Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Symbolic Computation 38 (2004) 1375–1415
Journal of Symbolic Computation
www.elsevier.com/locate/jsc

Automatic verification of secrecy properties for linear logic specifications of cryptographic protocols

Marco Bozzano[a,*], Giorgio Delzanno[b]

[a] *ITC-IRST, Via Sommarive 18, Povo, 38050 Trento, Italy*
[b] *DISI, Università di Genova, via Dodecaneso 35, 16146 Genova, Italy*

Received 9 July 2003; accepted 9 April 2004
Available online 17 July 2004

Abstract

In this paper we investigate the applicability of a *bottom-up evaluation strategy* for a first-order fragment of affine linear logic that we introduced in Theory Prac. Log. Program. 4 (2004) 1 for the purposes of automated verification of *secrecy* in *cryptographic protocols*. Following the Proceedings of the 12th Computer Security Foundations Workshop (1999) 55, 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 verifying 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 verify secrecy for a protocol. We apply our method to analyse several classical examples of authentication protocols. Among them we consider the *ffgg* protocol (Proceedings of the Workshop on Formal Methods and Security Protocols (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 of the Otway–Rees protocol and several formulations of the Needham–Schroeder protocol.

Keywords: Linear logic; Bottom-up evaluation; Model checking; Authentication protocols

* Corresponding author. Tel.: +39-0461-314367; fax: +39-0461-302040.
E-mail addresses: bozzano@irst.itc.it (M. Bozzano), giorgio@disi.unige.it (G. Delzanno).

0747-7171/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jsc.2004.04.002

1. Introduction

Linear logic (Girard, 1987) provides a logical characterization of concepts and mechanisms peculiar to concurrency such as *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) and Miller (1996), *provability* in fragments of linear logic can be used then as a formal tool for reasoning about behavioural aspects of *concurrent systems* (see e.g., McDowell et al., 1996). In other paradigms for concurrency such as 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, 2002), we made a first attempt at relating these techniques to propositional fragments of linear logic, and, more precisely, to 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 such as 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 for computing 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*[1] 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 *colour* 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 style of π-calculus (Miller, 1992), or as a mechanism to generate *fresh names* as in Cervesato et al. (1999).

In Bozzano et al. (2004) and 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

[1] 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.

is crucial when trying to prove properties of *parametrized* 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*.

1.1. New contribution

In this paper we investigate the applicability of the bottom-up evaluation strategy of Bozzano et al. (2004) 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 analyse protocols. An incomplete list includes 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 the 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 this paper we will show that first-order LO (enriched with universally quantified goal formulas) is a reformulation of basic specification languages used for cryptographic protocols based on multiset rewriting such as MSR (Cervesato et al., 1999) and CIL (Millen, 1997). Working in LO, however, allows us to *directly* exploit the connection between *provability* and *verification of safety properties* that we explored in Bozzano et al. (2004). The formulation of LO that we propose here is indeed one possible syntax for an *affine* fragment of linear logic. However, as we have shown in our previous work (Bozzano et al., 2002), LO and its non-affine extensions have interesting operational interpretations in terms of goal-driven reachability. For this reason, having in mind possible future extensions of our work, we have decided to use LO syntax, and not to base our work directly on affine linear logic.

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 for verifying 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 in 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 analyse well-known examples of authentication protocols taken from the literature on security, such as the Needham–Schroeder protocol (Needham and Schroeder, 1978), a *corrected* version of the Needham–Schroeder protocol, and the Otway–Rees protocol (Otway and Rees, 1987). In addition we report on the experiment with the *ffgg* protocol introduced by 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 for analysing interesting aspects of authentication such as secrecy and confidentiality.

1.2. 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 process of compilation from security protocol descriptions into rewrite rules. The resulting specifications are then executed using the daTac theorem prover. Unlike that of 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 analyse 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 modelled 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 the 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 (1996) to analyse the Needham–Schroeder public key protocol. Other works which fall into this class are Marrero et al. (1997) and 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 analyse a fixed number of concurrent protocol runs, an approach which is infeasible as this number increases. A reduction to finite-state models is also used in Armando and Compagna (2003) and Armando et al. (2003), where a protocol insecurity problem is reduced to a planning problem, and then to a problem in propositional logic which is verified using a state-of-the-art SAT solver. The authors discuss a specialized encoding and an abstraction/refinement strategy to optimize the search. The approach is still limited to protocol debugging.

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), and 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). In contrast to the above mentioned approaches, ours uses 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 analysed using the theorem prover Isabelle. Here, analysis is a *semi-automatic* process which can take several days. The NRL protocol analyser (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 in our procedure, 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. Inductive lemmas, which are generated within the analyser with human help, are used to guide the search.

The Athena algorithm proposed in Song (1999) automates 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 analyser, Athena can be used both for debugging and verification, and needs pruning techniques (e.g., the unreachability lemma of Song (1999) and 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. In contrast, 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, as compared to Blanchet (2001), we use a cleaner treatment for nonces, and we do not 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). Unlike Blanchet's method, the TAPS verifier generates *state dependent* invariants from the protocol model. These invariants are combined 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.

As regards 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 the CIL intermediate language (essentially, multiset rewriting with name generation) and used to feed tools such as Maude (Denker et al., 1998) or the NRL analyser (Meadows, 1996). Finally, Jacquemard et al. (2000) presents an automatic process of compilation into rewriting rules which is able to manage infinite-state models.

As regards 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. (2004) (where, as a practical example, we have studied a parametrized *mutual exclusion* protocol). The first author's Ph.D. Thesis (Bozzano, 2002) also contains a detailed presentation and proofs for the results presented in this paper. Some preliminary results (e.g., the Needham–Schroeder protocol) were also discussed in Bozzano (2001).

1.3. 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 notation 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, and 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 in 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 being intercepted, and the presence of malicious agents who can impersonate honest principals. One of the key issues in authentication is to ensure *confidentiality*, i.e., to avoid private information being disclosed to unintended 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 as 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 an *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 and 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 protocols require the use of *digital signature*, in order to ensure authenticity (rather than *confidentiality*) of the messages. The basic mechanism of digital signature also requires the use of a public and a private key. The private key is used by a principal to encrypt a message whose authenticity needs to be certified, whereas the public key can be used by the receiver of the message to decrypt the message (successful decryption entails authenticity of the message).

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 who 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 of *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 that 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 the intruder 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.

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. $A \rightarrow B \;:\; \{N_b\}_{K_b}$

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 for digital signature).

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 that 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 that nonces are not guessable, the protocol should prevent a malicious intruder from impersonating 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 $LO_\forall$

LO (Andreoli and Pareschi, 1991) is a logic programming language based on a fragment of LinLog (Andreoli, 1992). Its mathematical foundations lie in a proof-theoretical presentation of a fragment of affine linear logic (i.e., linear logic with weakening) 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 $LO_\forall$ 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:

$$\mathbf{G} ::= \mathbf{G} \parr \mathbf{G} \mid \mathbf{G} \,\&\, \mathbf{G} \mid \forall x.\mathbf{G} \mid \mathbf{A} \mid \top \mid \bot$$
$$\mathbf{H} ::= \mathbf{A} \parr \cdots \parr \mathbf{A} \mid \bot$$
$$\mathbf{D} ::= \forall (\mathbf{H} \circ\!\!- \mathbf{G})$$

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

An $LO_\forall$ 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

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

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

Fig. 1. A proof system for $LO_\forall$.

