The 2007 Federated Conference on

Rewriting, Deduction and Programming

Paris, France

June 25 – 29, 2007

SecReT'07

Workshop on Security and Rewriting Techniques

June 29th, 2007

Proceedings

Editors:

Monica Nesi and Ralf Treinen

Preface

This volume contains the papers presented at the second International Workshop on Security and Rewriting Techniques (SecReT'07) which was held on June 29, 2007, in Paris as part of the fourth Federated Conference on Rewriting, Deduction, and Programming (RDP'07). The first SecReT workshop had been held in Venice, Italy, in 2006 as part of the ICALP conference.

The aim of the SecReT workshop is to bring together rewriting researchers and security experts, in order to foster their interaction and develop future collaborations in this area, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in this area.

The workshop focuses on the use of rewriting techniques in all aspects of security. Specific topics include: authentication, encryption, access control and authorization, protocol verification, specification of policies, intrusion detection, integrity of information, control of information leakage, control of distributed and mobile code, etc.

We would like to thank the program committee as well as the additional referees for their work. Thomas Genet gracefully accepted to give an invited talk on *Rewriting and Reachability for Software Security.* Last but not least we would like to thank the institutional sponsors of RDP'07 without whom it would not have been possible to organize RDP'07: the Conservatoire des Arts et Métiers (CNAM), the Centre National de la Recherche Scientifique (CNRS), the École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise (ENSIEE), the GDR Informatique Mathématique, the Institut National de Recherche en Informatique et Automatique (INRIA) unit Futurs, and the Région Île de France.

L'Aquila and Cachan, June 2007

Monica Nesi and Ralf Treinen
SecReT'07 workshop chairs

Program Commitee

Véronique Cortier	Nancy, France
Maribel Fernández	London, UK
Paliath Narendran	Albany, NY, USA
Monica Nesi	L'Aquila, Italy, co-chair
Tobias Nipkow	München, Germany
Hitoshi Ohsaki	Osaka, Japan
Graham Steel	Edinburgh, UK
Mark-Oliver Stehr	Stanford, CA, USA
Ralf Treinen	Cachan, France, co-chair
Luca Viganò	Verona, Italy

Table of Contents

Rewriting and Reachability for Software Security

Thomas Genet

IRISA Rennes

This talk is about using reachability analysis for proving security properties on software modeled by Term Rewriting Systems (TRS in short). Reachability analysis consists in proving that a set of final terms can, or cannot, be reached by rewriting a set of initial terms. For instance, unreachability can be used to prove that a software system *cannot* evolve from a state modeled by an initial term to a forbidden state modeled by a final term. In the field of verification, this is generally known as *safety properties*. Similarly, many interesting *security properties* are related to safety and can thus be proved using reachability analysis on TRS.

When the TRS modeling the software does not terminate or too big to be handled by usual proof tools of rewriting, tree automata techniques can provide simple and efficient proofs for (un)reachability properties. The principle is generally to compute or to approximate the set of reachable terms using tree automata, and then check that final (forbidden) terms do not belong to this set. In this talk, I will first sum up some of the results concerning the exact computation of reachable terms using tree automata. Then, I will focus on the approximated case and explain the choices we made in the Timbuk reachability analysis tool. Next, I will give some indications on what can be proved using regular approximations and ways to define them. This will be illustrated on two applications we have on cryptographic protocols and Java applets verification. Finally, I will present some research orientations in reachability analysis on TRS.

Action-Status Access Control as Term Rewriting

Steve Barker and Maribel Fernández

King's College London, Dept. of Computer Science, London WC2R 2LS, U.K.
{Steve.Barker, Maribel.Fernandez}@kcl.ac.uk

Abstract. We propose an access control model that generalizes Role-Based Access Control by making a distinction between what we call ascribed status and action status. The model is based upon the key notion of an event to enable changes in access control requirements to be performed autonomously. Our access control model is specified as a term rewriting system that permits declarative representation of access control requirements, proving of properties of access control policies defined in terms of our model, and fast prototyping of access control checking.

1 Introduction

We describe a form of access control model that is based on an interpretation of ideas from *Role-based Access Control* (RBAC, see [28]). This interpretation, we will argue, is of particular relevance in situations where access control requirements change in a highly dynamic way, and where the actions of users, that request access to protected resources, are important in rendering a decision to allow the requested access or not.

To address the requirements of access policy representation for dynamic, distributed information systems, we propose a variant of the *Action Status Access Control (ASAC)* model [7]. In the *ASAC* model, a user (a human user or software user) is viewed as a *rational* entity that can, within certain constraints, choose the *actions* it performs in order to increase its access to resources by changing its *action status*. A user's action status relates to a status a user may *achieve* by acting in a way that warrants the assignment of the user to a particular status level. The assignment of a user to a role, as it is usually understood in the RBAC context, is viewed by us as a particular form of *ascribed* status. An ascribed status is a status that is associated with a particular role, a categorization of users (e.g., by attributes like *age*), an office holder, ... A user's overall status is a user's standing relative to other users, and is determined by considering the user's ascribed status (if any) or action status (if any). To model user actions, we make use of the primitive notion of an *event*: a happening at an instance of time that involves a user performing an action. The notion of an event provides, as we will show, a basis for allowing *ASAC* policies to change dynamically in response to the occurrence of events. An access control model that takes user actions into account and that allows users to manage, to some extent, their access to resources will be increasingly important in distributed applications, like secure e-trading and e-contracting, where flexible access specifications are essential and where notions like job functions and roles are of peripheral significance.

In this paper we define the *ASAC* model as a *term rewrite system* [17,24,4]. The *ASAC* model that we propose, and its representation using term rewriting, contributes to the literature on formal access control models by demonstrating how access control models may be defined that permit the autonomous changing of access control policies, the proving of properties of policies, and the evaluation of access requests when the sources of access control and user requested information may be widely dispersed. Due to space constraints, in this paper we restrict our attention to conveying the key elements of the *ASAC* model and their representation in the form of a term rewriting system. The rewrite systems we will define give a formal specification, and an operational semantics, for *ASAC* policies. They can also be seen as a prototype implementation (since they can be directly executed); efficiency issues will not be addressed in this paper.

The remainder of the paper is organized in the following way. In Section 2, we recall some basic notions in term rewriting and access control models, to make the paper self-contained. In Section 3, we describe the main features of the *ASAC* model, and in Section 4 we define *ASAC* policies as term rewriting systems. In Section 5, we discuss related literature more fully. In Section 6, conclusions are drawn, and further work suggested.

2 Preliminaries

2.1 Term Rewriting

In this section we recall some basic notions and notations for first-order term rewriting. We refer the reader to [4] for additional information.

A *signature* $\mathcal{F}$ is a finite set of *function symbols* together with their (fixed) arity. $\mathcal{X}$ denotes a denumerable set of *variables* $X_1, X_2, \ldots$, and $T(\mathcal{F}, \mathcal{X})$ denotes the set of *terms* built up from $\mathcal{F}$ and $\mathcal{X}$.

Terms are identified with finite labeled trees. The symbol at the root of t is denoted by $root(t)$. *Positions* are strings of positive integers. The *subterm* of t at position p is denoted by $t|_p$ and the result of replacing $t|_p$ with u at position p in t is denoted by $t[u]_p$.

$\mathcal{V}(t)$ denotes the set of variables occurring in t. A term is *linear* if variables in $\mathcal{V}(t)$ occur at most once in t. A term is *ground* if $\mathcal{V}(t) = \emptyset$. Substitutions are written as in $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where t_i is assumed to be different from the variable X_i. We use Greek letters for substitutions and postfix notation for their application. We say that two terms unify if there is some substitution that makes them equal. Such a substitution is called a *unifier*. The *most general unifier* (mgu) is the unifier that will yield instances in the most general form.

Definition 1. *Given a signature $\mathcal{F}$, a* term rewriting system *on $\mathcal{F}$ is a set of rewrite rules $R = \{l_i \to r_i\}_{i \in I}$, where $l_i, r_i \in T(\mathcal{F}, \mathcal{X})$, $l_i \notin \mathcal{X}$, and $\mathcal{V}(r_i) \subseteq \mathcal{V}(l_i)$. A term t* rewrites *to a term u at position p with the rule $l \to r$ and the substitution σ, written $t \to_p^{l \to r} u$, or simply $t \to_R u$, if $t|_p = l\sigma$ and $u = t[r\sigma]_p$. Such a term t is called* reducible*. Irreducible terms are said to be in* normal form*.*

We denote by $\to_R^+$ (resp. $\to_R^*$) the transitive (resp. transitive and reflexive) closure of the rewrite relation $\to_R$. The subindex R will be omitted when it is clear from the context.

Example 1. Consider a signature for lists of natural numbers, with function symbols:

– z (with arity 0) and s (with arity 1, denoting the successor function) to build numbers;
– nil (with arity 0) to denote an empty list, cons (with arity 2) to construct non-empty lists, and append (with arity 2) to concatenate lists.

The list containing the numbers 0 and 1 is written: $\mathsf{cons}(\mathsf{z}, \mathsf{cons}(\mathsf{s}(\mathsf{z}), \mathsf{nil}))$, or simply $[\mathsf{z}, \mathsf{s}(\mathsf{z})]$ for short. We can specify list concatenation with the following rewrite rules:

$$\begin{aligned} \mathsf{append}(\mathsf{nil}, x) &\to x \\ \mathsf{append}(\mathsf{cons}(y, x), z) &\to \mathsf{cons}(y, \mathsf{append}(x, z)) \end{aligned}$$

Then we have a reduction sequence:

$$\mathsf{append}(\mathsf{cons}(\mathsf{Z}, \mathsf{nil}), \mathsf{cons}(\mathsf{S}(\mathsf{Z}), \mathsf{nil})) \to^* \mathsf{cons}(\mathsf{Z}, \mathsf{cons}(\mathsf{S}(\mathsf{Z}), \mathsf{nil}))$$

Let $l \to r$ and $s \to t$ be two rewrite rules (we assume that the variables of $s \to t$ were renamed so that there is no common variable with $l \to r$), p the position of a non-variable subterm of s, and μ a most general unifier of $s|_p$ and l. Then $(t\mu, s\mu[r\mu]_p)$ is a *critical pair* formed from those rules. Note that $s \to t$ may be a renamed version of $l \to r$. In this case a superposition at the root position is not considered a critical pair.

A term rewriting system R is:

– *confluent* if for all terms t, u, v: $t \to^* u$ and $t \to^* v$ implies $u \to^* s$ and $v \to^* s$, for some s;
– *terminating* (or *strongly normalizing*) if all reduction sequences are finite;
– *left-linear* if all left-hand sides of rules in R are linear;
– *non-overlapping* if there are no critical pairs;
– *orthogonal* if R is left-linear and non-overlapping;
– *non-duplicating* if for all $l \to r \in R$ and $X \in \mathcal{V}(l)$, the number of occurrences of X in r is less than or equal to the number of occurrences of X in l.

For example, the rewrite system in Example 1 is confluent, terminating, left-linear and non-overlapping (therefore orthogonal), and non-duplicating.

A *hierarchical union* of rewrite systems consists of a set of rules defining some basic functions (this is called the *basis* of the hierarchy) and a series of *enrichments*. Each enrichment defines a new function or functions, using the ones previously defined. Constructors may be shared between the basis and the enrichments.

We recall a modularity result for termination of hierarchical unions from [20] (Theorem 14), which will be useful later:

If in a hierarchical union the basis is non-duplicating and terminating, and each enrichment satisfies a general scheme of recursion, where each recursive call in the right-hand side of a rule uses subterms of the left-hand side, then the hierarchical union is terminating.

2.2 Role Based Access Control Models

In simple terms, the fundamental idea of RBAC is that:

– a user u of a resource o may be assigned to a set of roles $\{r_1, \ldots, r_n\}$ (usually as a consequence of the user performing a job function in an organisation e.g., *doctor*, *CEO*, etc.);
– access privileges on resources are also assigned to roles;
– a user u may exercise an access privilege p on a resource o if and only if u is assigned to a role r to which the privilege p on o is also assigned.

It follows, from the discussion above, that RBAC models/policies are specified with respect to a domain of discourse that includes the sets $\mathcal{U}$ of users, $\mathcal{O}$ of objects, and $\mathcal{P}$ of access privileges, together with a (finite) set $\mathcal{R}$ of *roles.*

The capability of assigning users to roles and permissions (i.e., access privilege assignments on objects) to roles are primitive requirements of all RBAC models. The most basic category of RBAC model, flat RBAC [29] (or $RBAC_F$ for short), requires that these types of assignment are supported. The $RBAC_{H2A}$ model extends $RBAC_F$ to include the notion of an RBAC role hierarchy (see below) in addition to user-role and permission-role assignments. The flat RBAC and $RBAC_{H2A}$ models from [29] are referred to, respectively, as $RBAC_F$ as $RBAC_{H2A}$ logic theories in the formal representation of RBAC models in [9]. In the remainder of the paper, we will refer to $RBAC$ theories rather than models.

In the $RBAC_{H2A}$ theory, the semantics of user-role assignment may be defined in terms of a 2-place *ura* predicate (where *ura* is short for "user role assignment") and permission-role assignment can be defined in terms of a 3-place predicate *pra* (where *pra* is short for "permission role assignment"). The extensions of these predicates define role and permission assignments in a world of interest.

Definition 2. *Let Π be an $RBAC_{H2A}$ theory. Then,*

– $\Pi \models ura(u, r)$ *if and only if user* $u \in \mathcal{U}$ *is assigned to role* $r \in \mathcal{R}$*;*
– $\Pi \models pra(a, o, r)$ *if and only if the access privilege* $a \in \mathcal{A}$ *on object* $o \in \mathcal{O}$ *is assigned to the role* $r \in \mathcal{R}$*.*

An $RBAC_{H2A}$ *role hierarchy* is defined as a (partially) ordered (and finite) set of roles. The ordering relation is a role seniority relation. A 2-place predicate $senior\_to(r_i, r_j)$ is used to define the seniority ordering between pairs of roles i.e., the role $r_i \in \mathcal{R}$ is a more senior role (or more powerful role) than role $r_j \in \mathcal{R}$. If r_i is senior to r_j then any user assigned to the role r_i has at least the permissions that users assigned to role r_j have. Role hierarchies are important for specifying implicitly the inheritance of access privileges on resources.

Example 2. Suppose that the users u_1 and u_2 are assigned to the roles r_2 and r_1 respectively, and that write (w) permission on object o_1 is assigned to r_1 and read (r) permission on o_1 is assigned to r_2. Moreover, suppose that r_1 is senior to r_2 in an $RBAC_{H2A}$ role hierarchy. Then, using the notation introduced above, this $RBAC_{H2A}$ policy is represented by the relations:

$$ura(u_1, r_2), ura(u_2, r_1), pra(w, o_1, r_1), pra(r, o_1, r_2), senior\_to(r_1, r_2).$$

User-role and permission-role assignments are related via the notion of an *authorisation.* An authorisation is a triple (u, a, o) that expresses that the user u has the a access privilege on the object o. Given an $RBAC_{H2A}$ theory Π, the set of authorisations $\mathcal{AUTH}$ defined by Π may be expressed thus:

$$(u, a, o) \in \mathcal{AUTH} \Leftrightarrow \exists r_1, r_2.ura(u, r_1) \ \wedge \ senior\_to(r_1, r_2) \ \wedge \ pra(a, o, r_2)$$

According to the definition of the set $\mathcal{AUTH}$ above, a user u may exercise the a access privilege on object o if:

> *u is assigned to the role r_1, r_1 is senior to a role r_2 in an $RBAC_{H2A}$ role hierarchy, and r_2 has been assigned the a access privilege on o.*

Example 3. By inspection of the user-role assignments, permission-role assignments, and the role seniority relationships that are specified in Example 2, it follows that the set of authorisations that are included in $\mathcal{AUTH}$ is:

$$\{(u_2, w, o_1), (u_2, r, o_1), (u_1, r, o_1)\}.$$

3 The ASAC Model

A key feature of the *ASAC* model [7] is that a decision on an agent's request to access resources is determined by considering the agent's *ascribed status*, the agent's *action status*, and any additional conditions of relevance in answering the access request. An agent's ascribed status together with the agent's action status gives a measure of the agent's overall *status level.* The agent's status level is used as the basis for determining authorized actions and thus is used in rendering a decision on the agent's access request.

