violations with this further rule:

$$u_3') \quad pr(alice, step1(NA, bob)) \ \mathcal{B} \ m(NA) \ \circ\!\!- \ \top$$

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 (see the experimental results in Table 14). Note that the following invariant is instead violated (the attack follows the same scheme of the one in Figure 10):

$$u_4') \quad pr(bob, step2(NA, NB, alice)) \ \mathcal{B} \ m(NB) \ \circ\!\!- \ \top$$

We stress 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 can be found in the original one.

8.3. Corrected Needham-Schroeder

As observed by Lowe [Lowe, 1995], the Needham-Schroeder 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 responder. Adding the responder's identity to this message prevents the intruder from replaying it, because now the initiator is expecting a message from the intruder. The corrected version of Needham-Schroeder protocol is

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

We need to make minor modifications to our previous specification. Namely, we modify rules p_2) and p_3) by adding the additional argument B:

p_2') $pr(B, init) \parr n(pubk(B), mess(NA, A)) \multimap \forall NB.(pr(B, step2(NA, NB, A)) \parr n(pubk(A), mess(B, NA, NB)))$

p_3') $pr(A, step1(NA, B)) \parr n(pubk(A), mess(B, NA, NB)) \multimap pr(A, step3(NA, NB, B)) \parr n(pubk(B), mess(NB))$

and we add two rules for composition and decomposition of messages with three components:

u_9) $dec(mess(M, N, O)) \multimap m(M) \parr m(N) \parr m(O)$

u_{10}) $m(M) \parr m(N) \parr m(O) \multimap m(M) \parr m(N) \parr m(O) \parr comp(mess(M, N, O))$

We can now use our algorithm to automatically verify if the protocol satisfy *strong correctness* (axioms u_1 through u_4). As in Section 8.2, we can accelerate convergence by means of invariant strengthening (see the experimental results in Table 14). We can use the two invariants u_3' and u_4' discussed in Section 8.2, which should now *both* hold. The verification algorithm terminates, as expected, proving that the modified version of the Needham-Schroeder protocol is correct w.r.t. the notion of *strong* correctness.

8.4. The Otway-Rees Protocol

The Otway-Rees protocol [Otway and Rees, 1987] provides a typical example of a *type flaw attack*. It is intended for *key distribution* between two principals communicating with a central server by means of shared keys (the protocol assumes *symmetric key encryption*, see Section 2). The protocol is as follows (the form presented here is the one given in [Burrows et al., 1989]).

1. $A \rightarrow B \;:\; N, A, B, \{N_a, N, A, B\}_{K_{as}}$
2. $B \rightarrow S \;:\; N, A, B, \{N_a, N, A, B\}_{K_{as}}, \{N_b, N, A, B\}_{K_{bs}}$
3. $S \rightarrow B \;:\; N, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
4. $B \rightarrow A \;:\; N, \{N_a, K_{ab}\}_{K_{as}}$

The protocol is run between two principals A and B, communicating with a *trusted server* S by means of shared keys K_{as} and K_{bs}. The purpose of the protocol is to get a new key K_{ab}, generated by the trusted server, to be used as a shared key in subsequent communications between A and B. At the first step, principal A generates a nonce N, to be used as a *run identifier*, and a nonce N_a, and sends

$p_1)$ $pr(A, init) \parr pr(B, init) \multimap pr(B, init) \parr \forall N.\forall NA.$
$(n(plain(cons(N, cons(A, B))), enc(sk(A, s), cons(NA, cons(N, cons(A, B)))) \parr$
$pr(A, step1(B, N, NA))))$

$p_2)$ $n(plain(cons(N, cons(A, B))), enc(sk(A, s), cons(NA, cons(N, cons(A, B))))) \parr$
$pr(B, init) \multimap \forall NB.(pr(B, step2(A, N, NB)) \parr$
$n(plain(cons(N, cons(A, B))), enc(sk(A, s), cons(NA, cons(N, cons(A, B)))),$
$enc(sk(B, s), cons(NB, cons(N, cons(A, B))))))$