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 $LO_\forall$ sequent has the form

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

where P is an $LO_\forall$ program over Σ_P and $\mathcal{V}$, $G_1, \ldots, G_k$ is a context 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 (the top-down construction of LO proofs, and in particular the rule dealing with the universal quantifier, may require extending the original signature, as we will explain in Section 3.1).

3.1. Top-down provability

We now define provability in $LO_\forall$. Let Σ be a signature with predicates and $\mathcal{V}$ a denumerable set of variables. Given an $LO_\forall$ 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 $Gnd_\Sigma(P)$, is defined as follows:

$$Gnd_\Sigma(P) = \{(H \multimap G)\theta \mid \forall (H \multimap G) \in P\},$$

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 $LO_\forall$ sequent $P \vdash_\Sigma\ G_1, \ldots, G_k$.

The operational semantics of $LO_\forall$ is given via the *uniform* (*focusing*) (Andreoli, 1992) proof system presented in Fig. 1, where P is a set of clauses, $\mathcal{A}$ is a multiset of atomic formulas, and Δ is a context, i.e., 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 Fig. 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 Fig. 1 is a specialization of more general uniform proof systems for linear logic such as 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 that $\bot \circ\!\!- G \in Gnd_\Sigma(P)$

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:

$$\frac{\dfrac{}{P \vdash_\Sigma \top, \mathcal{A}}\, \top_r}{P \vdash_\Sigma \widehat{H}, \mathcal{A}}\, bc$$
provided that $H \circ\!\!- \top \in Gnd_\Sigma(P)$

The previous observation can be stated as follows. Let $\preccurlyeq$ denote the multiset inclusion relation. Given an $LO_\forall$ program P and two multisets of goals Δ, Δ' such that $\Delta \preccurlyeq \Delta'$, if $P \vdash_\Sigma \Delta$ then $P \vdash_\Sigma \Delta'$. This property is known as *admissibility* of the weakening rule (Bozzano et al., 2004). It qualifies the language LO as being a subset of *affine* linear logic.

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 Fig. 2 (where $bc^{(i)}$ denotes the 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.

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

Fig. 2. An example of $LO_\forall$ 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 $LO_\forall$ 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 3.2 (Operational Semantics). Given an $LO_\forall$ program P, its operational semantics, denoted $\mathcal{O}(P)P$, is given by

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

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 $LO_\forall$

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 the notation

$$pr(id, s)$$

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 (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 such as *init* (indicating the initial state of a principal, before protocol execution), or a term such as

$$step_i(data),$$

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 for a given principal to take part in different protocol runs, possibly with different *roles*. Messages sent over a given network can in turn be represented by terms such as

$$n(mess\_content),$$

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*. The *generic* intruder theory introduced in Cervesato et al. (1999) can be directly reformulated in our framework by introducing a *concatenation* operator for forging new messages starting from simpler ones. For instance, let *cons* be such a binary constructor. Then, a message consisting of three components a, b, and c, can be represented as a term $cons(a, cons(b, c))$. Under these assumptions, the intruder theory contains generic rules of the following form (see Cervesato et al., 1999, for more details):

$$\begin{array}{ll}
n(X) \multimap m(X) & (intercept)\\
m(cons(X,Y)) \multimap m(X) \parr m(Y) & (decompose)\\
m(X) \parr m(Y) \multimap m(cons(X,Y)) & (compose)\\
\ldots & \\
m(X) \multimap m(X) \parr n(X) & (forge)
\end{array}$$

Through these rules, the intruder can forge messages of arbitrary shape. However, as shown in Lowe (1999), the intruder only needs to generate messages that the honest agents can recognize. The shape of these messages is determined by the protocol rules. Following this idea, in this paper we will define intruder theories specific to every example by specializing the generic rules of Cervesato et al. (1999) to the message patterns of the protocol taken into consideration. We will further discuss this point in Section 9.

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 that 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:

$$e_1)\ \ \bot \circ\!\!- \forall ID.(pr(ID, init))$$
$$e_2)\ \ pr(Z, S) \circ\!\!- pr(Z, S) \parr pr(Z, init)$$

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. Finally, we have the following two rules for the intruder theory:

$$t_1)\ \ pr(Z, S) \circ\!\!- pr(Z, S) \parr m(Z)$$
$$t_2)\ \ \bot \circ\!\!- \forall N.(m(N))$$

The first rule allows the intruder to store a principal identifier, whereas the second one formalizes the capability of the intruder of generating 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: 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, i.e., the protocol is secure if processes are serialized. The protocol is as follows.

1. $A \rightarrow B\ :\ A$
2. $B \rightarrow A\ :\ N_1, N_2$

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

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

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

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

$p_5)$ $pr(A, step3(B, S)) \parr n(plain(N1, X), enc(pubk(B), plain(X, Y, N1))) \circ\!\!- pr(A, step5(B))$

Fig. 3. Specification of the *ffgg* protocol.

3. $A \rightarrow B \;:\; \{N_1, N_2, S\}_{K_b} \% \{N_1, X, Y\}_{K_b}$
4. $B \rightarrow A \;:\; N_1, X, \{X, Y, N_1\}_{K_b}$

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 that one is interested in analysing is whether the secret S can be disclosed to a malicious intruder. We have implemented the *ffgg* protocol through the specification shown in Fig. 3, while the intruder theory is presented in Fig. 4. The specification consists of a set of protocol rules (rules p_1 to p_5 in Fig. 3) and an intruder theory (rules i_1 to i_{12} in Fig. 4). We remind the reader that the 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 $step_i$ to model the different steps of a protocol run. At every step, each principal needs to remember the identifier of the other principal that he/she is executing the protocol with. In addition, at step 2 the responder stores the first nonce created (in order to be 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 modelled 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 the term constructors *plain*(...) and *enc*(...) (to be precise, we should say a *family* of term constructors, as we find it convenient to overload the same symbol with different *arities*). The *plain* constructor is used to group plain components inside messages, whereas the *enc* constructor is used to represent encryption (the first argument is the key the message is to be encrypted with, and the second component is the message content).