An ascribed status is any grouping of agents by some common criterion that agents of the group share. An ascribed status may be associated with, for instance: a particular role, a classification of trustworthiness, membership of an organization, etc. Although notions like roles, attributes, security classifications, and class membership have been used as a basis for access control models that have hitherto been proposed, we view these notions as instances of the more general notion of ascribed status.

The *ASAC* model also incorporates the additional concept of an agent's action status. An agent's action status is determined from a history of the *deliberative* actions performed by the agent. In *ASAC*, an agent is viewed as a *rational* entity that can, within certain constraints, choose the *actions* it performs. As access to resources is determined from an agent's status level it immediately follows that, in *ASAC*, an agent can (to some extent) determine its access to resources because the agent chooses the actions it performs and thus its action status.

It should be noted that an ascribed status may, of course, require an agent to have performed some action (e.g., to be assigned to a role of *manager* an agent may have had to apply to become a *manager*). However, the specific actions that result in the assignment of an ascribed status are not necessarily of significance in determining an agent's status level; *what* ascribed status an agent has rather than *how* the ascribed status was assigned may be the only criterion of interest in making a decision on access. Notice too that we will henceforth use the term status level to mean an agent's overall status that is determined from the agent's ascribed status (if any) or action status (if any); we will only refer to ascribed status and action status specifically when it is important to make the distinction.

To understand the difference between ascribed status and action status, consider an example of a human agent that is under eighteen years of age. The agent's age is the basis for an ascribed status *minor*, a classification that is not a consequence of any deliberative act performed by the agent. Now, suppose that the agent performs the deliberative action of stealing a car and assume that no mitigating circumstances apply to excuse the act of stealing. Then, the ascribed status together with the action might give the agent an overall status of *young offender*. In turn, the status of *young offender* may be used as a basis for determining what actions an agent with *young offender* status can and cannot do. For example, agents with the status of *young offender* may be subject to a curfew order. In a similar way, in $ASAC$ the ascribed status and action status of an agent u, that requests to access a resource r, is used to determine what actions u can and cannot perform on r.

Access control models, like $ASAC$ that take agent actions into account and that allow agents to manage, to some extent, their access to resources will be increasingly important in distributed applications, like secure e-trading and e-contracting. In such cases, notions like job functions and roles are often of peripheral (or no) significance. Instead, criteria that are based on actions, such as the quantity of units ordered by a customer agent via acts of purchasing, may be used as the basis for determining access to resources e.g., customers ordering large volumes of merchandise may, from their history of purchases, be determined to have *preferred* status and, as a consequence, may be permitted to access *special offers* information that is not accessible to customers without *preferred* status. By addressing such requirements, the $ASAC$ model contributes to work by the access control community that aims to increase the functionality and application of access control models.

The $ASAC$ model was also developed to deal with changing access control requirements that can be effected autonomously, and to allow for agent determined access control policies i.e., by choosing what actions they wish to perform, agents can determine what policies apply to them. For example, by purchasing at least 1000 widgets a month, an agent may preserve its *preferred customer* status.

Definition 3. *An ASAC policy is a triple,*

$$(\mathcal{SED}, \mathcal{CORE}, \mathcal{AS}),$$

where:

- $\mathcal{SED}$ *is a Security Event Description – a history of events.*
- $\mathcal{CORE}$ *is the set of axioms defining the ASAC model.*
- $\mathcal{AS}$ *is the set of application-specific axioms that define an ASAC policy.*

A $\mathcal{SED}$ is a finite set of events describing a happening at a time involving an act by an agent, and optionally may include some additional elements. Events are represented by tuples (E, U, A, O, T) where E is an identifier, U is a user, A an action, O an object, and T a time point.

Example 4. Consider the following tuple:

$$(e_1, bob, depositing(1000), a_1, 20060612)$$

This describes an event e_1 that happens on 12th June 2006, and involves the agent *Bob* depositing an amount of 1000 Euros into an object (a bank account) denoted by a_1.

The axioms in $\mathcal{CORE}$ follow next. These axioms are included in the formulation of every *ASAC* policy, and specify a binary relation *sla* (status level assignment):

$$\begin{aligned} sla(U, L) \Leftrightarrow\ & \exists E, A, O, T_s.\ event(E, U, A, O, T_s) \in \mathcal{SED}\ and \\ & T_s < currentTime\ and\ sla\_init(E, U, A, O, L, T_s, currentTime) \\ & and\ not\ ended\_sla(U, L, T_s, currentTime). \end{aligned}$$

$$\begin{aligned} ended\_sla(U, L, T_s, T) \Leftrightarrow\ & \exists E', A, O, T'.\ event(E', U, A, O, T') \in \mathcal{SED}\ and \\ & T_s < T' \leqslant T\ and\ sla\_term(E', U, A, O, L, T_s, T). \end{aligned}$$

The definition of *sla* captures the following form of reasoning:

An agent U is currently assigned the status level L if:
1. *an event E happened at a time T_s, which is earlier than the current time T, and resulted in the initiation of U's assignment to L, and*
2. *this assignment has not been ended before T as a consequence of an event E' happening at a time T' between T_s and T that causes U's assignment of the status level L to be terminated.*

The definition of *ended_sla* expresses that:

The assignment of an agent U to a status level L is ended, in the interval $(T_s, T]$, if an event E' happens at time T', where T' is a time point in $(T_s, T]$, and E' causes the termination of U's assignment to L.

Application-specific $\mathcal{AS}$ axioms are included with $\mathcal{SED}$ and $\mathcal{CORE}$ to specify an *ASAC* policy by defining the *sla_init* and *sla_term* axioms, that represent the initiation of a user's status level assignment and its termination.

Example 5. The following clause specifies conditions on the initiation of the assignment of an agent to the status level l_1 as a consequence of an agent's act of depositing a certain amount X of money into a bank account identified by C:

$$\begin{aligned} sla\_init(E, U, depositing(X), C, l_1, T_s, T) \Leftrightarrow\ & X \geqslant 1000\ and\ balance(C) \geqslant 0 \\ & and\ sla(U, loyal)\ and \\ & not\ role(U, manager)\ and \\ & earlier\_by(T_s, T, 90). \end{aligned}$$

That is, an act of depositing a sum of at least 1000 Euros into account C within 3 months of the current-time T by an agent U, currently in credit, and such that U has the status of loyal customer and who does not have the ascribed status (i.e., role) of manager, is a sufficient condition for an agent to be assigned the status l_1.

In the context of access control, the notion of an *authorization* is of central importance. An authorization is typically defined as a triple (u, a, o) that is used to specify that user u is able to perform the action a on the object o. In the *ASAC* model, we admit a more general notion of authorization:

In the *ASAC* model, an authorization is a tuple (a, u, o, s, l), which specifies that in the context of a *history of events* l, a *user* u may perform the *action* a on *object* o located in a site s.

4 The *ASAC* Model as a Term Rewrite System

In this section, we illustrate the use of rewriting systems for specifying the core elements of the *ASAC* model. We do not claim that this is the only way to formalize the *ASAC* model as a rewrite system. Instead, our goals are to show how an executable[1] specification of an *ASAC* model may be represented, and to hint at the possibilities of using term rewriting for access request evaluation and for the proving of properties of *ASAC* policies when these policies are formally represented, using term rewriting.

In the *ASAC* model, a request from a user u to perform an action a on object o in site s will be granted or denied depending on the user's *status-level.*

In our representation of *ASAC* as a term rewrite system, we define authorizations in the following way, where the symbols U, A, O, S, L are variables, the operator member is the standard membership test operator, and the function level computes the status level of the user.

$$\begin{aligned} \mathsf{authorized}(A, U, O, S, L) &\to \mathsf{check}(\mathsf{member}((A, \mathsf{level}(U, L)), \mathsf{privileges}(O, S))) \\ \mathsf{check}(\mathsf{true}) &\to \mathsf{grant} \\ \mathsf{check}(\mathsf{false}) &\to \mathsf{deny} \end{aligned}$$

Here we assume that the function privileges returns a list of pairs *(action, status level of users allowed to perform the action)* for a given object at a given site. The privileges function is specified as part of the application-specific information that is included in a formulation of an *ASAC* policy.

For example, privileges could be defined extensionally by rules such as:

$$\mathsf{privileges}(o, s) \to [(a_{11}, l_{11}), \ldots, (a_{1n}, l_{1n})]$$

A user's status level is determined from the user's *ascribed status* and *action status*:

– Ascribed status: This is determined from ground terms or rules that define the categorization of user by whatever application-specific criteria is appropriate. For instance, $role(u) \to r$ may be used to specify the assignment of user u to the role r. Here, a role is a particular instance of the wider notion of ascribed status that is a central part of the *ASAC* model. More generally, the following rule may be used to specify the assignment of a user u to a set of roles $r_{11}, \ldots, r_{1i}$:

$$\mathsf{roles}(u) \to [r_{11}, \ldots, r_{1i}]$$

[1] For instance, the language MAUDE [16] can be used to execute rewrite-based specifications.

Any number of ascribed status assignments may be represented in the same way as user-role assignments. In general, we will assume that there is a function $\mathsf{ascribed}$ that returns, for each user, a list of ascribed status assignments.

$$\mathsf{ascribed}(u) \rightarrow [l_{11}, \ldots, l_{1i}]$$

– Action status: As we have said, events provide an homogeneous basis for representing *change*, a feature that is an essential aspect of our *ASAC* model. In *ASAC*, a user's action status is determined from the actions that are recorded as part of the event descriptions that relate to a user u. We view events as structured and described via a sequence h of ground terms of the form $\mathsf{event}(e_i, u, a, o, t)$ where event is a data constructor of arity five, e_i denotes a unique event identifier, u identifies a user, a is a user action on object o associated to the event, and t is the time at which the event happened. We assume that the list h of events is chronologically ordered. In this paper, we only admit atomic events. However, conjunctions, disjunctions, and sequences of events can also be naturally accommodated in our approach.
  Given an event e_i, we will use standard generic functions to extract the component information from an event description. For instance, we may define a function user that returns the user involved in a given event, as follows:

$$\mathsf{user}(\mathsf{event}(E, U, A, O, T)) \rightarrow U$$

The *ASAC* model is further defined in terms of *generic* and *specific* functions. The generic functions are level and status, together with the function $\mathsf{ascribed}$ and the auxiliary functions such as user (see above). The generic functions can be defined by the following rules:

$$\begin{aligned}
\mathsf{level}(U, H) &\rightarrow F(\mathsf{append}(\mathsf{status}(U, H), \mathsf{ascribed}(U)))\\
\mathsf{status}(U, \mathsf{nil}) &\rightarrow \mathsf{cons}(l_0, \mathsf{nil})\\
\mathsf{status}(U, \mathsf{cons}(E, H)) &\rightarrow \mathit{if}\ U = \mathsf{user}(E)\\
&\qquad \mathit{then}\ \mathsf{cons}(\mathsf{actionStatus}(E), \mathsf{filter}(E, \mathsf{status}(U, H)))\\
&\qquad \mathit{else}\ \mathsf{status}(U, H)
\end{aligned}$$

where append is the standard operator for list concatenation, l_0 is a default level, status looks for events involving a user U in the list H, and uses a *specific* function $\mathsf{actionStatus}$ that associates a level l_i to a user according to the particular event e_i in which the user was involved (this, of course, is specific to the application that we are modeling). The function F, which is used in the definition of level, takes as argument a list of status-levels associated to a user (both ascribed and action based), and returns one specific level. The particular definition of F depends on the application. For example, we can take F to be the functions max or min returning the highest or lowest level in the list, respectively; more elaborate functions are also possible. The function filter is used to check that the occurrence of a particular event E does not terminate some previous status-level assignment (if it does, those status-levels are not included in the list computed by the function status). For instance, in the context of a banking system, a user that has had an account balance greater than a certain amount for more than a year may have acquired certain privileges, but if the account is suddenly overdrawn some or all of those priviliges will stop applying. This is a specific function, and could be defined for instance by rules such as:

$$\begin{aligned}\mathsf{filter}(E, \mathsf{nil}) &\to \mathsf{nil}\\ \mathsf{filter}(E, \mathsf{cons}(L, List)) &\to \textit{if}\ \mathsf{end}(E, L)\\ &\quad\ \ \textit{then}\ \mathsf{filter}(E, List)\\ &\quad\ \ \textit{else}\ \mathsf{cons}(L, \mathsf{filter}(E, List))\end{aligned}$$

In some of the rules above we use conditionals: *if* b *then* s_1 *else* s_2. This is syntactic sugar for the term if-then-else(b, s, t), with the rewrite rules:

$$\begin{aligned}\text{if-then-else}(\mathsf{true}, X, Y) &\to X\\ \text{if-then-else}(\mathsf{false}, X, Y) &\to Y\end{aligned}$$

We also use equality tests (for example $u = user(e)$); again this is sugar for $equal(u, user(e))$, where the function $equal$ is defined by rewrite rules as usual.

It follows from the discussion above that access requests from users can be evaluated by using a rewrite system to grant or deny the request according to the history of events, from which a requester's action status is derived, and the user's ascribed status, which is specified as part of an $ASAC$ policy. More specifically, access request evaluation is performed by reducing an access request in the form $\mathsf{authorized}(a, u, o, s, l)$ to grant or deny. Below we show that grant and deny are the only normal forms for access requests, with respect to the rewrite system R_{ASAC} containing the set of rules that we have defined so far.

4.1 Properties

As we stated at the start of this section, an attraction of representing $ASAC$ as a term rewriting system is that this representation admits the possibility of proving properties of $ASAC$ policies. Notions like confluence (which implies the unicity of normal forms) and termination (which implies the existence of normal forms for all terms) may be applied to $ASAC$ policy specifications to demonstrate satisfaction of essential properties of policies.

In order for an $ASAC$ policy to be "acceptable", it is necessary that the policy satisfies certain acceptability criteria. As an informal example, it may be necessary to ensure that an access policy formulation does not specify that any user is granted and denied the same access privilege on the same data item (i.e., that the policy is consistent).

The following properties of R_{ASAC} are easy to check and will be used to show that the specification is consistent, correct and complete:

Property 1. The rewrite system R_{ASAC} is terminating and confluent.

Proof. Termination is proved using a modularity result for hierarchical unions (see Section 2 and [20]).

To prove confluence, first note that there are no critical pairs, therefore the system is locally confluent. Termination and local confluence imply confluence, by Newman's Lemma [27].

Corollary 1. *Every term has a unique normal form in R_{ASAC}.*

As a consequence of the unicity of normal forms, our specification of the $ASAC$ policy R_{ASAC} is *consistent*.

Property 2 (Consistency). For any list of events l, user u, and action a on an object o in site s, it is not possible to derive, from R_{ASAC}, both grant and deny for a request authorized(a, u, o, s, l).

We can give a characterization of the normal forms, assuming that the function symbol member is defined with the usual rules, and actionStatus, end are defined by a set of rewrite rules that satisfies the conditions of a hierarchical union and does not introduce critical pairs.

Property 3. The normal form of a ground term of the form authorized(a, u, o, s, l) where u is a user, and a an action on o in site s, is either grant or deny.

As a consequence, the policy is *total*.

Property 4 (Totality). Each access request authorized(a, u, o, s, l) is either granted or denied.

Correctness and *Completeness* are also easy to check (again assuming that member, actionStatus and end are correctly defined).

Property 5 (Correctness and Completeness).

– authorized$(a, u, o, s, l) \rightarrow^*$ grant if and only if u has the access privilege a on o in s according to the $ASAC$ policy.
– authorized$(a, u, o, s, l) \rightarrow^*$ deny if and only if u does not have the access privilege a on o in s according to the $ASAC$ policy.