$p_3)$ $n(plain(cons(N, cons(A, B))), enc(sk(A, s), cons(NA, cons(N, cons(A, B)))),$
$enc(sk(B, s), cons(NB, cons(N, cons(A, B))))) \parr pr(s, init) \multimap pr(s, init) \parr$
$\forall KAB.n(plain(N), enc(sk(A, s), cons(NA, KAB)), enc(sk(B, s), cons(NB, KAB))))$

$p_4)$ $n(plain(N), enc(sk(A, s), cons(NA, KAB)), enc(sk(B, s), cons(NB, KAB))) \parr$
$pr(B, step2(A, N, NB)) \multimap pr(B, step3(A, KAB)) \parr$
$n(plain(N), enc(sk(A, s), cons(NA, KAB)))$

$p_5)$ $pr(A, step1(B, N, NA)) \parr n(plain(N), enc(sk(A, s), cons(NA, KAB))) \multimap$
$pr(A, step4(B, KAB))$

Figure 11: Specification of the Otway-Rees protocol

to B the plaintext N, A, B and an encrypted message, readable only by the server S, of the form shown. In turn, principal B generates a nonce N_b and forwards A's message to S, together with a similar encrypted component. The server checks that the N, A, B components in both messages match, and, if so, generates a new key K_{ab} and replies to B with message 3 above, which includes a component intended for B and one for A. The component intended for A is forwarded to him/her by B with message 4.

We have encoded the Otway-Rees protocol with the rules in Figure 11. Let us discuss them in more detail. The encoding is similar to the one used for Millen's *ffgg* protocol (see Section 5), in particular we distinguish between the plaintext part of messages (term constructor *plain*) and the encrypted part (term constructor *enc*). Furthermore, we use a concatenation operator *cons* to glue together different components inside the plaintext or the ciphertext. In this way, we are able to capture type flaw attacks. We use an identifier s for the trusted server, and the notation $sk(id, s)$ to denote the shared key between principal id and s. Rules p_1 through p_5 are a direct translation of the protocol steps written in the usual notation. As usual, we have denoted the internal state of principals by means of term constructors like $step_i$. When the protocol is run to completion, the internal state of each of the two involved principals contains the identity of the other

$i_1)$ $n(plain(X), enc(sk(K,s),Y)) \multimap m(plain(X)) \parr m(enc(sk(K,s),Y))$

$i_2)$ $m(plain(X), enc(sk(K,s),Y), enc(sk(H,s),W)) \multimap m(plain(X)) \parr m(enc(sk(K,s),Y)) \parr m(enc(sk(H,s),W))$

$i_3)$ $m(plain(cons(X,Y))) \multimap m(plain(cons(X,Y))) \parr m(plain(X)) \parr m(plain(Y))$

$i_4)$ $m(plain(X)) \parr m(plain(Y)) \multimap m(plain(X)) \parr m(plain(Y)) \parr m(plain(cons(X,Y)))$

$i_5)$ $m(plain(X)) \parr m(enc(sk(K,s),Y)) \multimap m(plain(X)) \parr m(enc(sk(K,s),Y)) \parr n(plain(X), enc(sk(K,s),Y))$

$i_6)$ $m(plain(X)) \parr m(enc(sk(K,s),Y)) \parr m(enc(sk(H,s),W)) \multimap m(plain(X)) \parr m(enc(sk(K,s),Y)) \parr m(enc(sk(H,s),W)) \parr n(plain(X), enc(sk(K,s),Y), enc(sk(H,s),W))$

Figure 12: Intruder theory for the Otway-Rees protocol

principal and the new shared key obtained from the server. The intruder theory in presented in Figure 12. Rules i_1 and i_2 are the usual *decomposition* rules for, respectively, messages with one encrypted component and with two encrypted components. Rules i_3 and i_4 allow the intruder to arbitrarily decompose and reassemble plaintext messages, whereas encrypted messages are stored as they are. Finally, rules i_5 and i_6 allow the intruder to create new messages.