The intruder theory is made up of rules i_1 to i_{12} in Fig. 4. Let us discuss it in more detail. Rules i_1 to i_4 are the basic *decomposition* rules. We have four rules dealing with the different formats of messages exchanged in the *ffgg* protocol. For instance, rule i_1 deals

$$i_1)\quad n(plain(X)) \circ\!\!- m(plain(X))$$

$$i_2)\quad n(plain(X,Y)) \circ\!\!- m(plain(X)) \parr m(plain(Y))$$

$$i_3)\quad n(enc(X,Y)) \circ\!\!- m(enc(X,Y))$$

$$i_4)\quad n(plain(X,Y), enc(U,V)) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(enc(U,V))$$

$$i_5)\quad m(plain(X)) \circ\!\!- m(plain(X)) \parr n(plain(X))$$

$$i_6)\quad m(plain(X)) \parr m(plain(Y)) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr n(plain(X,Y))$$

$$i_7)\quad m(enc(X,Y)) \circ\!\!- m(enc(X,Y)) \parr n(enc(X,Y))$$

$$i_8)\quad m(plain(X)) \parr m(plain(Y)) \parr m(enc(U,V)) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(enc(U,V)) \parr n(plain(X,Y), enc(U,V))$$

$$i_9)\quad n(enc(pubk(intruder), plain(X,Y,Z))) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(plain(Z))$$

$$i_{10})\quad n(plain(X,Y), enc(pubk(intruder), plain(U,V,Z))) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(plain(U)) \parr m(plain(V)) \parr m(plain(Z))$$

$$i_{11})\quad m(plain(X)) \parr m(plain(Y)) \parr m(plain(U)) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(plain(U)) \parr n(enc(pubk(Z), plain(X,Y,U)))$$

$$i_{12})\quad m(plain(X)) \parr m(plain(Y)) \parr m(plain(U)) \parr m(plain(V)) \parr m(plain(W)) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(plain(U)) \parr m(plain(V)) \parr m(plain(W)) \parr n(plain(X,Y), enc(pubk(Z), plain(U,V,W)))$$

Fig. 4. Intruder theory for the *ffgg* protocol.

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. It is assumed that the intruder cannot further decompose encrypted components, which in fact are stored exactly as they are, whereas plain messages are decomposed into their atomic constituents. Rules i_5 to i_8 are the basic *composition* rules. As for the decomposition rules, each rule deals with a different message format. It is easy to see that these composition and decomposition rules are specializations of the generic intruder rules presented in Section 4.

In addition to the previous rules, we have formalized some further capabilities of the intruder (rules i_9 to i_{12}). In particular, rules i_9 and i_{10} allow the intruder to decompose messages that are directly addressed to himself/herself, whereas rules i_{11} and i_{12} allow the intruder to encrypt messages, starting from previously stored plain components, using an arbitrary key. Even if these capabilities are not strictly required to illustrate the attack which can mounted on the *ffgg* protocol (as we will see in Section 8.1), they have been added for completeness. These same capabilities, in contrast, are crucial in the case of the Needham–Schroeder protocol (see Section 8.2).

The intruder theory that 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).

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 $LO_\forall$ theory and MSR (Cervesato et al., 1999), a consolidated interleaving computational model for cryptographic protocols.

6. Adequacy of $LO_\forall$-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

$$A_1 \otimes \cdots \otimes A_n \multimap \exists x.\ B_1 \otimes \cdots \otimes B_m,$$

where $\otimes$ is a linear logic *multiplicative conjunction*, meaning that $A_1, \ldots, A_n$ evolve into $B_1, \ldots, B_m$ by creating a new name for x.

In LO with universal quantification the same effect is obtained via the clause

$$A_1 \parr \ldots \parr A_n \circ\!\!- \forall x. B_1 \parr \ldots \parr B_m.$$

In fact, in the goal-driven proof system of LO a computation step is obtained 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 effects of quantification are the same in the two cases. The immersion of MSR in linear logic and the duality of $\otimes$ and $\parr$ give us an immediate proof of equivalence of the two formalisms. However, in order to illustrate the *operational* flavour of LO, in the rest of the section we will present a direct comparison with rewriting in MSR. 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 consist 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 $G = \{p(a) \parr p(b) \parr q(f(b))\}$. Fig. 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 $LO_\forall$ 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 $LO_\forall$.

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

Fig. 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 such as

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

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 $\mathcal{M}_1$ 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

$$\forall (A_1 \parr \ldots \parr A_n \multimap \forall x_1 \ldots \forall x_k.\ B_1 \parr \ldots \parr B_m).$$

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 result 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 R 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 $\overline{\mathcal{N}_1} \circ\!\!- \forall x_1 \ldots \forall x_k.\, \overline{\mathcal{N}_2}$ 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 the $\forall_r$ rule k times, 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 the $\parr_r$ rule m times 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 $\mathrm{LO}_\forall$ 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 that 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, the $\forall_r$ rule a finite number of times, and, finally, the $\parr_r$ rule a finite number of times, until the current goal

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

Fig. 6. A logical representation of an infinite set of unsafe configurations for the *ffgg* protocol.

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 an intruder) is represented by a partial* $\mathrm{LO}_\forall$ *derivation in $lo(S)$. Vice versa, every partial* $\mathrm{LO}_\forall$ *derivation of a protocol T specified in simple* $\mathrm{LO}_\forall$ *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* (to be defined formally in Section 7.1) of a collection of *minimal violations* (similarly to the specification of mutual exclusion properties of communication protocols Abdulla et al. (2000)). Intuitively, this property means that whenever a given configuration is considered to be unsafe (e.g., two principals are running a protocol session, and a given secret has been unintentionally disclosed to the intruder) every possible extension of this configuration is still unsafe (e.g., no matter how many additional principals are running, no matter how many additional messages are being exchanged on the network).

Thanks to this property, it often becomes possible to *finitely* represent infinite collections of unsafe configurations (i.e., by means of finite configurations representing their minimal violations). Symbolic procedures can then be applied in order to saturate the set of predecessor states (by iteratively applying a transition relation *backwards*[2]). 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*.

Let us see an example. In the case of the *ffgg* protocol, we can consider a configuration to be *unsafe* if there exists a run of the protocol between two honest principals, say *alice* and *bob*, such that *alice* has generated the secret S (at step 3 of the protocol execution), *bob* has run the protocol to completion with *alice* (step 4), 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 Fig. 6. Every *top-down* derivation leading from an initial goal (state)

[2] 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 in a *bottom-up* fashion.

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 the $\mathrm{LO}_\forall$ program we introduced in Bozzano et al. (2004). 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 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 (such 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 such as 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. (2004) and 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* with respect to 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 definitions. In the following, Σ_P is the signature associated to the program P, and Sig_P is the set of all extensions of Σ_P with new constant symbols.

Definition 7.1 (Herbrand Base). Given an $\mathrm{LO}_\forall$ program P, the Herbrand base of P, denoted as $HB(P)$, is given by

$$HB(P) = \{\mathcal{A} \mid \mathcal{A} \text{ is a multiset of (non-ground) atoms in } A^{\mathcal{V}}_{\Sigma_P}\}.$$

Definition 7.2 (Interpretation). Given an $\mathrm{LO}_\forall$ program P, an interpretation I is any subset of $HB(P)$. The denotation of an interpretation I, denoted as $[\![I]\!]$, 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.