Proof. Since the specification is consistent and total, it is sufficient to show that authorized$(a, u, o, s, l) \rightarrow^*$ grant if and only if u has an overall status that is assigned the access privilege a on the resource o in s. This is easy to check in the examples above, by inspection of the rewrite rules.

The rewrite rules provide an executable specification of the policy. The rules given above can be transformed into a MAUDE program by adding type declarations for the function symbols and variables used and by making minor syntactical changes.

5 Related Work

Reduction systems have been used to model a variety of problems in security. For example, the SPI-calculus [1] was developed as an extension of the π-calculus for proving the correctness of authentication protocols. The π-calculus itself has been used to reason about a number of basic access control policies and access mechanisms (see for example [3]). Term rewriting has been used in the analysis of security protocols [10], for defining policies for controlling information leakage [19], and for intrusion detection [2]. Concerning access control, Koch et al [25] use graph transformation rules to formalize RBAC, and more recently, [8,30,15] use term rewrite rules to model access control policies. The formalization used by Koch et al provides a basis for proving properties of RBAC specifications, based on the categorical semantics of the graph transformations. The approach used in [8,30,15] is operational: access control policies are specified as sets of rewrite rules and access requests are specified as terms that are evaluated using the rewrite rules. Our work addresses similar issues to [8,25,30,15] but provides a new type of access control model, *ASAC*, that we argue generalizes RBAC. On the latter point, RBAC may be viewed as a special case of *ASAC* in which the only necessary events are those involving security administrators performing the actions of user role assignment and user role deassignment, and the actions of permission assignment (to a role) and permission deassignment (from a role). The events of consequence that involve users are events of activating a role and deactivating a role. These events can be naturally accommodated in the *ASAC* model; however, the *ASAC* model additionally permits any number of other events to be represented. The *ASAC* model is also more general than the DEBAC model defined in [15]: *ASAC* policies can specify not only the way in which the actions performed by the users allow them to acquire rights, but also the way in which actions can cause a certain assignment to terminate.

The work on access control by Jajodia et al [21] and Barker and Stuckey [9] is also related to ours. In [21] and [9], access control requirements are represented in (constraint) logic programming languages. In these approaches, the requirements that must be satisfied in order for requesters to access resources are specified by using rules expressed in (C)LP languages and access request evaluation may be viewed as being performed by reducing an access request to a non-reducible clause (using, for example, SLG-resolution [32] or constraint solvers [26]). The term rewriting approach is similarly based on the idea of computation by reduction, and has similar attractions to the (C)LP approaches of Jajodia et al and Barker and Stuckey. However, in contrast to these approaches, our proposal does not require that the syntactic restriction to access policies that are *locally stratified* [6] be adopted (to ensure the existence of a categorical semantics and thus unambiguous access control policies). The *ASAC* model that we have described is more expressive than any of the *Datalog*-based languages that have been proposed for distributed access control (see [5,22,18,11]); these languages, being based on a monotonic semantics, are not especially well suited for representing dynamically *changing* access request policies of the form that we have considered in this paper.

Work on temporal RBAC [9,12], is related to our work on *ASAC* in the sense that TRBAC models are concerned with the important notion of change and events. However, in [9] and [12], the events of interest are restricted to time/clock events. In the *ASAC* model, any number of application-specific events may be represented, in addition to clock events. In [13], event expressions are used to trigger the enabling and disabling of roles in an RBAC model. To ensure that a categorical semantics exists, syntactic restrictions are imposed on the language described in [13], to treat conflicting (prioritized) event expressions; such restrictions do not need to be imposed in our approach.

We also note that although our approach is based on rules and events the framework that we describe has a well defined declarative semantics that is quite different to the *ad hoc* operational semantics that are used in, for example, *active rule systems* [14].

6 Conclusions and Further Work

We have described the representation of a novel form of access control model, the $ASAC$ model, as a term rewriting system. Our $ASAC$ model distinguishes between user ascribed statuses and action statuses and is based on the notion of an event to facilitate the autonomous changing of access control policy requirements, which may be specified by different access policy authors.

In future work, we will investigate the proving of a wide range of properties of $ASAC$ policies (which follow directly from the syntax of $ASAC$ policy specifications), and the use of efficient operational methods for the evaluation of user access requests with respect to $ASAC$ policies represented as term rewriting systems. We also note that the idea of access policy composition [31] is especially important in distributed environments (where access control information may need to be shared across multiple sites). Hence, a key matter for future work is to define appropriate algebras for $ASAC$ policy composition. A related matter for future work is to relax our assumption of atomic events and to treat access request evaluation in terms of sequences, disjunctions and conjunctions of events (for which an event algebra may be defined). We also intend to investigate the issue of evaluating access requests when conflicting information is received about the same user from different sites in a distributed system.

References

1. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conf. on Computer and Communication Security*, pages 36–47, 1997.
2. T. Abbes, A. Bouhoula, and M. Rusinowitch. Protocol analysis in intrusion detection using decision tree. In *Proc. ITCC'04*, pages 404–408, 2004.
3. J. Abendroth and C. Jensen. A unified security mechanism for networked applications. In *SAC2003*, pages 351–357, 2003.
4. F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, Great Britain, 1998.
5. J. Bacon, K. Moody, and W. Yao. A model of OASIS RBAC and its support for active security. *TISSEC*, 5(4):492–540, 2002.
6. C. Baral and M. Gelfond. Logic programming and knowledge representation. *JLP*, 19/20:73–148, 1994.
7. S. Barker. Action Status Access Control. In Proceedings ACM Symposium on Access Control Models and Technologies (SACMAT), 2007.
8. S. Barker and M. Fernández. Term rewriting for access control. In *Proc. DBSec'2006*, volume 4127 of *LNCS*. Springer-Verlag, 2006.
9. S. Barker and P. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. on Information and System Security*, 6(4):501–546, 2003.
10. G. Barthe, G. Dufay, M. Huisman, and S. Melo de Sousa. Jakarta: a toolset to reason about the JavaCard platform. In *Proceedings of e-SMART'01*, volume 2140 of *LNCS*. Springer-Verlag, 2002.
11. M. Becker and P. Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *POLICY 2004*, pages 159–168, 2004.

12. E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM TODS*, 23(3):231–285, 1998.
13. E. Bertino, P. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. In *Proc. 5th ACM Workshop on Role-Based Access Control*, pages 21–30, 2000.
14. E. Bertino, B. Catania, and G. Zarri. *Intelligent Database Systems.* Addison Wesley, 2001.
15. C. Bertolissi, M. Fernández, and S. Barker. Dynamic Event-based Access Control as Term Rewriting. In *Proc. DBSEC 2007*, LNCS, Springer, to appear.
16. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 system. In *RTA 2003*, volume 2706 in *LNCS*, pages 76–87. Springer-Verlag, 2003.
17. N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Methods and Semantics*, volume B. North-Holland, 1989.
18. J. DeTreville. Binder, a logic-based security language. In *Proc. IEEE Symposium on Security and Privacy*, pages 105–113, 2002.
19. R. Echahed and F. Prost. Security policy in a declarative style. In *Proc. PPDP'05.* ACM Press, 2005.
20. M. Fernández and J.-P. Jouannaud. Modular termination of term rewriting systems revisited. In *Proc. ADT'94*, volume 906 in *LNCS*. Springer-Verlag, 1995.
21. S. Jajodia, P. Samarati, M. Sapino, and V.S. Subrahmaninan. Flexible support for multiple access control policies. *ACM TODS*, 26(2):214–260, 2001.
22. T. Jim. SD3: A trust management system with certified evaluation. In *IEEE Symp. Security and Privacy*, pages 106–115, 2001.
23. C. Kirchner, H. Kirchner, and M. Vittek. *ELAN user manual.* Nancy (France), 1995. Technical Report 95-R-342, CRIN.
24. J.-W. Klop. Term Rewriting Systems. In S. Abramsky, Dov.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.
25. M. Koch, L. Mancini, and F. Parisi-Presicce. A graph based formalism for RBAC. In *Proc. SACMAT'04*, pages 129–187, 2004.
26. K. Marriott and P.J. Stuckey. *Programming with Constraints: an Introduction.* MIT Press, 1998.
27. M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942.
28. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
29. R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proc. 4th ACM Workshop on Role-Based Access Control*, pages 47–61, 2000.
30. A. Santana de Oliveira. Rewriting-based access control policies. In *Proc. of SECRET'06*, ENTCS. Elsevier, 2007.
31. D. Wijesekera and S. Jajodia. Policy algebras for access control the predicate case. In *ACM Conf. on Computer and Communications Security*, pages 171–180, 2002.
32. *The XSB System Version 2.7.1, Programmer's Manual*, 2005.

Automated Detection of Information Leakage in Access Control

Charles Morisset[1] and Anderson Santana de Oliveira[2]

[1] SPI - LIP6 - Université Paris 6
104 av. du Président Kennedy
F-75016 Paris, France
charles.morisset@lip6.fr
[2] INRIA & LORIA, campus scientifique, BP 239
F-54506 Vandœuvre lès Nancy, France
santana@loria.fr

Abstract. The prevention of information flow is an important concern in several access control models. Even though this property is stated in the model specification, it is not easy to verify it in the actual implementation of a given security policy. In this paper we model-check rewrite-based implementations of access control policies. We propose a general algorithm that allows one to automatically identify information leakage. We apply our approach to the well-known security model of Bell and LaPadula and show that its generalization proposed by McLean does not protect a system against information leakage.

Keywords : Access Control, Bell and LaPadula, Term Rewriting, Model-Checking, Information Flow

1 Introduction - Motivations

Data protection within information systems has become a major problem during the last years with the growth and the decentralization of computer systems. One of the key issues is to guarantee that information can be accessed and/or modified only by authorized users. A solution to this problem is to enforce an access control policy within the system, which specifies the requests for access that should be granted or denied. Moreover, an access control mechanism should also prevent leak of information. For instance, a user must be denied to copy a sensitive file, such as /etc/shadow in the UNIX file system, to a file with less restrictive rights. The mechanisms used to enforce this kind of policy can involve several methods, from abstract state machines, term rewriting systems, to static information flow analysis, etc. The latter approach consists in statically finding out every flow of data inside the code of a program, and then to avoid the ones considered to violate the security policy. This was done, for instance, by using some extension of λ-calculus as in [HR98], or directly using extensions of programming languages as in [Mye99]. Other works also use term rewriting systems [BF06,dO07] to model access control, but they are slightly different since a rewrite system can specify both the security policy and the state transitions.

Several frameworks for defining access control assume that the security policy specifying the authorized accesses is correctly defined. Indeed, they suppose it is not possible to circumvent the access control mechanisms, and there does not exist a sequence of authorized operations which can lead to a forbidden situation.

In this paper, we show that this is not always the case. We propose a method in which policy specification, given as a state machine, is separated from its implementation, given as a term rewriting system. Then we present a method based on model-checking for automatically discovering information leakages, which uses specific rules to explicit the potential information flow. We show the usefulness of the technique by applying it on an example of policy that allows the leak of information [McL88].

This paper is organized as follows. Sect. 2 presents the model of Bell and LaPadula and its implementation using both state machines and term rewriting systems. Sect. 3 introduces the McLean model, seen as a generalization of Bell and LaPadula, together with the demonstration that this model allows a leak of information. Then, in Sect. 4 we define an algorithm based on model-checking to detect such leaks.

In this paper, we adopt the usual definitions and notation for term rewriting systems contained in [Ter02,BN98], which we assume the reader to be familiar with.

2 The Bell and LaPadula Security Model and its Rewrite-Based Implementation

2.1 A framework for access control policies

In order to express access control policies, we use the mathematical framework defined in [JM06]. We do not intend to present here the whole framework, but rather the notations used in the definition of an access control policy.

An access control policy is defined as:

$$\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$$

where $\mathcal{S}$ is a set of subjects (the active entities), $\mathcal{O}$ is a set of objects (the passive entities), ρ is security information (any relevant information necessary to the definition of the security policy), $\mathcal{A}$ is the set of access modes (`read`, `write`), Σ is a set of states of the system where the policy is enforced, and Ω is the security predicate, defining the secure states. Roughly speaking, an access control policy specifies the system states, and which of the states are secure. We then need to specify how to move from one state to another. So we introduce the notion of access control model:

$$\mathbb{M}[\rho] = (\mathbb{P}[\rho], [\![\mathcal{R}]\!]_{\Sigma})$$

where $\mathbb{P}[\rho]$ is a security policy, $\mathcal{R}$ is a set of requests and $[\![\mathcal{R}]\!]_{\Sigma} \subseteq \mathcal{R} \times \Sigma$ is the semantics of requests. This relation allows to express the modification expected by the application of a request. For instance, a request R asking to add the access A should have a semantics defined such that for all $\sigma \in \Sigma$, if $(R, \sigma) \in [\![\mathcal{R}]\!]_{\Sigma}$, then $A \in \sigma$.

Finally, an implementation of a model $\mathbb{M}[\rho]$ is a pair (τ, Σ_I), where $\tau : \mathcal{R} \times \Sigma \rightarrow \mathcal{D} \times \Sigma$ is a transition function over states (and $\mathcal{D}$ is a set of decisions such yes or no) and Σ_I is a set of initial states.

2.2 The Bell and LaPadula model

The Bell and LaPadula model [LB96,BL73] was originally defined for the military domain. The main idea is that every subject and every object is associated with a security level, such as *Top Secret, Secret*, etc. The original model deals with discretionary access control (in order to access an object, any subject needs the authorization from the owner of the object) and with mandatory access control (an access is granted or not according to the security level of the subject and of the object). Since discretionary access control can be seen as a completely separated part of the policy, we have decided, without loss of generality, to focus only on the mandatory part in this paper, which is really specific to the Bell and LaPadula model.

More formally, we write $\rho_{BLP} = (\mathcal{L}, \preceq, \sqcup, \sqcap)$ the *lattice of security levels*, where $\mathcal{L}$ is a set of security levels, $\preceq$ is the order defined over this set, $\sqcup$ is the least upper bound and $\sqcap$ is the greatest lower bound. Then, a state $\sigma \in \Sigma_{BLP}$ is a tuple $\sigma = (m, f_s, f_o)$ where m is the set of current accesses and $f_s : \mathcal{S} \to \mathcal{L}$ (resp. $f_o : \mathcal{O} \to \mathcal{L}$) defines levels of security associated with subjects (resp. objects). An access is represented by a tuple (s, o, x) expressing that a subject s has an access over the object o according to access mode x.

The *Bell and LaPadula security policy* is specified by a predicate Ω_{BLP} as follows. Given a state $\sigma = (m, f_s, f_o)$, $\Omega_{BLP}(\sigma)$ holds iff the following two properties are satisfied:

$$\forall s \in \mathcal{S}\ \forall o \in \mathcal{O} \quad (s, o, \texttt{read}) \in m \Rightarrow f_o(o) \preceq f_s(s) \tag{1}$$

$$\forall s \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O} \quad (s, o_1, \texttt{read}) \in m \wedge (s, o_2, \texttt{write}) \in m \Rightarrow f_o(o_1) \preceq f_o(o_2) \tag{2}$$

The property (2), also known as the $\star$-security property, allows to prevent the copy of an object to a lower security level by a malicious subject. We write $\mathbb{P}_{BLP}[\rho_{BLP}] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma_{BLP}, \Omega_{BLP})$ such an access control policy.

The set $\mathcal{R}$ of requests is defined as follows:

$$\mathcal{R} = \{\langle +, s, o, \texttt{read}\rangle, \langle +, s, o, \texttt{write}\rangle, \langle -, s, o, \texttt{read}\rangle, \langle -, s, o, \texttt{write}\rangle\} \tag{3}$$

The request $\langle +, s, o, x\rangle$ (resp. $\langle -, s, o, x\rangle$) is the request where the subject s asks for getting (resp. releasing) the access to the object o according to access mode x. The semantics of requests $[\![\mathcal{R}]\!]_\Sigma$ is defined as follows:

$$\begin{array}{l} (\langle +, s, o, x\rangle, \sigma) \in [\![\mathcal{R}]\!]_\Sigma \Leftrightarrow (s, o, x) \in \Lambda(\sigma) \\ (\langle -, s, o, x\rangle, \sigma) \in [\![\mathcal{R}]\!]_\Sigma \Leftrightarrow (s, o, x) \notin \Lambda(\sigma) \end{array} \tag{4}$$

where $x \in \{\texttt{read}, \texttt{write}\}$ and $\Lambda(\sigma)$ stands for the set of current accesses of σ.