We wish to verify if the intruder can get the shared key which comes from a protocol run between two honest principals, say *alice* and *bob*. The specification of unsafe states is straightforward:

$$u_1) \quad pr(bob, step3(alice, KAB)) \parr m(plain(KAB)) \multimap \top$$
$$u_2) \quad pr(alice, step4(bob, KAB)) \parr m(plain(KAB)) \multimap \top$$

Running our verification tool, we automatically find the type flaw attack described in [Clark and Jacob, 1997]. The corresponding trace is shown in Figure 13 (we have followed the usual conventions, with $\Sigma_1 = \Sigma, n1, na$). The attack takes place because a malicious intruder can intercept the first message, and, after stripping it of the A and B components (in the plaintext part), replay it as the last message of the protocol. The attack is successful under the hypothesis that the triple N, A, B may be erroneously accepted, by the initiator of the protocol, as the desired key. This is clearly a security flaw because the triple N, A, B is sent in clear in the first message, and therefore publicly known. The Otway-Rees protocol provides a classical example of a type flaw attack. This kind of attacks are very

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash_{\Sigma_1} \top, bob^{init}, \mathcal{M}(\{na \cdot n1 \cdot al \cdot bob\}_{K_{as}}, n1, al \cdot bob)}\,\top_r}{P \vdash_{\Sigma_1} al^4_{bob,n1 \cdot al \cdot bob}, bob^{init}, \mathcal{M}(n1 \cdot al \cdot bob, \{na \cdot n1 \cdot al \cdot bob\}_{K_{as}}, n1, al \cdot bob)}\,bc^{(u_2)}}{P \vdash_{\Sigma_1} al^1_{bob,n1,na}, bob^{init}, \mathcal{M}(n1 \cdot al \cdot bob, \{na \cdot n1 \cdot al \cdot bob\}_{K_{as}}, n1, al \cdot bob), n(n1, \{na \cdot n1 \cdot al \cdot bob\}_{K_{as}})}\,bc^{(p_5)}}{P \vdash_{\Sigma_1} al^1_{bob,n1,na}, bob^{init}, \mathcal{M}(n1 \cdot al \cdot bob, \{na \cdot n1 \cdot al \cdot bob\}_{K_{as}}, n1, al \cdot bob)}\,bc^{(i_5)}}{P \vdash_{\Sigma_1} al^1_{bob,n1,na}, bob^{init}, \mathcal{M}(n1 \cdot al \cdot bob, \{na \cdot n1 \cdot al \cdot bob\}_{K_{as}})}\,bc^{(i_3)}}{P \vdash_{\Sigma_1} al^1_{bob,n1,na}, bob^{init}, n(n1 \cdot al \cdot bob, \{na \cdot n1 \cdot al \cdot bob\}_{K_{as}})}\,bc^{(i_1)}}{P \vdash_{\Sigma} al^{init}, bob^{init}}\,bc^{(p_1)}$$

Figure 13: An attack to the Otway-Rees protocol

Protocol	Invar	Steps	Size	MSize	Time	Verified
Millen's ffgg		14	306	677	1335	*attack*
Millen's ffgg (corrected)		14	27	810	2419	*yes*
Needham-Schroeder (strong correctness)		13	294	323	45	*attack*
Needham-Schroeder (weak correctness)		13	304	755	516	*yes*
	$\surd$	12	299	299	63	*yes*
Corrected Needham-Schroeder (strong correctness)		14	1575	1575	791	*yes*
	$\surd$	9	402	402	31	*yes*
Otway-Rees		5	10339	10339	4272	*attack*

Figure 14: Analysis of authentication protocols: experimental results

pervasive in authentication. Another classical example of type-flawed protocol is the Yahalom protocol [Clark and Jacob, 1997].

8.5. Summary of the Experimental Results

In Fig. 14 we summarize the experimental results for the examples presented in this paper. All the experiments have been performed on a Pentium II 233MhZ under Linux 2.0.32, running Standard ML of New Jersey, Version 110.0.7. The tag **Invar** indicates the use of *invariant strengthening* for accelerating fixpoint

convergence. Furthermore, **Size** denotes the number of multisets inferred during the evaluation, **MSize** the maximal number of multisets computed at any step, and **Time** the execution time in seconds.