The $[\![\cdot]\!]_\Sigma$ operator performs instantiation and upward closure. Given a set of multisets, it saturates the set with respect to all multisets which can be obtained by variable instantiation and multiset union from the original ones. 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 to/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 with respect 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. 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 the interpretation I (in the following definition, I is intended to denote the upward closure of an interpretation, according to Definition 7.2). Intuitively, if $I \Vdash_{\Sigma_P} \Delta \blacktriangleright \mathcal{C} \blacktriangleright \theta$ holds then the sequent $P, P' \vdash_{\Sigma_P} \Delta\theta\gamma, \mathcal{C}\theta\gamma$, γ being a grounding substitution, is provable by augmenting P with the program P' consisting of clauses such as $\mathcal{A} \circ\!\!- \top$ 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 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.

We give the following definition (for simplicity, we present only the formal definition of the judgment for goals without conjunction; see Bozzano et al. (2004) for the complete definition). Below, $\mathcal{A} \backslash \mathcal{B}$ 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*. Furthermore, we remark that in general, two multisets may have more than one (not necessarily equivalent) most general unifier (see Bozzano et al. (2004)). 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 $\mathcal{A}'$.

Definition 7.3 (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 : $I \Vdash_\Sigma \top, \Delta \blacktriangleright \epsilon \blacktriangleright \text{nil};$

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), $\mathcal{B}' \preccurlyeq \mathcal{B}$, $\mathcal{A}' \preccurlyeq \mathcal{A}$, $|\mathcal{B}'| = |\mathcal{A}'|,\ \mathcal{C} = \mathcal{B} \backslash \mathcal{B}'$, and $\theta = m.g.u.(\mathcal{B}', \mathcal{A}')_{|FV(\mathcal{A},\mathcal{C})}$

The rules presented above have a direct correspondence with the rules defining the top-down provability of LO presented in Fig. 1 (as previously mentioned, here for simplicity we have left out the rule for the conjunction). The first four rules are derived directly from their top-down counterparts. The last rule deals with the case of atomic multisets. The intuition underlying this rule is that the multiset of atomic formulas $\mathcal{A}$ is provable if it possible to single out a sub-multiset $\mathcal{A}'$ of $\mathcal{A}$ and a sub-multiset $\mathcal{B}'$ of $\mathcal{B}$, such that $\mathcal{A}'$ and $\mathcal{B}'$ are unifiable. If this is the case, then what is left out in $\mathcal{B}$ (i.e., $\mathcal{B}\backslash\mathcal{B}'$) is returned as the output of the judgment (intuitively, $\mathcal{B}\backslash\mathcal{B}'$ is *required* for $\mathcal{A}$ to be provable in I), whereas what is left out in $\mathcal{A}$ (i.e., $\mathcal{A}\backslash\mathcal{A}'$) can be disregarded (intuitively, the admissibility of the weakening rule ensures that if $\mathcal{A}'$ is provable, then also $\mathcal{A}$ is provable). Finally, the output substitution θ provides the variable bindings produced by the unification operation.

Remark 7.4. 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. In this way, universal quantification is always resolved *locally*.

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 recall that, given $H = A_1 \parr \ldots \parr A_k$, $\widehat{H}$ is the multiset $A_1, \ldots, A_k$.

Definition 7.5. Given an LO$_\forall$ program P and an interpretation I, the symbolic fixpoint operator S_P is defined as follows:

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

Example 7.6. 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 braces in multiset notation), and P the program

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

Let us 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 $p(f(w')), r(w')$ belongs to $S_P(I)$ (in fact, any of its instances is provable in P enriched with $p(x) \parr q(x) \circ\!\!- \top$). 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 $S_P(I)$, too. Notice that the latter multiset denotes *redundant* information with respect to the denotations of $\mathcal{B} = p(x'), q(x')$. In fact $[\![\{\mathcal{A}\}]\!] \subseteq [\![\{\mathcal{B}\}]\!]$.

Let us 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 the $\Vdash_{\Sigma_P}$ 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 cannot 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 $p(x) \parr q(x) \circ\!\!- \top$.

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., 2004). 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. (2004)) holds.

Theorem 7.7 (Soundness and Completeness (Bozzano et al., 2004)). *Let P be an $\mathrm{LO}_\forall$ program. Then, $\mathcal{O}(P) = [\![\mathcal{F}(P)]\!]_{\Sigma_P}$.*

Finally, we can define an effective test for *subsumption* between multisets of non-ground goals, in accordance with the notion of rich denotations of Definition 7.2. 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 7.8 (Bozzano et al., 2004). *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 both to prune the set of multisets which are generated by the least fixpoint computation built on top of the operator S_P (i.e., to discard multisets which are subsumed by other ones, during the intermediate steps), and also as a symbolic termination test for the fixpoint computation. 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. (2004).

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 $\mathcal{A}_1 \circ\!\!- \top, \ldots, \mathcal{A}_k \circ\!\!- \top$ (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 7.9 (Ensuring Secrecy). *The protocol $\mathcal{P}$ is secure with respect to 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 7.7. □

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 7.10 (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 Fig. 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 Fig. 6.

As in traditional model checking, counterexample traces can be automatically generated whenever a violation is found. In particular, the trace corresponding to the above attack is shown in Fig. 7 (we only post-processed the output of our verification tool to show the trace in a more human-readable form). The trace in Fig. 7 corresponds to an $\mathrm{LO}_\forall$ derivation which leads from an initial state to a state violating the security property of Fig. 6. The attack is exactly the *parallel session* one described in Millen (1999), that is, using

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

Fig. 7. A parallel session attack on 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; 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 $\Sigma_3 = \Sigma, n1, n2, n3, n4, s$.

$p_1)$ $pr(A, init) \parr pr(B, init) \multimap pr(B, init) \parr \forall NA.(pr(A, step1(NA, B)) \parr n(pubk(B), plain(NA, A)))$

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

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

$p_4)$ $pr(B, step2(NA, NB, A)) \parr n(pubk(B), plain(NB)) \multimap pr(B, step4(NA, NB, A))$

Fig. 8. Specification of the Needham–Schroeder protocol.

$i_1)$ $n(enc(pubk(A), X)) \multimap m(enc(pubk(A), X))$

$i_2)$ $n(enc(pubk(intruder), plain(X))) \multimap m(plain(X))$

$i_3)$ $n(enc(pubk(intruder), plain(X, Y))) : -m(plain(X)) \parr m(plain(Y))$

$i_4)$ $m(enc(pubk(A), X)) \multimap m(enc(pubk(A), X)) \parr n(enc(pubk(A), X))$

$i_5)$ $m(plain(X)) \multimap m(plain(X)) \parr n(enc(pubk(Z), plain(X)))$

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

Fig. 9. Intruder theory for the Needham–Schroeder protocol.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, i^{init}, \mathcal{M}(al, na)}\top_r
}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^4_{na,nb,al}, i^{init}, \mathcal{M}(al, na, nb)}bc^{(u_4)}
}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, \mathcal{M}(al, na, nb), n(\{nb\}_{K_{bob}})}bc^{(p_4)}
}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, \mathcal{M}(al, na, nb)}bc^{(i_5)}
}{P \vdash_{\Sigma_2} al^3_{na,nb,i}, bob^2_{na,nb,al}, i^{init}, \mathcal{M}(al, na), n(\{nb\}_{K_i})}bc^{(i_2)}
}{P \vdash_{\Sigma_2} al^1_{na,i}, bob^2_{na,nb,al}, i^{init}, \mathcal{M}(al, na), n(\{na, nb\}_{K_{al}})}bc^{(p_3)}
}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, \mathcal{M}(al, na), n(\{na, al\}_{K_{bob}})}bc^{(p_2)}
}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, \mathcal{M}(al, na)}bc^{(i_6)}
}{P \vdash_{\Sigma_1} al^1_{na,i}, bob^{init}, i^{init}, n(\{na, al\}_{K_i})}bc^{(i_3)}
}{P \vdash_{\Sigma} al^{init}, bob^{init}, i^{init}}bc^{(p_1)}
$$