We write $\mathbb{M}_{BLP}[\rho_{BLP}] = (\mathbb{P}_{BLP}[\rho_{BLP}], [\![\mathcal{R}]\!]_\Sigma)$ the Bell and LaPadula access control model.

Given a set of initial states Σ_I^{BLP}, we introduce the implementation $(\tau_{BLP}, \Sigma_I^{BLP})$ of $\mathbb{M}_{BLP}[\rho_{BLP}]$ where τ_{BLP} is the transition function defined in Table 1 and corresponds to the restriction of the original function of Bell and LaPadula, without discretionary access control and only taking into account `read` and `write` access modes, since the access modes `append`, `control` and `execute` are not relevant in mandatory access control.

$$\tau_{BLP}(R,(m,f_s,f_o))$$

$$= \begin{cases} (\text{yes}, (m \cup \{(s,o,\texttt{read})\}, f_s, f_o)) \\ \text{if } \ R = \langle +, s, o, \texttt{read}\rangle \\ \wedge \ f_o(o) \preceq f_s(s) \wedge \{o' \in \mathcal{O} \mid (s,o',\texttt{write}) \in m \ \wedge \neg(f_o(o) \preceq f_o(o'))\} = \varnothing \\ \\ (\text{yes}, (m \cup \{(s,o,\texttt{write})\}, f_s, f_o)) \\ \text{if } \ R = \langle +, s, o, \texttt{write}\rangle \\ \wedge \ \{o' \in \mathcal{O} \mid (s,o',\texttt{read}) \in m \ \wedge \neg(f_o(o') \preceq f_o(o))\} = \varnothing \\ \\ (\text{yes}, (m \backslash \{(s,o,x)\}, f_s, f_o)) \\ \text{if } \ R = \langle -, s, o, x\rangle \\ \\ (\text{no}, (m, f_s, f_o)) \text{ otherwise} \end{cases}$$

Table 1. Implementation of the Bell and LaPadula policy

2.3 Rewrite implementation of the Bell and LaPadula model

Implementing the Bell and LaPadula policy according to the transition function defined in Table 1 leads to the following system:

– Let $\mathcal{F}$ be the signature:

τ	:	$\mathcal{R} \times M$	$\rightarrow$	M
req	:	$Mode \times \mathcal{S} \times \mathcal{O} \times \mathcal{A}$	$\rightarrow$	$\mathcal{R}$
s	:	$Id \times Level$	$\rightarrow$	$\mathcal{S}$
o	:	$Id \times Level$	$\rightarrow$	$\mathcal{O}$
m	:	$\mathcal{S} \times \mathcal{O} \times \mathcal{A}$	$\rightarrow$	$Access$
$\wedge$	:	$Access \times Access$	$\rightarrow$	M
`read`, `write`	:		$\rightarrow$	$\mathcal{A}$
$+,\ -$	:		$\rightarrow$	$Mode$
$\top, I, \bot$	:		$\rightarrow$	$Level$
yes, no	:		$\rightarrow$	$Access$

– The conditional rewrite rules implementing the policy contain the variables i_1, i_2, i_3 whose sort is *Id*, l_1, l_2, l_3 which have sort *Level*, x has sort *Access* and y is of sort M. Please remark that we consider the operator $\wedge$ to be associative and commutative. While analyzing the rules below the reader must be aware that there is an underlying strategy, which tries to apply the rules in the order they are presented, which is actually the semantics of several implementations of term rewriting systems.

$$
\begin{array}{ll}
(r_1)\ \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{read}), x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{write})) & \to \\
\quad \text{no} \wedge x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{write}) \qquad \textbf{if } (l_2 \preceq l_3) \to false & \\
(r_2)\ \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{read}), y) & \to \\
\quad \text{no} \wedge y \qquad \textbf{if } (l_2 \preceq l_1) \to false & \\
(r_3)\ \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{read}), y) & \to \\
\quad \text{yes} \wedge m(s(i_1,l_1), o(i_2,l_2), \texttt{read}) \wedge y & \\
(r_4)\ \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{write}), x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{read})) & \to \\
\quad \text{no} \wedge x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{read}) \qquad \textbf{if } (l_3 \preceq l_2) \to false & \\
(r_5)\ \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{write}), y) & \to \\
\quad \text{yes} \wedge m(s(i_1,l_1), o(i_2,l_2), \texttt{write}) \wedge y & \\
(r_6)\ \tau(req(-, s(i_1,l_1), o(i_2,l_2), a), m(s(i_1,l_1), o(i_2,l_2), a) \wedge y) & \to \\
\quad \text{yes} \wedge y &
\end{array}
$$

3 The McLean Model

The algebra of security [McL88] can be seen as a generalization of the model of Bell and LaPadula. One of the concepts introduced in this algebra is the one of joint access. Indeed, in some fields, as the military one, it could be useful to consider access as done by a group of subjects rather than by a subject alone. A typical example is the one where in order to launch a missile, two individuals have to push a button *at the same time*. Accesses are then represented as a tuple (S, o, x), where S is a non-empty set of subjects, o is an object and x is an access mode.

In order to take into account joint access, the $\star$-security property, corresponding to the property 2, is defined in [McL88], as follows:

"a state is $\star$-secure if for any subjects S_1, S_2 and objects o_1, o_2, if $(S_1, o_1, \texttt{read}) \in m$ and $(S_2, o_2, \texttt{write}) \in m$ and the classification of o_1 dominates that of o_2, then $S_1 \cap S_2 = \emptyset$ "

A direct translation of this sentence leads to the following logical formula:

$$
\begin{array}{l}
\forall S_1, S_2\ \forall o_1, o_2 \in \mathcal{O} \\
\quad ((S_1, o_1, \texttt{read}) \in m \wedge (S_2, o_2, \texttt{write}) \in m \wedge f_o(o_2) \prec f_o(o_1)) \\
\Rightarrow S_1 \cap S_2 = \emptyset
\end{array}
$$

which is logically equivalent (by contraposition) to:

$$
\begin{array}{l}
\forall S_1, S_2\ \forall o_1, o_2 \in \mathcal{O} \\
\quad ((S_1, o_1, \texttt{read}) \in m \wedge (S_2, o_2, \texttt{write}) \in m \wedge S_1 \cap S_2 \neq \emptyset) \\
\Rightarrow \neg(f_o(o_2) \prec f_o(o_1))
\end{array}
\tag{5}
$$

If we do not take into account joint access, every set of subjects S can be reduced to a singleton set $\{s\}$, and the property (5) is equivalent to:

$$
\begin{array}{l}
\forall s\ \forall o_1, o_2 \in \mathcal{O} \\
(\{s\}, o_1, \texttt{read}) \in m \wedge (\{s\}, o_2, \texttt{write}) \in m \Rightarrow \neg(f_o(o_2) \prec f_o(o_1))
\end{array}
\tag{6}
$$

With this new definition of the ⋆-property, the rewrite implementation of the McLean model can be obtained from the one of Bell and LaPadula, by replacing the rule (r_1) by

$$\begin{aligned}(r_7)\ &\tau(req(+, s(i_1, l_1), o(i_2, l_2), \texttt{read}), x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{write})) \rightarrow \\ &\text{no} \wedge x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{write}) \qquad\qquad \text{if } (l_3 \prec l_2) {\rightarrow} true\end{aligned}$$

and the rule (r_4) by

$$\begin{aligned}(r_8)\ &\tau(req(+, s(i_1, l_1), o(i_2, l_2), \texttt{write}), x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{read})) \rightarrow \\ &\text{no} \wedge x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{read}) \qquad\qquad \text{if } (l_2 \prec l_3) {\rightarrow} true\end{aligned}$$

Clearly, since $\preceq$ is only a partial order defining the lattice of security levels, the properties (2) and (6) are not equivalent (they become equivalent when $\preceq$ is a total order, but most of the time this is not the case).

Moreover, with such a definition of the security policy, a leak of information is possible. Indeed, let us consider the lattice $\mathcal{L} = \{min, \bot, I, \top, max\}$, where $\bot \preceq \top$, I is not comparable to $\bot$ or $\top$ and for all x in $\mathcal{L}$, $min \preceq x \preceq max$. Let $\mathcal{O}$ be the set of objects $\{o_1, o_2, o_3\}$ and f_o the security function such that $f_o(o_1) = \bot$, $f_o(o_2) = I$ and $f_o(o_3) = \top$. Let $\mathcal{S}$ be the set of subjects $\{s_1, s_2\}$ and f_s the security function such that $f_s(s_1) = \top$ and $f_s(s_2) = I$.

The state $\sigma_1 = (m_1, f_s, f_o)$, represented in Fig. 1, where $m_1 = \{(s_1, o_3, \texttt{read}), (s_1, o_1, \texttt{write})\}$, is clearly not secure, since $f_o(o_1) \prec f_o(o_3)$, and so the high-level information contained in o_3 cannot go to o_1.

Let us now consider the state $\sigma_2 = (m_2, f_s, f_o)$, represented in Fig. 2, where $m_2 = \{(s_1, o_3, \texttt{read}), (s_1, o_2, \texttt{write}), (s_2, o_2, \texttt{read}), (s_2, o_1, \texttt{write})\}$. According to the McLean policy, this state is secure since $\neg(f_o(o_2) \prec f_o(o_3))$ and $\neg(f_o(o_1) \prec f_o(o_2))$. So we clearly have a leak of information since this state is secure in spite of the fact that information contained in o_3 can go to o_1 passing through o_2.

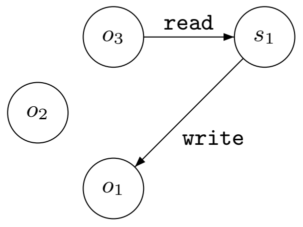

Fig. 1. Insecure state

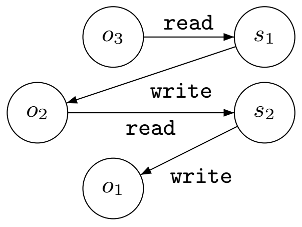

Fig. 2. Secure state

We propose in the next section a method that allows us to find out such leaks of information, based on model-checking of term rewriting systems.

4 Model-Checking Access Control Policies

Model-checking is well adapted for searching software flaws when proof methods cannot be fully automated for a given property. This is likely the situation for proving that one implementation of a certain security model meets its specification. When an access control policy is implemented by a term rewriting system, this kind of verification can be performed by using a technique that checks the reachability of a certain term, given a term rewriting system and an input term.

Reachability over term rewriting is a very general approach which constructs approximations of the set of normal forms w.r.t. a term rewriting system [Gen98]. This is done through the use of tree automata. Since the “target” terms represent unwanted situations in general, if they are captured by an approximation, then they will appear in one of the derivations of the actual term rewriting system. In our work, we use a simplified technique using the exact sets of normal forms, since computing the approximations is only necessary in a general framework for term rewriting systems.

Roughly speaking, the algorithm we propose here consists in making explicit every implicit information flow and it checks if this information flow is correct according to the security policy. An information flow is implicit when one can consider that a subject actually has access to an object, but the corresponding access tuple is not present in the set of current accesses. As an example of implicit information flow, we can consider the following situation: if a subject s_1 is reading an object o_1 and writing to an object o_2, and at the same time, a subject s_2 is reading the object o_2, then one can consider that s_2 is also reading the object o_1. In other words, if the accesses $(s_1, o_1, \texttt{read})$, $(s_1, o_2, \texttt{write})$ and $(s_2, o_2, \texttt{read})$ belong to a set of accesses, then we have to add the access $(s_2, o_1, \texttt{read})$ to this set. In this paper, we add this access thanks to a term rewriting system $\mathcal{R}'$, represented in Fig. 3. Note that adding a new access in a set of accesses may lead to the creation of new implicit information flows.

$$\begin{array}{lr} m(s_1, o_1, \texttt{read}) \wedge m(s_1, o_2, \texttt{write}) \wedge m(s_2, o_2, \texttt{read}) \wedge x & \rightarrow \\ m(s_1, o_1, \texttt{read}) \wedge m(s_1, o_2, \texttt{write}) \wedge m(s_2, o_2, \texttt{read}) \wedge m(s_2, o_1, \texttt{read}) \wedge x & \\ \text{if } m(s_2, o_1, \texttt{read}) \notin x & \end{array}$$

Fig. 3. The Rewrite System $\mathcal{R}'$

The system $\mathcal{R}'$ consists of only one rule, and can be applied only if the implicit flow detected does not already belong to the set of accesses.

In order to define the algorithm of detection of information leakage, we introduce the function `App`, which takes a term and a (confluent and terminating) term rewriting system, and which returns the term obtained after application of one rule of the system. In other words, $\texttt{App}(t,\ \mathcal{R})\ =\ t'$ if $t \rightarrow_{\mathcal{R}} t'$. If no rule can applied to the term, the function `App` raise the exception *Fail*.

The algorithm of detection of information leakage relies on two sub-algorithms. The first one, represented by the Algorithm 1, makes explicit every implicit information flow, and checks that none of these flows violates the security policy.

Algorithm 1 (Verification of a set of accesses)

```
Verification($M$: set of accesses,
             $\mathcal{R}, \mathcal{R}'$ : term rewriting systems)=
    $M_d \leftarrow M$;
    While App($M_d$, $\mathcal{R}'$) $\neq$ $Fail$ Do
       $M_d \leftarrow$ App($M_d$, $\mathcal{R}'$);
    End While;
    For each $m(s, o, x) \in M_d \backslash M$ Do
       If App($\tau(req(+, s, o, x), M)$, $\mathcal{R}$) = no $\wedge$ $M$ Then
             Return $false$;
    End For;
    Return $true$;
End Verification;
```

The second sub-algorithm, represented by the Algorithm 2, takes a set of subjects, a set of objects and two term rewriting systems. For each subject and each object, the algorithm creates two requests corresponding to the read access and the write access, and tries to apply these two requests, in order to obtain a new set of accesses. The function `Verification` is then applied to each set of accesses. If one of them creates an information leakage, then the algorithm stops.