9. Conclusions and Future Work

In this paper we have presented security protocols as a possible application field for our methodology based on a linear logic-based specification language and on a bottom-up evaluation strategy. Our verification procedure is tailored to study security violations which can be specified by means of *minimal conditions*. While this may rule out interesting properties, e.g., questions of *belief* [Burrows et al., 1989], the proposed approach can be used to study secrecy and confidentiality properties. No artificial limit is imposed on the number of simultaneous sessions.

We have performed some experiments on different authentication protocols that show that the methodology we propose can be effective either to find *attacks* of given protocols, or to prove that no attacks may exist (clearly, w.r.t. a given protocol theory and a given intruder theory). In other words, the bottom-up evaluation algorithm presented in Section 7.1 is correct and complete: is no proof is found (w.r.t. to the given protocol and intruder theories), then no proof at all may exist. We plan to overcome some current limitations of our approach, in particular we plan to refine and automatize the specification phase of protocols and of the intruder theory. Specifically, we want to study a (possibly automatic) translation between the usual informal description of protocols and our representation. As shown in the paper, 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. For efficiency reasons, it could also be worth to investigate some optimizations, in particular to the intruder theory (concerning, e.g., the rules for composition and decomposition). We plan to use techniques like *folding/unfolding* to automatize this process. Finally, we wish to optimize the ML prototype used for the experiments reported in Section 8.

Another topic we would like to investigate is *typed multiset rewriting* [Cervesato, 2001b], 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 [Cervesato, 2001b] 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 that 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 [Cervesato et al., 1999], one may ask if a more restricted subclass of protocols exists, for which the verification algorithm presented here is always terminating.

References

P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. Algorithmic Analysis of Programs with Well Quasi-Ordered Domains. *Information and Computation*, 160(1-2):109–127, 2000.

R. Amadio and D. Lugiez. On the Reachability Problem in Cryptographic Protocols. In C. Palamidessi, editor, *Proceedings 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pages 380–394, University Park, Pennsylvania, 2000. Springer-Verlag.

J.-M. Andreoli. Logic Programming with Focusing Proofs in Linear Logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

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

D. Basin. Lazy Infinite-State Analysis of Security Protocols. In R. Baumgart, editor, *Proceedings International Exhibition and Congress on Secure Networking (Cqre Secure'99)*, volume 1740 of *LNCS*, pages 30–42, Dusseldorf, Germany, 1999. Springer-Verlag.

B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Proceedings 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *LNCS*, pages 667–681, Heraklion, Crete, Greece, 2001. Springer-Verlag.

M. Bozzano. Ensuring Security through Model Checking in a Logical Environment (Preliminary Results). In *Proceedings Workshop on Specification, Analysis and Validation for Emerging Technologies in Computational Logic (SAVE'01)*, Paphos, Cyprus, 2001.

M. Bozzano. *A Logic-Based Approach to Model Checking of Parameterized and Infinite-State Systems*. PhD thesis, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, 2002.

M. Bozzano and G. Delzanno. Automated Protocol Verification in Linear Logic. In *Proceedings 4th International Conference on Principles and Practice of Declarative Programming (PPDP'02)*, pages 38–49, Pittsburgh, Pennsylvania, 2002. ACM Press.

M. Bozzano, G. Delzanno, and M. Martelli. A Bottom-up Semantics for Linear Logic Programs. In *Proceedings 2nd International Conference on Principles and Practice of Declarative Programming (PPDP'00)*, pages 92–102, Montreal, Canada, 2000. ACM Press.

M. Bozzano, G. Delzanno, and M. Martelli. An Effective Fixpoint Semantics for Linear Logic Programs. *Theory and Practice of Logic Programming*, 2(1):85–122, 2002a.

M. Bozzano, G. Delzanno, and M. Martelli. Model Checking Linear Logic Specifications, 2002b. Manuscript submitted for publication.

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

I. Cervesato. Petri Nets and Linear Logic: a Case Study for Logic Programming. In M. Alpuente and M. I. Sessa, editor, *Proceedings Joint Conference on Declarative Programming (GULP-PRODE'95)*, pages 313–318, Marina di Vietri, Italy, 1995. Palladio Press.