Fig. 10. An attack on the Needham–Schroeder protocol: i stands for *intruder*, $\Sigma_1 = \Sigma, na$, and $\Sigma_2 = \Sigma, na, nb$.

the usual protocol notation (where primed numbers denote a session which is carried out in parallel with the first one, whereas identifiers in parentheses indicate interference by the intruder, i.e., interception of messages or sending of messages forged or modified by the intruder):

1. $A \to B \quad : A$
1′. $(A) \to B' \; : A$
2. $B \to (A) \; : N_1, N_2$
2′. $B' \to (A) \; : N_3, N_4$
2. $(B) \to A \; : N_1, N_3$

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

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

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

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

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

Fig. 11. Specification of the Otway–Rees protocol.

$i_1)$ $n(plain(X), enc(Y, Z)) \circ\!\!- m(plain(X)) \parr m(enc(Y, Z))$

$i_2)$ $m(plain(X), enc(Y, U), enc(V, W)) \circ\!\!- m(plain(X)) \parr$
$m(enc(Y, U)) \parr m(enc(V, W))$

$i_3)$ $m(plain(cons(X, Y))) \circ\!\!- m(plain(cons(X, Y))) \parr m(plain(X)) \parr m(plain(Y))$

$i_4)$ $m(plain(X)) \parr m(plain(Y)) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr$
$m(plain(cons(X, Y)))$

$i_5)$ $m(plain(X)) \parr m(enc(Y, U)) \circ\!\!- m(plain(X)) \parr m(enc(Y, U)) \parr$
$n(plain(X), enc(Y, U))$

$i_6)$ $m(plain(X)) \parr m(enc(Y, U)) \parr m(enc(V, W)) \circ\!\!- m(plain(X)) \parr$
$m(enc(Y, U)) \parr m(enc(V, W)) \parr n(plain(X), enc(Y, U), enc(V, W))$

Fig. 12. Intruder theory for the Otway–Rees protocol.

3. $A \to B \quad : \{N_1, N_3, S\}_{K_b}$
4. $B \to (A) \quad : N_1, N_3, \{N_3, S, N_1\}_{K_b}$
3′. $(A) \to B' \quad : \{N_3, S, N_1\}_{K_b}$
4′. $B' \to (A) \quad : N_3, S, \{S, N_1, N_3\}_{K_b}$

This attack is also an example of a *type flaw* attack, in that it relies on the secret S being 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 Fig. 7, we present below some of the steps performed by the bottom-up evaluation algorithm. In the following we follow the

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

Fig. 13. An attack on the Otway–Rees protocol: $\Sigma_1 = \Sigma, n1, na$.

Protocol	Invar	Steps	Size	MSize	Time	Verified
Millen's ffgg		14	304	2226	5779	*attack*
Millen's ffgg (*corrected*)		14	36	10787	54210	*yes*
Needham–Schroeder (*strong correctness*)		9	216	216	9	*attack*
Needham–Schroeder (*weak correctness*)		11	232	651	207	*yes*
	√	11	227	227	17	*yes*
Corrected Needham–Schroeder (*strong correctness*)		11	755	755	141	*yes*
	√	8	258	258	9	*yes*
Otway–Rees		6	40259	40259	222848	*attack*

Fig. 14. Analysis of authentication protocols: experimental results.

same syntactical notation as in Fig. 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 is decomposition rule i_4. We can apply a variant of i_4, let it be

$$n(X', Y', V'_{K_{U'}}) \circ\!\!- m(X') \parr m(Y') \parr m(V'_{K_{U'}})$$

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 $m(V'_{K_{U'}})$) 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

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

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

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

to m_2, in the following way: unify V' with $\{X'', Y'', N1''\}$, $N1''$ with X', X'' with S, and B'' with U' (thus unifying the atoms $n(N1'', X'', \{X'', Y'', N1''\}_{K_{B''}})$ and $n(X', S, V'_{K_{U'}})$). 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. The reader may notice that an instance of a principal B'' has been introduced in multiset m_3. We also invite the reader to observe the correspondence between the *bottom-up* construction that we are sketching and the *top-down* construction illustrated in Fig. 7. Notice that the sequence of rules that 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 of 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 the 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 in m_3) will be unified with *bob*, and similarly, A'' will be unified with *al*. Proceeding as above, by applying the rules i_7 and i_4 we get the multisets (we leave the details to the reader)

$$m_4) \quad al^3_{bob,S}, bob^4_{al}, (B'')^2_{A'',N1''}, m(\{N1'', S, Y''\}_{K_{B''}})$$
$$m_5) \quad al^3_{bob,S}, bob^4_{al}, (B'')^2_{A'',N1''}, n(W, Z, \{N1'', S, Y''\}_{K_{B''}})$$

Now, we can apply a variant of rule p_4, let it be

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

to m_5, in the following way: unify B' and B'' with *bob*, A' with *al*, $N1''$ with X' and Z, Y'' with $N1'$ and W, and S with Y'. We get the multiset

$$m_6) \quad al^3_{bob,S}, bob^2_{al,Y''}, bob^2_{A'',N1''}, n(\{Y'', N1'', S\}_{K_{bob}})$$

The reader may notice that the variable B'' has been unified with *bob*, as we previously anticipated. We conclude with an example of application of a clause involving

universal quantification. We can now apply a variant of protocol rule p_3 (compare with 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. According to rule *forall* for the satisfiability judgment (see Definition 7.3), a new constant, let it be c, has to be introduced in place of the universally quantified variable S' in the body of the above clause. By unifying A' with *al*, B' with *bob*, S with c, Y'' with $N1'$, and $N1''$ with $N2'$, we get the following multiset:

$$m_7) \quad al^1_{bob}, bob^2_{al,Y''}, bob^2_{A'',N1''}, n(Y'', N1'')$$

Notice that, according to rule *forall* for the satisfiability judgment (see also Remark 7.4), 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 (note that the binding between the variable S and the constant c is not contained in the output unifier, because of the restriction to $FV(\mathcal{A}, \mathcal{C})$ which is enforced in the rule for atomic multisets in Definition 7.3); therefore 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 Fig. 7) is reached. We leave the details to the reader.