Algorithm 2 (Creation of sets of accesses)

```
Check($\mathcal{S}$: set of subjects,
        $\mathcal{O}$: set of objects,
        $\mathcal{R}, \mathcal{R}'$ : term rewriting systems)=
   $M \leftarrow \emptyset$;
   For each $s \in \mathcal{S}$ and each $o \in \mathcal{O}$ Do
       $m_w \leftarrow req(+, s, o, \texttt{write})$;
       $m_r \leftarrow req(+, s, o, \texttt{read})$;
       If App($\tau(m_w, M)$, $\mathcal{R}$) = yes $\wedge$ $x \wedge M$ Then
          $M \leftarrow x \wedge M$;
       If App($\tau(m_r, M)$, $\mathcal{R}$) = yes $\wedge$ $x \wedge M$ Then
          $M \leftarrow x \wedge M$;
       If Verification($M, \mathcal{R}, \mathcal{R}'$) = $false$ Then
          Return "information leakage detected";
    End For;
    Return "no information leakage detected";
End Check;
```

The different sets of accesses created by the Algorithm 2 depend on the "order" of the sets $\mathcal{S}$ and $\mathcal{O}$. Indeed, let us consider for instance the sets $\mathcal{S} = \{s_1, s_2, s_3\}$ and $\mathcal{O} = \{o_1, o_2\}$. If the loop `For` considers each subject and each object in this order, then the first requests created are $m_w = req(+, s_1, o_1, \texttt{write})$ and $m_r = req(+, s_1, o_1, \texttt{read})$. If these two requests are granted, then, in the following, the set of accesses M will contain the access tuples $m(s_1, o_1, \texttt{write})$ and $m(s_1, o_1, \texttt{read})$. Hence, the algorithm will not verify other sets of accesses which do not contain both of these tuples. Since such sets of accesses can be reachable and respect the security policy, our algorithm is clearly not complete. Indeed, let us consider for instance that only the set of accesses:

$$m(s_2, o_1, \texttt{read}) \wedge m(s_2, o_2, \texttt{write}) \wedge m(s_3, o_2, \texttt{read})$$

creates an information flow, and that this set is secure and reachable. Let us also consider that the security policy is defined in a way that s_1 can read o_1, s_3 can read o_2 but these two accesses cannot appear at the same time. If the access $m(s_1, o_1, \texttt{read})$ is first added, and never removed, then the access $m(s_3, o_2, \texttt{read})$ will never be added, and so the information leakage is never detected. On the contrary, if the access $m(s_3, o_2, \texttt{read})$ is added first, then the information leakage can be detected. In order to obtain a complete algorithm, we need to call the function `Check` on every permutation of the sets $\mathcal{S}$ and $\mathcal{O}$. For instance, if $\mathcal{S} = \{s_1, s_2, s_3\}$ and $\mathcal{O} = \{o_1, o_2\}$, we need to call the function `Check` as described in Fig. 4.

```
Check({s1, s2, s3}, {o1, o2}, R, R');
Check({s1, s3, s2}, {o1, o2}, R, R');
Check({s2, s1, s3}, {o1, o2}, R, R');
Check({s2, s3, s1}, {o1, o2}, R, R');
Check({s3, s1, s2}, {o1, o2}, R, R');
Check({s3, s2, s1}, {o1, o2}, R, R');
Check({s1, s2, s3}, {o2, o1}, R, R');
Check({s1, s3, s2}, {o2, o1}, R, R');
Check({s2, s1, s3}, {o2, o1}, R, R');
Check({s2, s3, s1}, {o2, o1}, R, R');
Check({s3, s1, s2}, {o2, o1}, R, R');
Check({s3, s2, s1}, {o2, o1}, R, R');
```

Fig. 4. Calls of function `Check`

This method is clearly naive, because it leads to verifying several times the same set of accesses, but it allows to have a complete algorithm.

In the case of the policy described in Sect. 3, information flow can be detected starting from $n_s = 2$ and $n_o = 2$, provided that at least three different confidentiality levels exist, in only four steps of the execution of the main loop, whose trace can be seen on Fig. 5.

We implemented this algorithm in Tom[3] [KMR05,BBK+07]. The implementation follows the same general lines of the model-checking approach employed to find attacks in security protocols, as described in [CMR05], which explores the entire search space by applying all possible rules, over each successive state of the system, until the undesired situation is identified.

The technique we presented here can be adapted to check other access control models. It is is only necessary to substitute the policy implementation, with the corresponding implementation for the new model. However, if this algorithm does not capture an information flow to a certain instance of a policy model, as in any other model-checking technique, it does not mean that the implementation is secure with respect to the information flows.

[3] `http://tom.loria.fr`

Initialization.

Generation of the sets of $\mathcal{S} = \{s(0,\top), s(1,I)\}$, and $\mathcal{O} = \{o(0,\top), o(1,I)\}$.

The set of current accesses is empty: $M = \varnothing$

Step 1

Requests generated	Current accesses
$req(+, s(0,\top), o(0,\top), \texttt{read})$ $req(+, s(0,\top), o(0,\top), \texttt{write})$.	$m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\ m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = M \Rightarrow M_d \backslash M = \varnothing$, no information flow detected.

Step 2

Requests generated	Current accesses
$req(+, s(0,\top), o(1,I), \texttt{read})$ $req(+, s(0,\top), o(1,I), \texttt{write})$	$m(s(0,\top), o(1,I), \texttt{write})$ $\wedge\ m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\ m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = M \Rightarrow M_d \backslash M = \varnothing$, no information flow detected.

Step 3

Requests generated	Current accesses
$req(+, s(1,I), o(0,\top), \texttt{read})$ $req(+, s(1,I), o(0,\top), \texttt{write})$	$m(s(1,I), o(0,\top), \texttt{write})$ $\wedge\ m(s(0,\top), o(1,I), \texttt{write})$ $\wedge\ m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\ m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = M \Rightarrow M_d \backslash M = \varnothing$, no information flow detected.

Step 4.

Requests generated	Current accesses
$req(+, s(1,I), o(1,I), \texttt{read})$ $req(+, s(1,I), o(1,I), \texttt{write})$	$m(s(1,I), o(1,I), \texttt{write})$ $\wedge\ m(s(1,I), o(1,I), \texttt{read})$ $\wedge\ m(s(1,I), o(0,\top), \texttt{write})$ $\wedge\ m(s(0,\top), o(1,I), \texttt{write})$ $\wedge\ m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\ m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = m(s(1,I), o(0,\top), \texttt{read}) \wedge \ldots \wedge m(s(0,\top), o(0,\top), \texttt{write})$

$M_d \backslash M = m(s(1,I), o(0,\top), \texttt{read})$. When we apply the transition rule to this request we obtain $\tau(req(+, s(1,I), o(0,\top), \texttt{read}), M) = \text{no} \wedge M$ which characterizes the information flow. The execution then stops.

Fig. 5. Execution trace for $n_s = 2$ and $n_o = 2$

5 Conclusion

In this paper we proposed a technique based on model-checking to detect information flow in access control models. We define the state transition function that implements a security policy using a term rewriting systems. The rule-based algorithm we provided explores the space of all possible requests in order to automatically identify information flow in these implementations. We illustrated the application of our approach over the McLean's security algebra, a generalization of the Bell and LaPadula security model. We showed that it fails to prevent information leakage even on a relatively small setting.

Some related works use model-checking in the domain of access control policies. In [GRS04], the authors verify the satisfiability of a temporal logic formula in the policy model they presented, but it is not clear how they could address information flow. In [DFK06], the authors suggest to perform goal reachability over Datalog programs as a means to compare access control policies, but information flow is not addressed as well. The main advantage of our approach is that it can be easily adapted to several access control policies. Furthermore, one can use automated approaches to translate logic programs into term rewriting systems [SKGST07] and then apply the algorithm we introduce here.

As future work, we consider to extend our technique to check other properties such as answering more general queries, for instance, whether access is always denied for a given object, or whether access is granted and denied at the same time to a given request. We also intend to enhance the algorithm `Check`, in order to avoid superflous calls of this function on the several permutations of an access tuple.

Acknowledgments Many thanks to Horatiu Cirstea, Thérèse Hardin, Eric Jaeger, Mathieu Jaume and Claude Kirchner for enlightening discussions on this subject. We also thank Guillaume Burel. Oana Andrei, and Radu Kopetz for proof reading previous drafts of this paper and the anonymous referees for their very useful remarks.

References

BBK$^+$07. Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking rewriting on java. In *Proceedings of the 18th Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science. Springer-Verlag, 2007.

BF06. Steve Barker and Maribel Fernández. Term rewriting for access control. In *DBSec*, pages 179–193, 2006.

BL73. D. Bell and L. LaPadula. Secure Computer Systems: a Mathematical Model. Technical Report MTR-2547 (Vol. II), MITRE Corp., Bedford, MA, May 1973.

BN98. Franz Baader and Tobias Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.

CMR05. Horatiu Cirstea, Pierre-Etienne Moreau, and Antoine Reilles. Rule-based programming in java for protocol verification. *Electr. Notes Theor. Comput. Sci.*, 117:209–227, 2005.

DFK06. Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 632–646. Springer, 2006.

dO07. Anderson Santana de Oliveira. Rewriting-based access control policies. In *Proceedings of - SECRET'06*, ENTCS. Elsevier, 2007.

Gen98. Thomas Genet. *Contraintes d'ordre et automates d'arbres pour les preuves de terminaison*. PhD thesis, Université Henri Poincaré - Nancy I, 1998.

GRS04. Dimitar P. Guelev, Mark Ryan, and Pierre-Yves Schobbens. Model-checking access control policies. In Kan Zhang and Yuliang Zheng, editors, *ISC*, volume 3225 of *Lecture Notes in Computer Science*, pages 219–230. Springer, 2004.

HR98. Nevin Heintze and Jon G. Riecke. The SLam calculus: programming with secrecy and integrity. In ACM, editor, *Symposium on Principles of Programming Languages, San Diego, California*, pages 365–377, New York, NY, USA, 1998. ACM Press.

JM06. M. Jaume and C. Morisset. Towards a formal specification of access control. In P. Degano, R. Kusters, L. Vigano, and S. Zdancewic, editors, *Proceedings of FCS-ARSPA'06*, pages 213–232, 2006.

KMR05. Claude Kirchner, Pierre-Etienne Moreau, and Antoine Reilles. Formal validation of pattern matching code. In Pedro Barahona and Amy P. Felty, editors, *PPDP*, pages 187–197. ACM, 2005.

LB96. L.J. LaPadula and D.E. Bell. Secure Computer Systems: A Mathematical Model. *Journal of Computer Security*, 4:239–263, 1996.

McL88. McLean. The algebra of security. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–7. IEEE Computer Society Press, 1988.

Mye99. Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Symposium on Principles of Programming Languages*, pages 228–241, 1999.

SKGST07. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs by term rewriting. In German Puebla, editor, *LOPSTR*, volume 4407 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2007.

Ter02. Terese. *Term Rewriting Systems*. Cambridge University Press, 2002.

Diffie-Hellman Cryptographic Reasoning in the Maude-NRL Protocol Analyzer

Santiago Escobar[1], Joe Hendrix[2], Catherine Meadows[3], and José Meseguer[4]

[1] Universidad Politécnica de Valencia, Spain. sescobar@dsic.upv.es
[2] University of Illinois at Urbana-Champaign, USA. jhendrix@cs.uiuc.edu
[3] Naval Research Laboratory, Washington, DC, USA. meadows@itd.nrl.navy.mil
[4] University of Illinois at Urbana-Champaign, USA. meseguer@cs.uiuc.edu

Abstract. The Maude-NRL Protocol Analyzer (Maude-NPA) is a tool and inference system for reasoning about the security of cryptographic protocols in which the cryptosystems satisfy different equational properties. It both extends and provides a formal framework for the original NRL Protocol Analyzer, which limited itself to an equational theory Δ of convergent rewrite rules. In this paper we extend our framework to include theories of the form $\Delta \uplus B$, where B is the theory of associativity and commutativity and Δ is convergent modulo B. Order-sorted B-unification plays a crucial role; to obtain this functionality we describe a sort propagation algorithm that filters out unsorted B-unifiers provided by the CiME unification tool. We show how extensions of some of the state reduction techniques of the original NRL Protocol Analyzer can be applied in this context. We illustrate the ideas and capabilities of the Maude-NPA with an example involving the Diffie-Hellman key agreement protocol.

1 Introduction

The Maude-NPA is a tool and inference system for reasoning about the security of cryptographic protocols in which the cryptosystems satisfy different equational properties. It is the next generation of the NRL Protocol Analyzer [21], a tool that supported limited equational reasoning and was successfully applied to the analysis of many different protocols. In Maude-NPA we improved on the original NPA in two ways. First of all, in [15] we formalized the inference system of NPA, providing the first formal description of the tool, in terms of rewriting logic and narrowing. We also provided proofs of soundness and completeness. More recently, we have been extending the equational reasoning capabilities of the tool. In [15], we considered equational theories for cryptographic primitives and other functions given by a confluent and terminating set of equations Δ, as did the original NPA. In this work, we extend the framework to theories of the form $\Delta \uplus B$, where B is the theory of associativity and commutativity (AC) and Δ is confluent and terminating modulo B. We illustrate the ideas and capabilities of this extension of the Maude-NPA with an example using the Diffie-Hellman protocol, involving exponentiation and associative-commutative multiplication.

Maude-NPA uses backwards search from an insecure state to find attacks or to prove unreachability. This is implemented using backwards narrowing with the protocol rules *modulo* $\Delta \uplus B$. There are two ways to do this. One is to use built-in unification algorithms for each theory and combination of theories. The other is to use a hybrid approach, for example to use built-in algorithms for B, and a generic algorithm, such as narrowing modulo B, for Δ. We choose the second approach, as being more readily extensible to different theories. That is, we use narrowing at two levels: for backwards search from an insecure state (i.e., $\Delta \uplus B$-narrowing with the rewrite rules describing the protocol) and for the $\Delta \uplus B$-unification used in each backwards narrowing step (i.e., B-narrowing with the rewrite rules obtained by orienting Δ).

We have found important technical advantages in using *order-sorted* theories. In order-sorted theories, narrowing will terminate, providing a finitary unification algorithm, in many cases in which unsorted narrowing will not. Furthermore, even in the case in which both terminate, order-sorted narrowing will often produce a smaller search space. Two interesting examples of this use of order-sorted theories to obtain finitary unification algorithms are our approximate theory for associativity in [14], and the theory $\Delta \uplus B$ for Diffie-Hellman exponentiation in this paper. In both cases, narrowing with the corresponding unsorted theories is non-terminating, whereas narrowing with the order-sorted theories does terminate. The current support of order-sorted unification in the Maude-NPA leverages CiME's [11] rich library of composable unsorted unification algorithms, including any combination of associativity, commutativity and identity axioms (except associativity without commutativity). The Maude-NPA then filters out with order-sorted information the unsorted unifiers provided by CiME, using the sound and complete sort propagation algorithm presented in Section 4 to obtain the corresponding order-sorted unifiers.

1.1 Related work

Recently, the modeling and formal analysis of cryptographic protocols in which the cryptographic and other functions used obey different equational theories has been an active research topic. Much of the work in this area has concentrated on problems of secrecy and static equivalence in bounded session protocols, which have been proved to be decidable for an important class of equational theories [23,7,1,9,6,12]. For unbounded sessions, however, the problem is less well understood and has been recently studied in [5,4]. In [17] and [7] tree-automata based approximations are applied to associative-commutative theories to develop abstract approximations. These techniques have been implemented in tools (for Diffie-Hellman theories in [17], and for exclusive-or in [7]) that guarantee protocol security if they find no attacks, but may find spurious attacks even if the protocol is sound. This is in contrast to the aim of our work, which is to provide both genuine proofs of security and genuine attacks, while using heuristic techniques, such as grammars, to make termination more likely even if it is not guaranteed.

Our work, which relies on backwards narrowing, requires a sound and complete unification algorithm for associative-commutative theories such as exponentiation and exclusive-or. Probably the most closely related work to ours in this area is the unification algorithms for exponentiation in Narendan and Meadows [20] and Kapur, Narendran, and Wang [19]. The theory we use in this paper is an order-sorted version of a fragment of the theories for which unification algorithms are developed there. In this work, however, we use a hybrid approach that we believe is more readily extensible. Instead of developing special-purpose algorithms for individual theories, we make use of an order-sorted unification algorithm for associative-commutative theories, obtained in this case by combining the AC algorithm from the CiME tool with our sort propagation algorithm. This is then combined with rewrite theories that are terminating and confluent with respect to the AC theory. Some other work that is closely related to ours is the work of Comon-Lundh and Delaune [10] on the finite variant property, in which techniques are developed for achieving termination even when narrowing by itself does not terminate. Although this paper does not make use of their results, we expect to find them helpful to our future work.

1.2 A Diffie-Hellman Example

In order to demonstrate the use of associativity and commutivity (AC) in the Maude-NPA, we provide an example involving the well-known Diffie-Hellman key agreement protocol. This protocol uses exponentiation in order to generate a shared secret between two parties, and is the basis for most existing key agreement protocols today. However, if it is used without authentication, it is subject to man-in-the-middle attacks in which an attacker can convince two principals who are trying to share a key with each other that they actually do, while in fact they share a key with the attacker. Analyzing Diffie-Hellman without authentication allows us not only to demonstrate how Maude-NPA handles cryptographic functions involving AC axioms, but also how it can be used to find attacks on protocols that use AC operators combined with other algebraic properties of the underlying cryptographic functions.

Example 1. This protocol uses exponentiation to share a secret between two parties, Alice and Bob. The protocol involves an initiator, Alice, and a responder, Bob. We use the common notation $A \hookrightarrow B : M$ to stand for "A sends message M to B". Raising message M to the power of exponent X is denoted by $(M)^X$. There is a public term denoted by g, which will be the base of our exponentiations. We represent the product of exponents by using the symbol $*$. Nonces are represented by N_X, denoting a nonce created by principal X. The protocol description is as follows.