I. Cervesato. The Dolev-Yao Intruder is the Most Powerful Attacker. In J. Halpern, editor, *Proceedings 16th Annual International Symposium on Logic in Computer Science (LICS'01)*, Boston, Massachusetts, 2001a. Short paper.

I. Cervesato. Typed Multiset Rewriting Specifications of Security Protocols. In A. Seda, editor, *Proceedings 1st Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT'00)*, Cork, Ireland, 2001b.

I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A Meta-notation for Protocol Analysis. In *Proceedings 12th Computer Security Foundations Workshop (CSFW'99)*, pages 55–69, Mordano, Italy, 1999. IEEE Computer Society Press.

Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E. Brinksma and K. G. Larsen, editors, *Proceedings 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 324–337, Copenhagen, Denmark, 2002. Springer-Verlag.

H. Cirstea. Specifying Authentication Protocols Using Rewriting and Strategies. In I. V. Ramakrishnan, editor, *Proceedings of Conference on Practical Aspects of Declarative*

Languages (PADL'01), volume 1990 of *LNCS*, pages 138–152, Las Vegas, Nevada, 2001. Springer-Verlag.

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/`.

E. Cohen. TAPS: A First-Order Verifier for Cryptographic Protocols. In *Proceedings 13th Computer Security Foundations Workshop (CSFW'00)*, pages 144–158, Cambridge, England, 2000. IEEE Computer Society Press.

G. Delzanno. Specifying and Debugging Security Protocols via Hereditary Harrop Formulas and λProlog - A Case-study -. In H. Kuchen and K. Ueda, editors, *Proceedings 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, volume 2024 of *LNCS*, pages 123–137, Tokyo, Japan, 2001. Springer-Verlag.

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

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

F. Fages, P. Ruet, and S. Soliman. Linear Concurrent Constraint Programming: Operational and Phase Semantics. *Information and Computation*, 165(1):14–41, 2001.

T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proceedings 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *LNCS*, pages 271–290. Springer-Verlag, 2000.

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

F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'00)*, volume 1955 of *LNCS*, pages 131–160, Reunion Island, France, 2000. Springer-Verlag.

N. Kobayashi and A. Yonezawa. Asynchronous Communication Model based on Linear Logic. *Formal Aspects of Computing*, 7(2):113–149, 1995.

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

G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In T. Margaria and B. Steffen, editors, *Proceedings 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166, Passau, Germany, 1996. Springer-Verlag.

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

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.

R. McDowell, D. Miller, and C. Palamidessi. Encoding Transition Systems in Sequent Calculus. In J.-Y. Girard, M. Okada, and A. Scedrov, editors, *Proceedings Linear Logic 96 Tokyo Meeting*, volume 3 of *ENTCS*, Keio University, Tokyo, Japan, 1996.

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

J. K. Millen. CAPSL: Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997.

J. K. Millen. A Necessarily Parallel Attack. In *Proceedings Workshop on Formal Methods and Security Protocols*, Trento, Italy, 1999.

J. K. Millen and V. Shmatikov. Constraint Solving for Bounded-Process Cryptographic Protocol Analysis. In *Proceedings Conference on Computer and Communication Security (CCS'01)*, pages 166–175, Philadelphia, Pennsylvania, 2001. ACM Press.

D. Miller. The π-Calculus as a Theory in Linear Logic: Preliminary Results. In E. Lamma and P .Mello, editors, *Proceedings Workshop on Extensions of Logic Programming*, volume 660 of *LNCS*, pages 242–265, Bologna, Italy, 1992. Springer-Verlag.

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

D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

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

D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review*, 21(1):8–10, 1987.

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

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.

M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-Complete. In *Proceedings 14th Computer Security Foundations Workshop (CSFW'01)*, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

D. X. Song. A New Efficient Automatic Checker for Security Protocol Analysis. In *Proceedings 12th Computer Security Foundations Workshop (CSFW'99)*, pages 192–202, Mordano, Italy, 1999. IEEE Computer Society Press.

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