We conclude by mentioning that we have also performed some further experiments as regards Millen's *ffgg* protocol, which we do not 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 protocol specification introduced in Section 5, principal B only checks that the first component of the last message 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 with respect to the security property and the intruder theory that 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* for 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 the 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 exist. 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 the generation of fresh names. In contrast, the backward application of $\mathrm{LO}_\forall$ rules allows us instead to observe only formulas defined over the signature of the original program. As an example of generation of a fresh name, the reader may refer to the explanation of how the bottom-up search deals with the generation of the secret S in the case of the *ffgg* protocol (discussed earlier in this section). In general, assume 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 7.3) 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 7.3) and then we match the resulting multiset of formulas against the current interpretation (see rule *atomic multiset* of Definition 7.3). In the end a *static check* is made in order to ensure that the *output multiset* and the *resulting unifier* 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. In contrast, 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 analyse 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:

1. $A \rightarrow B \;:\; \{N_a, A\}_{K_b}$
2. $B \rightarrow A \;:\; \{N_a, N_b\}_{K_a}$
3. $A \rightarrow B \;:\; \{N_b\}_{K_b}$

and has been implemented using the specification illustrated in Fig. 8 and the intruder theory in Fig. 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. All messages sent over the network are encrypted; therefore we have modelled them using atomic formulas such as $n(pubk(id), mess\_content)$, where id is a principal identifier and $mess\_content$ represents the message content. Internal states of principals have been enriched in order to express the security violations to be investigated. The intruder theory is shown in Fig. 9. It is analogous to the intruder theory for the *ffgg* protocol presented in Section 5. In particular, rules i_1 to i_3 are the decomposition rules, whereas rules i_4 to i_6 are the composition rules. The intruder can decrypt messages addressed to himself/herself, whereas messages encrypted with other keys can only be stored as they are. The composition rules allow the intruder to replay encrypted messages which have been previously stored, and to compose messages from plain components and encrypt them with an arbitrary key.

The specification of unsafe states is as follows:

$u_1)$ $pr(alice, step3(NA, NB, bob)) \parr m(NA) \multimap \top$
$u_2)$ $pr(alice, step3(NA, NB, bob)) \parr m(NB) \multimap \top$
$u_3)$ $pr(bob, step4(NA, NB, alice)) \parr m(NA) \multimap \top$
$u_4)$ $pr(bob, step4(NA, NB, alice)) \parr m(NB) \multimap \top$

That is, 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 (1995), Needham–Schroeder protocol is not safe with respect to the above security properties. In fact, we find the attack shown in Fig. 10, corresponding to the one presented in Lowe (1995) (we have followed the same conventions as in Fig. 7). With the usual protocol notation, the attack is as follows:

$$
\begin{array}{lll}
1. & A \to I & : \{N_a, A\}_{K_i} \\
1'. & (A) \to B & : \{N_a, A\}_{K_b} \\
2'. & B \to (A) & : \{N_a, N_b\}_{K_a} \\
2. & I \to A & : \{N_a, N_b\}_{K_a} \\
3. & A \to I & : \{N_b\}_{K_i} \\
3'. & (A) \to B & : \{N_b\}_{K_b}
\end{array}
$$

The attack takes place because *alice* decides to contact the intruder, *without knowing he/she is cheating*. Thus, the intruder is able to impersonate *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 that *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 that she has 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*):

$$
\begin{array}{ll}
u'_1) & pr(alice, step3(NA, NB, bob)) \parr pr(bob, step4(NA, NB, alice)) \parr m(NA) \circ\!\!- \top \\
u'_2) & pr(alice, step3(NA, NB, bob)) \parr pr(bob, step4(NA, NB, alice)) \parr m(NB) \circ\!\!- \top
\end{array}
$$

That is, we try to ascertain whether it is possible that two honest principals *alice* and *bob* *both* believe that they 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 without finding any attack, thus 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 be used to speed up the fixpoint computation, and consists in enriching the set of unsafe states with the negation of other invariants, which, intuitively, encode further configurations which are assumed not to be reachable from the initial configuration. A careful choice of the invariants to be added can speed up the fixpoint computation dramatically (see experimental results in Fig. 14). The underlying intuition is that, using the added invariants, the fixpoint computation may be able to generate some consequences of the initial theory, which may help in pruning (via *subsumption*; see Proposition 7.8) the current set of generated consequences (the net effect would be that the intermediate steps generate fewer consequences, and the final fixpoint is smaller). Even in the case in which the added invariant is already a consequence of the initial

theory, the addition of the invariant may speed up the computation in that it may generate consequences of the initial theory much earlier (in this case, the net effect would be that the intermediate steps generate fewer consequences, even though the final fixpoint is the same). We remark that the power of invariant strengthening is due to the combined use of *subsumption*.

In the case of the analysis of the Needham–Schroeder protocol, we can augment the set of security violations with this further axiom:

$$u'_3) \quad pr(alice, step1(NA, bob)) \parr m(NA) \circ\!\!- \top$$

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 Fig. 14). Note that the following invariant is instead violated (the attack follows the same scheme of the one in Fig. 10):

$$u'_4) \quad pr(bob, step2(NA, NB, alice)) \parr m(NB) \circ\!\!- \top$$

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. The corrected Needham–Schroeder protocol

As observed by 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

1. $A \to B \ : \ \{N_a, A\}_{K_b}$
2. $B \to A \ : \ \{B, N_a, N_b\}_{K_a}$
3. $A \to B \ : \ \{N_b\}_{K_b}$

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:

$$p'_2) \quad pr(B, init) \parr n(pubk(B), plain(NA, A)) \circ\!\!- \forall NB.(pr(B, step2(NA, NB, A)) \parr n(pubk(A), plain(B, NA, NB)))$$

$$p'_3) \quad pr(A, step1(NA, B)) \parr n(pubk(A), plain(B, NA, NB)) \circ\!\!- pr(A, step3(NA, NB, B)) \parr n(pubk(B), plain(NB))$$

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

$$i_7) \quad n(enc(pubk(intruder), plain(X, Y, Z))) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(plain(Z))$$

$$i_8) \quad m(plain(X)) \parr m(plain(Y)) \parr m(plain(U)) \circ\!\!- m(plain(X)) \parr m(plain(Y)) \parr m(plain(U)) \parr n(enc(pubk(Z), plain(X, Y, U)))$$

We can now use our algorithm to automatically verify whether the protocol satisfies *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 Fig. 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, proving that the modified version of the Needham–Schroeder protocol is correct with respect to 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 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 the two messages match, and, if they do, 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 Fig. 11. Let us discuss them in more detail. The encoding is similar to the previous ones, the only difference being that we use a concatenation operator *cons* to glue together different components inside the plaintext or the ciphertext. We have decided to use both the constructors *plain* and *cons* in order to be consistent with the previous specifications, although in principle using only one constructor might be sufficient. This example shows how it is possible to capture some kinds of type flaw attacks. We use an identifier s for the trusted server, and the notation $sk(id, s)$ to denote the key shared between principal id and s. Rules p_1 to 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 such as $step_i$. When the protocol is run to completion, the internal state of each of the two principals involved contains the identity of the other principal and the new shared key obtained from the server. The intruder theory is presented in Fig. 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 re-assemble

plain components. Finally, rules i_5 and i_6 are the usual composition rules. For simplicity, we do not present the intruder rules for decrypting messages addressed to himself/herself, and for encrypting messages with an arbitrary key (these rules can be formalized as was done in the previous encodings).