1. $A \hookrightarrow B : A$
   Alice sends her name to Bob.
2. $A \hookrightarrow B : B$
   Alice sends Bob's name to Bob.
3. $A \hookrightarrow B : g^{N_A}$
   Alice creates a new nonce N_A and sends g^{N_A} to Bob.
4. $B \hookrightarrow A : B$
   Bob sends his name to Alice.
5. $B \hookrightarrow A : A$
   Bob sends Alice's name to herself.
6. $B \hookrightarrow A : g^{N_B}$
   Bob creates a new nonce N_B and sends g^{N_B} to Alice.

Intuitively, when Bob receives g^{N_A}, he raises it to the N_B, to obtain ${g^{N_A}}^{N_B} = g^{N_A * N_B}$. Likewise, when Alice receives g^{N_B}, she raises it to the N_A, to obtain ${g^{N_B}}^{N_A} = g^{N_B * N_A}$. And due to the commutativity of the symbol $*$, they know the equivalence $g^{N_B * N_A} = g^{N_A * N_B}$. An observer of the exchange who does not know N_A nor N_B cannot find $g^{N_A * N_B}$, and so Alice and Bob have computed a shared secret, i.e., $g^{N_A * N_B}$.

Of course, the attacker can always learn a term $g^{(N_A * N_I)}$, where N_I is a nonce created by the intruder, even by using a passive intruder model. The point is that he can also make believe to Alice that $g^{(N_A * N_I)}$ is the shared key she is sharing with *Bob*. This is usually modelled by adding to the protocol a new message where Alice sends to Bob some secret, encrypted by $g^{(N_A * N_I)}$. Existence of an attack is expressed by saying that the attacker can obtain this secret. For the sake of simplicity and because we are focused in AC-theories, we omit this last part of the protocol and concentrate just in whether the intruder can learn X^{N_A} for some exponentiation X, where X^{N_A} is the key calculated by Alice.

In a rule-based representation of this protocol, parts of a received message whose make-up cannot be verified by a principal are represented by variables. That is, since nonces are known only to the principal who generated it, and retrieving the nonce would require the computation of a discrete logarithm, we say that Bob receives a variable X of a generic message sort instead of g^{N_A} and similarly for Alice. The symbol $*$ is associative and commutative and satisfies the following additional property with respect to exponentiation:

$$(X^Y)^Z = X^{(Y*Z)}$$

The intruder abilities to create, manipulate, and delete messages according to the Dolev-Yao attackerÕs capabilities [13] are described as follows, where we use the special symbol $\_{\in}\mathcal{I}$ to represent that the intruder knows something, and I denotes the intruder's name:

$$\frac{M_1{\in}\mathcal{I},\ M_2{\in}\mathcal{I}}{(M_1 * M_2){\in}\mathcal{I}} \qquad \frac{X{\in}\mathcal{I},\ Y{\in}\mathcal{I}}{X^Y{\in}\mathcal{I}} \qquad \frac{}{N_I{\in}\mathcal{I}}$$

The intruder also knows the names of all the principals and the base g.

If we ask ourselves whether the intruder can learn a message X^{N_A} for some variable X received by Alice (representing the nonce that Alice receives from Bob), the answer is yes for an infinite set of instances for X, e.g., g^{N_I}, $(g^{N_I})^{N_I'}$, $((g^{N_I})^{N_I'})^{N_I''}$, etc. If we take instantiation $X \mapsto g^{N_I}$, the intruder can learn the message $g^{(N_A * N_I)}$ by means of the following sequence of actions (only the three first steps are necessary but we need Alice to complete the protocol in order to believe she is sharing a shared key with Bob):

1. $A \hookrightarrow B : A$
   Alice sends her name to Bob, but it is intercepted by the intruder.
2. $A \hookrightarrow B : B$
   Alice sends Bob's name to Bob, but it is intercepted by the intruder.
3. $A \hookrightarrow B : g^{N_A}$
   Alice creates a new nonce N_A and sends g^{N_A} to Bob, but it is intercepted by the intruder.
4. $I \hookrightarrow A : B$
   The intruder sends Bob's name to Alice.
5. $I \hookrightarrow A : A$
   The intruder sends Alice's name to Alice.
6. $I \hookrightarrow A : g^{N_I}$
   The intruder creates a new nonce N_I and sends g^{N_I} to Alice.

The intruder is able to learn the message $g^{(N_A * N_I)}$ just by raising the intercepted message g^{N_A} to N_I. Note that the intruder does not need to know N_A, since he gets the desired effect thanks to the equational properties for exponentiation and product of exponents described above.

In this example, the commutativity of symbol $*$ and the equational property of exponentiation are the relevant cryptographic properties of the protocol that have to be considered in order to model both the correct execution of the protocol and to find the attack.

In Section 2, we briefly introduce the Maude-NPA tool. In Section 3, we define the protocol specification and show how the attack is found. We also motivate why we are able to find such an attack. In Section 4, we explain the sort propagation algorithm used to filter out unsorted AC-unifiers returned by the CiME tool. We conclude in Section 5.

2 The Maude-NPA

We briefly introduce the Maude-NPA tool. In [15], the reader can find further details for the case of a confluent and terminating equational theory $\Delta \uplus B$ where the set B of axioms is empty. We are currently extending the framework in [15] to the case where the set B of axioms is non-empty and satisfies appropriate requirements such as the existence of a finitary B-unification algorithm. This paper illustrates the use of the framework in an example where B is AC.

In the Maude-NPA, protocols are specified with a notation derived from strand spaces [16]. In a *strand*, a local execution of a protocol by a principal is indicated by a sequence of messages $[msg_1^-,\ msg_2^+,\ msg_3^-, \ldots,\ msg_{k-1}^-,\ msg_k^+]$ where nodes representing input messages are assigned a negative sign, and nodes representing output messages are assigned a positive sign. In Maude-NPA, strands evolve in time and thus we use the symbol $|$ to divide past and future in a strand, i.e.,

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

where $msg_1^{\pm}, \ldots, msg_{j-1}^{\pm}$ are the past messages, and $msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}$ are the future messages ($msg_j^{\pm}$ is the immediate future message).

A *state* is a set of Maude-NPA strands unioned together with an associative and commutativity union operator & along with an additional term describing the intruder knowledge at that point. The *intruder knowledge* is represented using two kinds of facts: positive knowledge facts (the intruder knows m, i.e., $m \in \mathcal{I}$), and negative knowledge facts (the intruder does not know m, i.e., $m \notin \mathcal{I}$), where m is a message expression. Negative facts are essential in our framework to denote terms the intruder will eventually learn in the future, and hence cannot know at the protocol state that we are processing. Negative facts are used by our grammars to describe states unreachable for the intruder; see Section 3.3. The following example illustrates the notion of a state, where we have two strands at different stages of execution, and the intruder does already know the messages g^{N_A}, A, and B, but does not yet know the nonce N_I and the message $(g^{N_I * N_A})$:

$$[\ A^+, B^+, (g^{N_A})^+ \mid B^-, A^-, X^-\ ]\ \&\ [\ nil \mid A^-, B^-, Y^-,\ B^+, A^+, (g^{N_B})^+\ ]\ \&$$
$$\{\ N_I \notin \mathcal{I},\ (g^{N_I * N_A}) \notin \mathcal{I},\ g^{N_A} \in \mathcal{I},\ A \in \mathcal{I},\ B \in \mathcal{I},\ K\ \}$$

Strands communicate between them via the intruder, i.e., by sending a message m to the intruder who will send it back to another strand. When the intruder receives a message, then it learns it, i.e., a message m is learned in a transition from a state with the fact $m \notin \mathcal{I}$ in its intruder knowledge part to a state with the fact $m \in \mathcal{I}$ in its intruder knowledge part (in a forward execution of the protocol). The intruder has the usual ability to read and redirect traffic, and can also perform operations, e.g., encryption, decryption, concatenation, etc., on messages that it has received. Intruder operations are described in terms of the intruder sending messages to itself, which are represented as different strands, one for each action. All intruder and protocol strands are described symbolically, using a mixture of variables and constants, so a single specification can stand for many concrete instances. There is no restriction in the number of principals, number of sessions, nonces, or time, i.e., no data abstraction or approximation is performed. It is also possible to include algebraic properties of the operators (cryptographic and otherwise) as an equational theory and also, as presented in this paper, we can include axioms such as associativity and commutativity with or without identity, or only commutativity; however, associativity without commutativity is problematic because in general it can produce and infinite set of unifiers (see [2]), although in some cases it can be approximated by weaker associative axioms with a finitary unification algorithm (see [14]).

The Maude-NPA's reachability analysis is based on two parameters: a protocol $\mathcal{P}$ represented by strands, and a grammar sequence $\mathcal{G} = \langle G_1, \ldots, G_m \rangle$ used to cut down the search space. Analysis is done in Maude-NPA via backwards narrowing search from an (insecure) goal state SS_{bad} to try to prove or disprove that the insecure state is unreachable from an initial state. States are ($\Delta \uplus B$-)unified with (reversed) rewrite rules describing state transitions via narrowing modulo an equational theory $\Delta \uplus B$. Grammars $\langle G_1, \ldots, G_m \rangle$ represent negative information (or co-invariants), i.e., infinite sets of states unreachable from the initial state. Seed terms $\langle sd_1, \ldots, sd_n \rangle$ represent knowledge that the user believes[5] is not known by the intruder and from which the tool generates the formal languages $\langle G_1, \ldots, G_m \rangle$ (with $m \leqslant n$) representing several infinite sets of states unreachable for the intruder. These grammars are very important in our framework, since in the best case they can reduce the infinite search space to a finite one, or, at least, can drastically reduce the search space.

The tool tries to deduce whether the protocol is safe for SS_{bad} or not. If the protocol is unsafe, Maude-NPA always terminates with an intruder learning sequence and the exchange message sequence, provided enough time and memory resources are available. If the protocol is unsafe, grammars can actually improve the time and memory consumption by reducing the number of states to be analyzed. If the protocol is safe, the algorithm can often prove it, provided the search space is finite. When the

[5] This initial belief from the user may not always be correct, since some exceptions of the form $X \not\sqsubseteq t$, describing that the actual value of variable X in the seed term cannot match the term t with variables, may be added to the seed term. The grammar generation process *guarantees* that the grammars finally generated always describe unreachable states.

protocol is safe, grammars are the key technique for producing a finite search space, since they provide a drastic reduction on the search space so that often an infinite search space is reduced to a finite one. If the protocol is safe but the search space is infinite, Maude-NPA runs forever. This provides a semi-decision algorithm. See [15] for further explanations.

The protocol to be analyzed is provided to the tool as an algebraic signature Σ including symbols, sorts, and subsort information (see [22,8]), together with the set $\mathcal{P}$ of strands defining the protocol. Moreover, the tool expects some *seed terms* $\langle sd_1, \ldots, sd_n \rangle$ for the generation of the grammars $\langle G_1, \ldots, G_m \rangle$ where $m \leqslant n$. In the specification of the protocol-specific signature Σ, there is a special sort Msg for messages. The user will add extra symbols involving the sort Msg. Special algebraic properties of a protocol may be specified with symbols in Σ by means of a set B of equational axioms and a set Δ of equations such that the terms in the axioms B or equations Δ must have sort Msg or a sort smaller than Msg. Note that we can control the level of type checking of the order-sorted protocol specification to fit the attacker model or protocol description (i.e., a completely unsorted protocol specification will be using only the sort Msg for the symbols describing the protocol without any other sort).

In security analyses it is often necessary to use fresh unguessable values, e.g., for nonces. The user can make use of a special sort Fresh in the protocol-specific signature Σ for representing them. The meaning of a variable of sort Fresh is that it will never be instantiated by an E-unifier generated during the backwards reachability analysis. This ensures that if nonces are represented using variables of sort Fresh, they will never be merged and no approximation for nonces is necessary. However, the framework is very flexible, and the user can specify some constant symbols of sort Fresh to play with nonces that can indeed be merged, i.e., approximated. Since variables of sort Fresh are treated in a special way, we make them explicit in the strand definition of the example by writing

$$(r_1, \ldots, r_k : \mathsf{Fresh})\ [msg_1^{\pm}, \ldots, msg_n^{\pm}],$$

where $r_1, \ldots, r_k$ are all the variables of sort Fresh appearing in $msg_1^{\pm}, \ldots, msg_n^{\pm}$.

We impose a requirement on the protocols that can be specified in our tool w.r.t. the algebraic properties specified with a set $E = B \uplus \Delta$ of axioms and equations. Intuitively, a principal can send whatever he/she likes but the pieces of information being processed by the intruder or by a principal are always simplified w.r.t. the oriented version of Δ modulo B. We say a term t is $\rightarrow_{\vec{\Delta}/B}$*-irreducible* if it is a normal form modulo B w.r.t. the oriented version $\vec{\Delta}$ of Δ, i.e., no rule in $\vec{\Delta}$ can be applied to t modulo B. We say that a term t is *strongly* $\rightarrow_{\vec{\Delta}/B}$*-irreducible* if for any substitution σ and variable x such that $\sigma(x)$ is $\rightarrow_{\vec{\Delta}/B}$-irreducible for each x in the domain of σ, then $\sigma(t)$ is $\rightarrow_{\vec{\Delta}/B}$-irreducible. Then, the requirement that we impose on the protocols is that all messages appearing in the reachability and grammar generation stages have to be strongly $\rightarrow_{\vec{\Delta}/B}$-irreducible, except for output messages in a strand (i.e., m^+) and positive knowledge facts (i.e., $m{\in}\mathcal{I}$); see [15] for details.

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

3 Finding Attacks by Equational Reasoning

3.1 Specification of the Diffie-Hellman Example

Continuing with Example 1, we can denote nonce N_X by $n(X, r)$, where r is a variable of sort Fresh. The generator used as a base is denoted by the symbol g. Raising a message M to the power of an exponent X is denoted by $exp(M, X)$. And the product of exponents X_1, X_2 is denoted by $X_1 * X_2$. The names of all the principals are fixed using constants a and b, since they are just roles. The name of the intruder is denoted by constant i. The order-sorted signature Σ defining the Diffie-Hellman based protocol is the following:

$$a : \to \mathsf{Name} \qquad b : \to \mathsf{Name} \qquad i : \to \mathsf{Name}$$
$$n : \mathsf{Name} \times \mathsf{Fresh} \to \mathsf{Nonce} \qquad g : \to \mathsf{Gen}$$
$$exp : \mathsf{Gen{\vee}Exp} \times \mathsf{NeNonceSet} \to \mathsf{Exp} \qquad exp : \mathsf{Gen} \times \mathsf{NeNonceSet} \to \mathsf{Exp}$$
$$\_{*}\_ : \mathsf{NeNonceSet} \times \mathsf{NeNonceSet} \to \mathsf{NeNonceSet}$$

together with the following subsort relations

$$\mathsf{Name}\ \mathsf{NeNonceSet}\ \mathsf{Gen{\vee}Exp} < \mathsf{Msg}$$
$$\mathsf{Nonce} < \mathsf{NeNonceSet} \qquad \mathsf{Gen}\ \mathsf{Exp} < \mathsf{Gen{\vee}Exp}$$

Algebraic properties are described using the following AC axioms B and equations Δ in E:

$$B = \{\ (X * Y) * Z = X * (Y * Z),\ (X * Y) = Y * X\ \}$$
$$\Delta = \{\ exp(exp(W, Y), Z) = exp(W, Y * Z)\ \}$$

where X, Y, Z are variables of sort $\mathsf{NeNonceSet}$ and W is a variable of sort Gen. Note that the sort of variable W has to be Gen to have a finitary unification algorithm, as discussed in Section 3.2 below. We omit the identity property for symbol $*$ because it is not relevant for this example, although we can handle it as explained in Section 4. In the following, we omit sorts of variables whenever there is no possible confusion. Variables are written in uppercase, except for variables r, r', r'' of sort Fresh.

The strands $\mathcal{P}$ associated to the three protocol steps shown in Example 1 are as follows, one for each principal (or role) in the protocol:

(s1) $(r : \mathsf{Fresh})\ [\ a^+,\ b^+,\ exp(g, n(a, r))^+,\ b^-,\ a^-,\ X^-\ ]$
This strand denotes principal Alice sending her name, Bob's name, and the generator g raised to the power of a new nonce generated by Alice using the Fresh variable r. Then, Alice waits for Bob's name, her name, and an unknown message X.

(s2) $(r' : \mathsf{Fresh})\ [\ a^-,\ b^-,\ Y^-,\ b^+,\ a^+,\ exp(g, n(b, r'))^+\ ]$
This strand denotes principal Bob waiting for Alice's name, his name, and an unknown message Y. Then, Bob sends his name, Alice's name, and the generator g raised to the power of a new nonce generated by Bob using the Fresh variable r'.

The following strands describe the intruder abilities according to the Dolev-Yao attacker's capabilities [13]. Note that the intruder cannot extract information from either an exponentiation or a product of exponents, only compose them.

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

(s5) $[M_1^-, M_2^-, exp(M_1, M_2)^+]$ Exponentiation

(s6) $[g^+]$ Generator

(s7) $[A^+]$ All names are public

(s8) $(r'' : \mathsf{Fresh})\ [\ n(i, r'')^+\ ]$ Generation of its own nonces

The strongly $\rightarrow_{\vec{\Delta}/B}$-irreducibility requirement implies that if a message of the form $exp(X, Y)$ where X is of sort Exp appears in an input message of a protocol strand or as a negative knowledge fact $\_\notin\mathcal{I}$ of the intruder's knowledge, it must be split into two cases (corresponding to two different protocol states) the message $exp(X, Y)$, which is restricted to strongly $\rightarrow_{\vec{\Delta}/B}$-irreducibility, and the message $exp(g, X' * Y)$, where substitution $\{X \mapsto exp(g, X')\}$ is propagated.

3.2 A Finite Unification Algorithm for $B \uplus \Delta$

We explain why narrowing modulo B provides a finitary unification algorithm for the theory $B \uplus \Delta$, allowing us to automate the backwards reachability analysis. Note that the theory $B \uplus \Delta$ is closely related to a fragment of the theory of exponentiation given in [20] and to larger exponentiation theories in [19], for which finitary unification algorithms are known [20,19]. However, instead of using those algorithms, we adopt the modular hybrid approach described in Section 1 and perform narrowing with Δ modulo B as a $B \uplus \Delta$-unification procedure. We say that $B \uplus \Delta$ is "closely related" to a fragment of the theories in [20,19], because these exponentiation theories are *unsorted*, whereas $B \uplus \Delta$ is an *order-sorted* theory, which turns out to be crucial for narrowing with Δ modulo B to terminate.

The key point is that the term $exp(W, Y * Z)$ is a *constructor term* in the order-sorted sense, but obviously *not* a constructor term in the unsorted sense. A *constructor term* in the unsorted sense is a term built up with only constructor symbols and variables. Given a set of rewrite rules $l_1 \rightarrow r_1, \ldots, l_n \rightarrow r_n$ over an unsorted signature Σ, a function symbol f in Σ is called a *constructor* if it does not appear as the root symbol of any left-hand side $l_1, \ldots, l_n$. In an order-sorted context this notion is more subtle, since the symbol f can be overloaded and can be a constructor (in the sense that no rules apply to it) for some typings and a defined symbol for other typings. That is, the *symbol* f can indeed appear in a lefthand side l_i, but it should never be possible to *type* that lefthand side, or any of its well-sorted substitution instances $\theta(l_i)$, with any of the constructor versions of f. In our case, exp is an overloaded function symbol, for which the typing $exp : \mathsf{Gen} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp}$ is indeed a constructor operator in the order-sorted sense, since any well-typed substitution instance of Δ's lefthand side $exp(exp(W, Y), Z)$ must necessarily have for its top symbol the typing $exp : \mathsf{Gen}{\vee}\mathsf{Exp} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp}$, and can never have the typing $exp : \mathsf{Gen} \times \mathsf{NeNonceSet} \rightarrow \mathsf{Exp}$. In particular, $exp(W, Y * Z)$ is a constructor term as claimed.

The next key observation is that Δ is confluent, terminating, and coherent modulo B (see [18] for these notions). Coherence modulo B is particularly easy to check, since the associative-commutative multiplication symbol in B does not appear in Δ's lefthand side $exp(exp(W, Y), Z)$. We can then use Theorem 5 in [18] to ensure that narrowing with Δ modulo B provides a complete $B \uplus \Delta$-unification algorithm; and Theorem 9 in [18], plus the fact that $exp(W, Y * Z)$ is a constructor term, to further conclude that narrowing with Δ modulo B terminates and therefore such a $B \uplus \Delta$-unification algorithm is finitary.

For all this to work we need the Maude-NPA to support *order-sorted* unification modulo B. How this is achieved, not only for the above theory B, but for any combination of associativity, commutativity, and identity axioms (except for associativity without commutativity) is explained in Section 4.

3.3 Reducing the Size of the Search Space

Grammars representing several infinite sets of states unreachable for the intruder are very important in our framework, since they represent negative information (co-invariants) that will be used to cut down the search space.

As explained in [15], the Maude-NPA starts out with the seed terms, which represent knowledge that the user believes is not known by the intruder. There are only two types of seed terms: (i) $\varnothing \mapsto t{\in}\mathcal{L}$, denoting that the term t of sort Msg is unknown for the intruder without any restriction, and (ii) $t|_p{\notin}\mathcal{I} \mapsto t{\in}\mathcal{L}$, denoting that the term t of sort Msg is unknown for the intruder provided that the subterm $t|_p$ is certainly unknown by the intruder, i.e., provided that the fact $t|_p{\notin}\mathcal{I}$ appears in the intruder's knowledge of the state of the protocol that we will be processing at each moment. The initial grammar for Example 1 containing four language productions is as follows:

$$\varnothing \mapsto X * n(a,r){\in}\mathcal{L} \qquad \varnothing \mapsto n(a,r){\in}\mathcal{L} \qquad \varnothing \mapsto X * n(b,r){\in}\mathcal{L} \qquad \varnothing \mapsto n(b,r){\in}\mathcal{L}$$

For instance, language production $\varnothing \mapsto X * n(a,r){\in}\mathcal{L}$ denotes a formal language including any message $t_1 * t_2$ such that subterm t_1 is of sort $\mathsf{NeNonceSet}$ and subterm t_2 is of the form $n(a,r)$ where a is the constant denoting Alice and r is a variable. The final grammar turns out to be the same as the initial grammar, and thus we omit the details about the grammar generation, which can be found in [15]. Therefore, any message m belonging to the formal language denoted by $\varnothing \mapsto X * n(a,r){\in}\mathcal{L}$ is unknown for the intruder and any state containing m as an input message (i.e., m^-) or as part of the intruder knowledge (i.e., $m{\in}\mathcal{I}$) is unreachable for the intruder and discarded.

In this work we also make use of an additional state-reduction feature, namely a notion similar to the "lazy intruder" of Basin et al. [3], but for backwards instead of forward search. This feature was already implemented in the original NRL Protocol Analyzer [21] and is independent of the use of any equational theory. Note that this feature is an *additional* state-reduction feature complementing the state reduction provided by grammars, but both features are useful.

We give the main intuitions but do not explain this feature in detail; a detailed explanation will appear elsewhere. The key idea behind our lazy intruder is to refrain from searching for terms that we know can easily be found by the intruder. At the same time, if the term becomes later instantiated to something that is not so readily obtainable, then we rollback to the state in which the term first appeared in the intruder knowledge. Terms that the intruder can easily find include variable terms, publicly known terms, and terms which an intruder could compute from variables and publicly known terms. The expression $exp(g, X)$ in our Diffie-Hellman protocol is an example of such a lazy term, since g is publicly known, X is a variable, and exp is computable by the intruder.

3.4 Backwards Reachability Analysis

The final state we are looking for is one in which Alice receives Y as the final message and computes $exp(Y, n(a, r))$, while the intruder also learns $exp(Y, n(a, r))$. As explained in Section 3.1, we actually specify two final attack states, one in which the intruder learns $exp(Y, n(a, r))$, and one in which the intruder learns $exp(g, X * n(a, r))$, where both terms are strongly $\rightarrow_{\vec{\Delta}/B}$-irreducible. Since Y is of type *Gen* or *Exp*, the first case is irreducible only when $Y = g$. This corresponds to the case in which the intruder sends g to a, which can be detected and discarded by a. We treat the second case in more detail below.

The *final attack state pattern* to be given as input to the system is:

$$(r : \mathsf{Fresh})\ [\ a^+,\ b^+,\ exp(g, n(a, r))^+,\ b^-,\ a^-,\ exp(g, X)^-\ |\ nil\ ]\ \&$$
$$\{\ exp(g, X * n(a, r)) {\in} \mathcal{I},\ \ K\ \}$$

The intruder knowledge is essential in the backwards reachability process and thus variable K will be appropriately instantiated by narrowing. Using the above attack state pattern, the grammar with four language productions, and the lazy intruder feature described above, our tool is able to find the following initial state of the protocol:

$$(r : \mathsf{Fresh})\ [\ nil\ |\ a^+,\ b^+,\ exp(g, n(a, r))^+,\ b^-,\ a^-,\ exp(g, X)^-\ ]\ \&$$
$$[\ nil\ |\ exp(g, n(a, r))^-,\ X^-,\ exp(g, X * n(a, r))^+\ ]\ \&$$
$$\{\ exp(g, n(a, r)) {\notin} \mathcal{I},\ exp(g, X * n(a, r)) {\notin} \mathcal{I}\ \}$$

Note that the second strand is, indeed, an intruder strand introduced by the backwards reachability process denoting that whenever the intruder knows the message $exp(g, n(a, r))$ (variable r is bound in this context) and knows any message X of sort NeNonceSet, he can combine them and generate message $exp(g, X * n(a, r))$. Intruder strands generating messages b^-, a^-, and $exp(g, X)^-$ are not included in the initial state, since the lazy intruder avoids searching for them. The previous state is an initial state of the protocol because all the strands are in their initial position (i.e., every message is in the future), and the intruder does not know but eventually will learn messages exchanged in the protocol run. The concrete message exchange sequence leading to this attack is:

$$a^+ . b^+ . exp(g, n(a, r))^+ . exp(g, n(a, r))^- . X^- . exp(g, X * n(a, r))^+ . b^- . a^- . exp(g, X)^-$$

which corresponds to the intuitive description of the attack explained in Section 1.2. Actually, the tool returns an infinite number of attacks, since X can also be the product of any number k of intruder nonces; one attack is returned for each positive k.

4 Order-sorted Unification Modulo Equations

As the protocol is specified using an order-sorted theory with AC axioms, narrowing requires an order-sorted AC unification procedure. In fact, the use of order-sorted features is essential for AC-narrowing to terminate with the given protocol's equations Δ. One major challenge in developing such a procedure is that producing an efficient AC unification procedure takes considerable effort, yet existing tools supporting AC unification such as CiME [11] only support *unsorted* theories rather than order-sorted theories.

Our solution has been to partition order-sorted AC unification into two steps: (1) we drop the sort information and use CiME [11] to compute a complete set of unsorted unifiers; and (2) we apply a *sort propagation* algorithm, described below, which constructs zero or more order-sorted unifiers from each unsorted unifier. Our approach is not restricted only to AC unification, since we allow any combination of associativity, commutativity, and identity, except associativity without commutativity.

Although theoretically there may be many more order-sorted unifiers than unsorted unifiers, in practice we have found that the opposite is usually the case. Many unsorted unifiers can be eliminated when considering sort constraints, because a variable is bound to a term with an incompatible type. For example, we can eliminate an unsorted unifier θ which binds a variable `X:Nonce` to a term `(Y:Nonce ; Z:Nonce)` representing a nonce set.

Due to the fact that the unsorted unification procedure ignores sort information and the sort propagation algorithm ignores the equations, the order-sorted theory $\mathcal{E} = (\Sigma, B)$ where $\Sigma = (S, F, \leqslant)$ is an order-sorted signature with poset of sorts $(S, \leqslant)$ and function symbols $F = \{F_{w,s}\}_{(w,s)\in S^* \times S}$, is required to satisfy several requirements:

- Each connected component $[s] \in S/\equiv_{\leqslant}$, where $\equiv_{\leqslant}$ is the equivalence generated by the subsort ordering $\leqslant$, must have a top-most sort $\top_{[s]}$;
- Each axiom $l = r \in B$ must be *sort-preserving* and each variable in $\mathcal{V}ar(l) \cup \mathcal{V}ar(r)$ must have a top-most sort; and
- Each term $t \in T_{\Sigma}(X)$ must have a unique *least sort* s with $t \in T_{\Sigma}(X)_s$.

These requirements are not too restrictive for our purposes. We can guarantee that each commutativity and identity equation is sort-preserving by introducing extra sorts and operators declarations to complete the theory. The other requirements are already required by Maude for efficiency purposes, and are checked automatically by Maude.

4.1 Unsorted Unification

As a first step, the sorts from the order-sorted signature $\Sigma = (S, F, \leqslant)$ are dropped to obtain an unsorted signature $\overline{F}$, where each operator $f \in F_{s_1,\ldots,s_n,s}$ is mapped to an unsorted and disambiguated operator $f_{\top_{s_1},\ldots,\top_{s_n},\top_s}$ where its original connected components are made explicit. Given the unification problem $t =_B u$, we rename the operators in t and u and make the variables unsorted in a set $\overline{X}$ to obtain unsorted terms $\overline{t}, \overline{u} \in T_{\overline{F}}(\overline{X})$. We then pass the unsorted unification problem $\overline{t} =_{\overline{B}} \overline{u}$ to CiME. The unsorted most-general unifiers modulo $\overline{B}$, denoted $\overline{\theta}_1, \ldots, \overline{\theta}_n$, are obtained from CiME, and we reverse the renaming process to obtain mappings $\theta_1, \ldots, \theta_n : X \to T_\Sigma(X)$. At this point, each mapping θ_i is not necessarily an order-sorted unifier, because variables may be bound to terms that have a sort incompatible with the sort of the variable. To remedy this situation and obtain a complete set of order-sorted unifiers, we pass each mapping θ_i to the sort propagation algorithm described below, and take the union of all the order-sorted unifiers returned for each unsorted unifier.

4.2 Sort Propagation

The *sort propagation algorithm* generates a complete set of order-sorted unifiers $\{\theta_1, \ldots \theta_n\}$ from a given mapping $\theta : X \to T_\Sigma(X)$ such that any possible order-sorted unifier ψ to the problem $t =_B u$ that is an instance of the mapping θ is an instance of one of the unifiers θ_i generated by the algorithm. The algorithm maintains a disjunctive set $D = \{ C_0, \ldots C_m \}$ of *sort constraint problems* where each C_i is a conjunction $(t_1 : s_1) \wedge \cdots \wedge (t_{n_i} : s_{n_i})$. A *solution* to C_i is a substitution $\phi : X \to T_\Sigma(X)$ such that $t_j \phi \in T_\Sigma(X)_{s_j}$ for $j \leqslant n_i$. A solution to D is a substitution ϕ that is a solution to at least one problem in the set. The correctness of the algorithm is obtained by observing that although the problems change while the algorithm executes, the set of solutions to the problems does not.