We wish to verify whether 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:

$$u_1)\ \ pr(bob, step3(alice, KAB)) \parr m(plain(KAB)) \circ\!\!- \top$$
$$u_2)\ \ pr(alice, step4(bob, KAB)) \parr m(plain(KAB)) \circ\!\!- \top$$

Running our verification tool, we automatically find the type flaw attack described in Clark and Jacob (1997). The corresponding trace is shown in Fig. 13. The attack in the usual protocol notation is as follows:

$$1.\ \ A \to (B)\ \ :\ \ N, A, B, \{N_a, N, A, B\}_{K_{as}}$$
$$4.\ \ (B) \to A\ \ :\ \ N, \{N_a, N, A, B\}_{K_{as}}$$

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. Such attacks are very pervasive in authentication. Another classical example of a type flaw 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 III 700 MHz under Linux RedHat 7.1, running Standard ML of New Jersey, Version 110.0.7. The tag **Invar** indicates the use of *invariant strengthening* for accelerating fixpoint convergence. Furthermore, **Steps** denotes the number of steps performed, **Size** the number of multisets inferred at the last step of the computation (either before finding an attack or before reaching the fixpoint), **MSize** is the maximal number of multisets inferred at any intermediate step, and **Time** is the execution time in seconds. In the case of the Otway–Rees protocol, the level of term nesting has been limited, in order to speed up the computation.

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 that we propose can be effective either for finding *attacks* of given protocols, or for proving that no attacks may exist (clearly, with respect to 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: if no proof is found (with respect 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 automize the specification phase of protocols and of the intruder theory. As regards the specification of protocols, we want to study a (possibly automatic) translation between the usual informal description and our representation. As shown in the paper, a one-to-one translation (one rule for every step) could be enough, provided that we have a way to store the information about the internal state of principals. As regards the specification of the intruder theory, in Section 4 we mentioned the fact that it is safe to limit the capabilities of the intruder to the generation of messages having a shape that the honest principals can recognize. In fact, in this paper we have defined intruder theories specific to every example by specializing the generic rules of Cervesato et al. (1999) to the message patterns of the different protocols. As part of our future work, we would like to investigate this point more formally. In particular, following Lowe (1999), we would like to have some formal results guaranteeing, under suitable hypotheses, that such a restriction does not rule out attacks. Furthermore, we would like to automize the generation of the specialized intruder theory from the corresponding protocol specification. For efficiency reasons, it could also be worth investigating some optimizations to the intruder theory (concerning, e.g., the rules for composition and decomposition of messages). We plan to use techniques such as *folding/unfolding* for this purpose.

Finally, there is a lot of potential for optimizing the ML prototype used for the experiments reported in Section 8. We believe that the tool can be engineered in such a way as to be competitive (in terms of efficiency and flexibility) with existing tools. Possible directions of improvement in order to optimize the performance include: using sophisticated term management techniques, improving the subsumption checking function, studying heuristics to prune and/or direct the search, and investigating the automatic generation of invariants to speed up convergence of the fixpoint computation.

Another topic that 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 that we have presented, our algorithm is always terminating, even without invariant strengthening (the only exception being the Otway–Rees protocol, for which the analysis is terminating—because an attack is eventually found—but the fixpoint computation does not converge, due to generation of terms with arbitrary nesting). 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 whether a more restricted subclass of protocols exists, for which the verification algorithm presented here is guaranteed to terminate.

Acknowledgements

We would like to thank the anonymous reviewers of this paper for their helpful suggestions and advice.

References

Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K., 2000. Algorithmic analysis of programs with well quasi-ordered domains. Information and Computation 160 (1–2), 109–127.

Amadio, R., Lugiez, D., 2000. On the reachability problem in cryptographic protocols. In: Palamidessi, C. (Ed.), Proceedings 11th International Conference on Concurrency Theory, CONCUR'00, University Park, Pennsylvania. LNCS, vol. 1877. Springer-Verlag, pp. 380–394.

Andreoli, J.-M., 1992. Logic programming with focusing proofs in linear logic. Journal of Logic and Computation 2 (3), 297–347.

Andreoli, J.-M., Pareschi, R., 1991. Linear objects: logical processes with built-in inheritance. New Generation Computing 9 (3–4), 445–473.

Armando, A., Compagna, L., 2003. Abstraction-driven SAT-based analysis of security protocols. In: Giunchiglia, E., Tacchella, A. (Eds.), Proceedings 6th International Conference on Theory and Applications of Satisfiability Testing, SAT'2003, S. Margherita Ligure, Italy. LNCS, vol. 2919. Springer-Verlag, pp. 257–271.

Armando, A., Compagna, L., Ganty, P., 2003. SAT-based model-checking of security protocols using planning graph analysis. In: Araki, K., Gnesi, S., Mandrioli, D. (Eds.), Proceedings 12th International FME Symposium, FM'2003, Pisa, Italy. LNCS, vol. 2805. Springer-Verlag, pp. 875–893.

Basin, D., 1999. Lazy infinite-state analysis of security protocols. In: Baumgart, R. (Ed.), Proceedings International Exhibition and Congress on Secure Networking, Cqre Secure'99, Dusseldorf, Germany. LNCS, vol. 1740. Springer-Verlag, pp. 30–42.

Blanchet, B., 2001. An efficient cryptographic protocol verifier based on Prolog rules. In: Proceedings 14th Computer Security Foundations Workshop, Cape Breton, Nova Scotia, Canada. IEEE Computer Society Press, pp. 82–96.

Boreale, M., 2001. Symbolic trace analysis of cryptographic protocols. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (Eds.), Proceedings 28th International Colloquium on Automata, Languages and Programming, ICALP'01, Heraklion, Crete, Greece. LNCS, vol. 2076. Springer-Verlag, pp. 667–681.

Bozzano, M., 2001. 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.

Bozzano, M., 2002. A logic-based approach to model checking of parameterized and infinite-state systems. Ph.D. Thesis, Dipartimento di Informatica e Scienze dell' Informazione, Università di Genova.

Bozzano, M., Delzanno, G., 2002. Automated protocol verification in linear logic. In: Proceedings 4th International Conference on Principles and Practice of Declarative Programming, PPDP'02, Pittsburgh, Pennsylvania. ACM Press, pp. 38–49.

Bozzano, M., Delzanno, G., Martelli, M., 2000. A bottom-up semantics for linear logic programs. In: Proceedings 2nd International Conference on Principles and Practice of Declarative Programming, PPDP'00, Montreal, Canada. ACM Press, pp. 92–102.

Bozzano, M., Delzanno, G., Martelli, M., 2002. An effective fixpoint semantics for linear logic programs. Theory and Practice of Logic Programming 2 (1), 85–122.

Bozzano, M., Delzanno, G., Martelli, M., 2004. Model checking linear logic specifications. Theory and Practice of Logic Programming 4 (5), 1–47.

Burrows, M., Abadi, M., Needham, R., 1989. A logic of authentication. Technical Report 39, Digital Equipment Corporation Systems Research Center.

Cervesato, I., 1995. Petri nets and linear logic: a case study for logic programming. In: Alpuente, M., Sessa, M.I. (Eds.), Proceedings Joint Conference on Declarative Programming, GULP-PRODE'95, Marina di Vietri, Italy. Palladio Press, pp. 313–318.

Cervesato, I., 2001a. The Dolev–Yao intruder is the most powerful attacker. In: Halpern, J. (Ed.), Proceedings 16th Annual International Symposium on Logic in Computer Science, LICS'01, Boston, Massachusetts (Short paper).

Cervesato, I., 2001b. Typed multiset rewriting specifications of security protocols. In: Seda, A. (Ed.), Proceedings 1st Irish Conference on the Mathematical Foundations of Computer Science and Information Technology, MFCSIT'00, Cork, Ireland.

Cervesato, I., Durgin, N.A., Lincoln, P.D., Mitchell, J.C., Scedrov, A., 1999. A meta-notation for protocol analysis. In: Proceedings 12th Computer Security Foundations Workshop, CSFW'99, Mordano, Italy. IEEE Computer Society Press, pp. 55–69.

Chevalier, Y., Vigneron, L., 2002. Automated unbounded verification of security protocols. In: Brinksma, E., Larsen, K.G. (Eds.), Proceedings 14th International Conference on Computer Aided Verification, CAV'02, Copenhagen, Denmark. LNCS, vol. 2404. Springer-Verlag, pp. 324–337.

Cirstea, H., 2001. Specifying authentication protocols using rewriting and strategies. In: Ramakrishnan, I.V. (Ed.), Proceedings of Conference on Practical Aspects of Declarative Languages, PADL'01, Las Vegas, Nevada. LNCS, vol. 1990. Springer-Verlag, pp. 138–152.

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

Cohen, E., 2000. TAPS: A first-order verifier for cryptographic protocols. In: Proceedings 13th Computer Security Foundations Workshop, CSFW'00, Cambridge, England. IEEE Computer Society Press, pp. 144–158.

Delzanno, G., 2001. Specifying and debugging security protocols via hereditary Harrop formulas and λProlog—A case-study. In: Kuchen, H., Ueda, K. (Eds.), Proceedings 5th International Symposium on Functional and Logic Programming, FLOPS'01, Tokyo, Japan. LNCS, vol. 2024. Springer-Verlag, pp. 123–137.

Denker, G., Meseguer, J., Talcott, C., 1998. Protocol specification and analysis in Maude. In: Heintze, N., Wing, J. (Eds.), Proceedings Workshop on Formal Methods and Security Protocols, Indianapolis, Indiana.

Dolev, D., Yao, A., 1983. On the security of public-key protocols. IEEE Transactions on Information Theory 2 (29).

Fages, F., Ruet, P., Soliman, S., 2001. Linear concurrent constraint programming: operational and phase semantics. Information and Computation 165 (1), 14–41.

Genet, T., Klay, F., 2000. Rewriting for cryptographic protocol verification. In: McAllester, D.A. (Ed.), Proceedings 17th International Conference on Automated Deduction, CADE-17. LNCS, vol. 1831. Springer-Verlag, pp. 271–290.

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

Jacquemard, F., Rusinowitch, M., Vigneron, L., 2000. Compiling and verifying security protocols. In: Parigot, M., Voronkov, A. (Eds.), Proceedings 7th International Conference on Logic for Programming and Automated Reasoning, LPAR'00, Reunion Island, France. LNCS, vol. 1955. Springer-Verlag, pp. 131–160.

Kobayashi, N., Yonezawa, A., 1995. Asynchronous communication model based on linear logic. Formal Aspects of Computing 7 (2), 113–149.

Lowe, G., 1995. An attack on the Needham–Schroeder public-key authentication protocol. Information Processing Letters 56, 131–133.

Lowe, G., 1996. Breaking and fixing the Needham–Schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (Eds.), Proceedings 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems, TACAS'96, Passau, Germany. LNCS, vol. 1055. Springer-Verlag, pp. 147–166.

Lowe, G., 1998. Casper: a compiler for the analysis of security protocols. Journal of Computer Security 6 (1), 53–84.

Lowe, G., 1999. Towards a completeness result for model checking of security protocols. Journal of Computer Security 7 (2–3), 89–146.

Marrero, W., Clarke, E., Jha, S., 1997. Model checking for security protocols. Technical Report CMU-CS-97-139, School of Computer Science, Carnegie Mellon University.

McDowell, R., Miller, D., Palamidessi, C., 1996. Encoding transition systems in sequent calculus. In: Girard, J.-Y., Okada, M., Scedrov, A. (Eds.), Proceedings Linear Logic 96 Tokyo Meeting, Keio University, Tokyo, Japan. ENTCS, vol. 3.

Meadows, C., 1996. The NRL protocol analyzer: an overview. Journal of Logic Programming 26 (2), 113–131.

Millen, J.K., 1999. A necessarily parallel attack. In: Proceedings Workshop on Formal Methods and Security Protocols, Trento, Italy.

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

Millen, J.K., Shmatikov, V., 2001. Constraint solving for bounded-process cryptographic protocol analysis. In: Proceedings Conference on Computer and Communication Security, CCS'01, Philadelphia, Pennsylvania. ACM Press, pp. 166–175.

Miller, D., 1992. The π-calculus as a theory in linear logic: preliminary results. In: Lamma, E., Mello, P. (Eds.), Proceedings Workshop on Extensions of Logic Programming, Bologna, Italy. LNCS, vol. 660. Springer-Verlag, pp. 242–265.

Miller, D., 1996. Forum: a multiple-conclusion specification logic. Theoretical Computer Science 165 (1), 201–232.

Miller, D., Nadathur, G., Pfenning, F., Scedrov, A., 1991. Uniform proofs as a foundation for logic programming. Annals of Pure and Applied Logic 51, 125–157.

Needham, R., Schroeder, M., 1978. Using encryption for authentication in large networks of computers. Communications of the ACM 21 (12), 993–999.

Otway, D., Rees, O., 1987. Efficient and timely mutual authentication. Operating Systems Review 21 (1), 8–10.

Paulson, L.C., 1998. The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6 (1–2), 85–128.

Roscoe, A.W., Broadfoot, P.J., 1999. Proving security protocols with model checkers by data independence techniques. Journal of Computer Security 7 (2–3), 147–190.

Rusinowitch, M., Turuani, M., 2001. Protocol insecurity with finite number of sessions is NP-complete. In: Proceedings 14th Computer Security Foundations Workshop, CSFW'01, Cape Breton, Nova Scotia, Canada. IEEE Computer Society Press.

Song, D.X., 1999. A new efficient automatic checker for security protocol analysis. In: Proceedings 12th Computer Security Foundations Workshop, CSFW'99, Mordano, Italy. IEEE Computer Society Press, pp. 192–202.

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