From θ, we construct the initial set $D_0 = \{(\theta(x_1) : s_1) \wedge \cdots \wedge (\theta(x_n) : s_n)\}$, where $x_1, \ldots x_n \in X$ are all the variables such that $\theta(x_i) \neq x_i$ and s_i is the sort of the variable x_i for $i \leqslant n$. We then exhaustively apply the transformation rules in Fig. 1 on D_0 to obtain disjunctive sets $D_1, D_2, \ldots, D_r$. This process terminates, because each application either reduces the number of operator symbols appearing in a problem or reduces the total number of predicates in a problem. In the final set $D_r = \{ C_1, \ldots, C_n \}$, each problem C_i has the form $(x_1 : s_1) \wedge \cdots \wedge (x_{n_i} : s_{n_i})$ where a variable $x \in X$ appears in a most one predicate $(x_i : s_i)$. From each C_i, we generate a sort specialization mapping $\phi_i : X \to X$ that maps each variable x_j with $j \leqslant n_i$ to a fresh variable with sort s_j. The final set of order-sorted unifiers generated by θ is then the set $\{\theta_1, \ldots, \theta_m\}$ where $\theta_i = \phi_i \circ \theta$.

The final set of order-sorted unifiers is complete, because each order-sorted unifier ϕ can be seen as an instance of a mapping derived from an unsorted unifier $\overline{\theta}$ returned by CiME, and a sort specialization mapping ϕ_i. The latter can be seen by observing that each application of an inference rule in Fig. 1 preserves the set of solutions under the assumptions that the equations B are sort-preserving and that each term $t \in T_\Sigma(X)$ has a least sort.

5 Conclusions

In this paper we have illustrated the extension of the Maude-NPA to reason about protocols whose equational theories are given by confluent and terminating equations Δ modulo some axioms B, using a Diffie-Hellman example where $B = AC$. Furthermore, the use of order-sorted theories has been shown to be important in obtaining unification algorithms for $\Delta \uplus B$. Although at present, the Maude-NPA can handle

$$\text{Valid} \quad \frac{\{\, (t : s) \wedge C \,\} \cup D}{\{\, C \,\} \cup D} \quad \text{if } t \in T_{\Sigma}(X)_s$$

$$\text{Conjoin} \quad \frac{\{\, (t : s_1) \wedge (t : s_2) \wedge C \,\} \cup D}{\bigcup_{s \in \mathrm{glb}(s_1, s_2)} \{\, (t : s) \wedge C \,\} \cup D}$$

$$\text{Prop.} \quad \frac{\{\, (f(t_1, \ldots, t_n) : s) \wedge C \,\} \cup D}{\bigcup_{s_1 \ldots s_n \in \sup(\mathrm{ar}(f, s))} \{\, (t_1 : s_1) \wedge \cdots \wedge (t_n : s_n) \wedge C \,\} \cup D}$$

$$\text{where} \quad \begin{aligned} \mathrm{glb}(s_1, s_2) &= \sup(\{\, s \in S \mid s \leqslant s_1 \wedge s \leqslant s_2 \,\}), \\ \mathrm{ar}(f, s) &= \{\, w \in S^* \mid f \in F_{w,s} \,\}, \text{ and} \\ \sup(W) &= \{\, w \in W \subseteq S^* \mid (\forall w' \in W)\, w \leqslant w' \Rightarrow w = w' \,\}. \end{aligned}$$

Fig. 1. Sort Propagation Transformation Rules

order-sorted theories $\Delta \uplus B$ where B can contain any combination of associativity, commutativity, and identity (except associativity without commutativity), much work remains ahead, such as: (i) building in order-sorted unification algorithms for efficiency purposes; (ii) improving grammar formalisms to reduce the search space in the modulo B case; (iii) developing additional optimization techniques; and (iv) developing other case studies involving other equational theories.

References

1. Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *CSFW*, pages 62–76. IEEE Computer Society, 2005.
2. F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1 of *Volume*, chapter 8, pages 445–532. Elsevier Science, 2001.
3. David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
4. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2007. To appear.
5. Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Handling algebraic properties in automatic analysis of security protocols. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings*, volume 4281 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2006.
6. Sergiu Bursuc, Hubert Comon-Lundh, and Stéphanie Delaune. Associative-commutative deducibility constraints. In Wolfgang Thomas and Pascal Weil, editors, *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 634–645, Aachen, Germany, February 2007. Springer.
7. Yannick Chevalier, Ralf Küsters, Michael Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In 23rd *Conference on Foundations Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 124–135, 2003.

8. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.
9. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive-or. In 18^{th} *Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 271–280, 2003.
10. Hubert Comon-Lundh and Véronique Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In R. Nieuwenhuis, editor, *Proc. of 14th International Conference on Rewriting Techniques and Applications, RTA'03*, volume 2706, pages 148–164, 2003.
11. E. Contejean, C. Marché, B. Monate, and X. Urbain. Proving termination of rewriting with CiME. In A. Rubio, editor, *6th International Workshop on Termination*, Techreport DSIC II/15/03, pages 71–73. Universidad Politécnica de Valencia, 2003.
12. Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis in presence of a homomorphism operator and *exclusive or*. In Michele Buglesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP'06) — Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 132–143, Venice, Italy, July 2006. Springer.
13. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transaction on Information Theory*, 29(2):198–208, 1983.
14. Santiago Escobar, Catherine Meadows, and José Meseguer. Equational cryptographic reasoning in the Maude-NRL Protocol Analyzer. In *Proc. of the First International Workshop on Security and Rewriting Techniques (SecReT 2006)*, Electronic Notes in Theoretical Computer Science. Elsevier Sciences Publisher, 2006. To appear.
15. Santiago Escobar, Catherine Meadows, and José Meseguer. A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.
16. F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.
17. Jean Goubault-Larrecq, Muriel Roger, and Kumar Neeraj Verma. Abstraction and resolution modulo AC: How to verify Diffie-Hellman-like protocols automatically. *Journal of Logic and Algebraic Programming*, 64(2):219–251, 2005.
18. J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *Proc. ICALP'83*, pages 361–373. Springer LNCS 154, 1983.
19. Deepak Kapur, Paliath Narendran, and Lida Wang. An E-unification algorithm for analyzing protocols that use modular exponentiation. In R. Nieuwenhuis, editor, *Proc. of 14th International Conference on Rewriting Techniques and Applications, RTA'03*, volume 2706, pages 165–179, 2003.
20. C. Meadows and P. Narendran. A unification algorithm for the group Diffie-Hellman protocol. In *Proc. Workshop on Issues in the Theory of Security WITS 2002*, 2002.
21. Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of logic programming*, 26(2):113–131, 1996.
22. José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.
23. J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *Proceedings of the IEEE Computer Security Foundations Workshop (CSFW-16)*, pages 47–61, 2003.

Local Deduction of Trust

Anders Moen Hagalisletto[1] and Olaf Owe[2]

[1] Birkeland Innovation and Norwegian Computing Center
[2] Department of Informatics, University of Oslo

Abstract. Simulators and analyzers for protocols do not distinguish between the local 'subjective' view of the agents and the global 'objective' view of the analysis perspective. In practice this means that security analysis in general and protocol analysis in particular do neither model the agents local beliefs nor their local deduction power in a satisfactory way. This paper suggests a solution to the problem by proposing a new approach to epistemic logic, explained in terms of term rewriting, in order to make agents capable of performing logical deductions themselves. Local deductions of security properties, like trust, are crucial to assure that the agents can expand their knowledge about the environment they inhabit.

1 Introduction

Languages of epistemic logic ([10], [13], [5]) have been used to express security properties ([2], [7]) and to analyze protocols ([1], [3], [4], [11], [12]). Yet, it is not clear from any of the standard approaches how beliefs and knowledge should be interpreted in the context of real agents – as the semantics of the systems are typically rather abstract. The typical approaches takes a global view on the modeling and analysis of security critical system. Reachability analysis is usually performed by generating executions in terms of sequences of protocol events, while theorem proving approaches deploy a high level and "ideal" perspective on the epistemic operators. Hence if an agent is equipped with a classical deduction engine, the agent would try to prove infinitely many theorems. In practice this would mean that the agent would use most of its computation resources proving theorems. This is a problem, of course, since the applications and protocols call for concrete answers to real problems. Real agents do not have unlimited reasoning power, and their memory is always bounded. In other words: agents are finitary. Moreover, most of the theorems that are proven in classical normal epistemic logic is entirely irrelevant to agents acting in a security critical environment. Abstract denotational semantics that interprets axiomatizations of *idealized* epistemic capabilities is inappropriate to understand agency. We turn the problem upside down: The belief operator is given a concrete operational interpretation, therefore the epistemic axioms are revised according to the practical needs of the agent. Thus the logic and its semantics approach actual agent behaviour. If agents perform logical deductions *locally*, then two aspects become important: first that the deductions terminate as efficient as possible and second that agents are focused towards obtaining useful beliefs. Since the life-cycle of agents are executed within a tool, termination of the local deductions is a particular important problem to solve.

Trust is one important example of a security property that involves local deductions. *Trust* is currently a large research field. We are only concerned with the straightforward communication-oriented interpretation, introduced by the BAN logic [3].

An agent a trusts an agent b, if a believes whatever b says is correct.

The part "whatever b says" includes the reference to a second order quantifier. Hence if agent b sends a message to a saying F, then a should be able to infer that a believes F, whatever the sentences F contains.

The paper addresses the following questions: Is it possible to build agents that are capable of making second order inferences? Are there efficient ways to do so? Fortunately the answer is yes, although problems about termination pop up at every corner. A tiny second order language is introduced in Section 2, in order to formalize the notion of trust and other security properties. Then in Section 3, an operational semantics is given in rewriting logic. In Section 4 we explore desirable epistemic inference schemes in the context of term rewriting, and revised temporal versions of the standard axioms are presented.

The minimal logical deduction system presented is the logical kernel of the agents in the simulator PROSA, as depicted in Figure 1. PROSA [8] is a framework for the specification and static and dynamic analysis of protocols. PROSA is implemented in Maude, which is based on rewriting logic. In this paper we mainly consider the modules Local deduction, and Communication, as depicted in Figure 1.

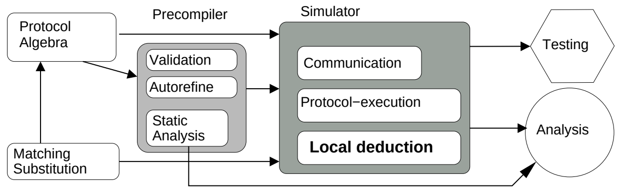

Fig. 1. Architecture of PROSA.

The concrete and operational interpretation of the belief operator also put new light on epistemic logic. Epistemic logic has been devoted to the investigation of two main interpretations: *implicit* or *ideal* beliefs and the notion of *explicit* or *concrete* beliefs. Neither of these approaches have explored real practical agents performing local deductions.

Finally the investigation served as a successful example of *method integration*, term rewriting is used to calibrate logical axioms. Unlike common approaches, the belief operator is given a concrete interpretation: The deduction engine is focused on the agents capability of performing effective deductions.

2 Formal specification of security properties

In this section we define a language for specifying security properties based on temporal epistemic logic, and give some examples of formalizations of security properties like trust, confidentiality and honesty.

2.1 The formal specification language

A second order language [14] for specifying security properties, denoted $\mathscr{L}_S$, can be defined as follows: The *agent terms* include *agent names* and *agent variables.* The agent terms are typically written a, b, c, while the agent *variables* are denoted $x, y, x_1, x_2, \ldots$. Second order variables X, Y are place-holders for arbitrary sentences.

Definition 1. *Let a and b be agent terms, x an agent variable, and X a sentence variable. $\mathscr{L}_S$ is the least language s.t.:*

(i)	$a = b$, $\mathsf{Agent}(a)$, $X \in \mathscr{L}_S$.	
(ii)	*If φ, ψ, $\Phi \in \mathscr{L}_S$, then so are the following:*	
	$\neg\varphi$, $\varphi \to \psi$,	*propositional logic*
	$\mathsf{Transmit}(a, b, \varphi)$,	*"a sends φ to b"*
	$\mathsf{Bel}_a(\varphi)$,	*"a believes φ"*
	$\varphi \mathcal{U} \psi$,	*"φ holds until ψ holds"*
	$\forall x\, \varphi$,	*1. order quantification*
	$\forall X\, \Phi$.	*2. order quantification*

Even though the language appears to be very expressive, due to the second order variables and quantifiers, there are some strict limitations: Unlike λ-calculus, general functional application in $\mathscr{L}_S$ is not permitted, like $X(y)$ or $X(Y)$. Since a sentence Φ might contain second order variables, an $\forall X$ instance Φ^{inst} might have degree larger than the original sentence Φ. We say that:

Definition 2. *The minimal syntactic complexity of a sentence φ, denoted* $\deg(\varphi)$*, is defined by obvious recursion:*

(i) $\deg(a = b) = \deg(\mathsf{Agent}(a)) = \deg(X) = 1$
(ii) $\deg(\neg\varphi) = \deg(\mathsf{Bel}_a(\varphi)) = \deg(\mathsf{Transmit}(a, b, \varphi)) = \deg(\varphi) + 1$
(iii) $\deg(\varphi \to \psi) = \deg(\varphi \mathcal{U} \psi) = \max(\varphi, \psi) + 1$
(iv) $\deg(\forall x\, \varphi) = \deg(\forall X\, \Phi) = \deg(\varphi) + 1$

Since there are both first and second order variables, two substitution operators are needed. First order substitution, $\mathrm{sub}(t, x, e)$ reads "replace the variable x with the term t in the expression e". The second order substitution function, denoted $\mathbf{sub}(F, X, \Phi)$, replaces every occurrence of the sentence variable X in the sentence Φ by the sentence F. Both $\mathrm{sub}(t, x, \Phi)$ and $\mathbf{sub}(F, X, \Phi)$ are defined by obvious recursion on $\deg(\Phi)$. Although it is possible to write sentences in $\mathscr{L}_S$ with free first or second order variables, it is prohibited in specifications. Substitution over quantifiers is then defined by $\mathrm{sub}(t, x, \forall y\, \Phi) = \forall y\, (\mathrm{sub}(t, x, \Phi))$ if $x \neq y$ and $\mathrm{sub}(t, x, \forall x\, \Phi) = \forall x\, \Phi$. The definition for second order substitution is similar. The function $\mathrm{SUB}(S, \Phi)$ recursively substitutes a set S consisting of pairs of agent variables and terms $\langle t, x\rangle$, and sentence variables and sentences $\langle F, X\rangle$ into the sentence Φ: It is defined by $\mathrm{SUB}(\varnothing, \Phi) = \Phi$, $\mathrm{SUB}(\langle t, x\rangle \cup S, \Phi) = \mathrm{SUB}(S, \mathrm{sub}(t, x, \Phi))$, $\mathrm{SUB}(\langle F, X\rangle \cup S, \Phi) = \mathrm{SUB}(S, \mathbf{sub}(F, X, \Phi))$.

The propositional connectives $\wedge$ and $\vee$ are definable from $\neg$ and $\to$, and $a \neq b$ iff $\neg(a = b)$. Let G denote a finite set of actual agent names $G = \{a_1, \ldots, a_n\}$, and x be an agent variable. The *next* and *future* operators are definable from until by the equations $\bigcirc\varphi \overset{\text{def}}{\longleftrightarrow} \bot\,\mathcal{U}\,\varphi$ and $\boxed{\mathsf{F}}\,\varphi \overset{\text{def}}{\longleftrightarrow} \top\,\mathcal{U}\,\varphi$ (in the proper future), $\boxed{\cdot}\,\varphi \overset{\text{def}}{\longleftrightarrow} \varphi \wedge \boxed{\mathsf{F}}\varphi$ (now and always in the future). An *epoch* is defined as an interval in discrete time, in which a given sentence φ holds, hence epoch is an operator definable within $\mathscr{L}_S$ by $\mathbb{E}_n^m\,\varphi \overset{\text{def}}{\longleftrightarrow} \bigwedge_{i=n}^{n+m} \bigcirc^i \varphi$, which means that the sentence φ holds from time n to time $n+m$.

2.2 Security properties

Although the core language $\mathscr{L}_S$ presented in definition 1 is small, it has the power to express a large set of high level properties, both properties about single agents, and relations between agents [9].

Trust The sentence "agent a trusts agent b", which means that the agent a believes everything that b says, is denoted $\text{Trust}(a,b)$. We say that a is the *subject* (trustor) in the relation and b is the *object* (trustee) in the relation. A standard formalization of the relation is:

$$\text{Trust}(a,b) \overset{\text{def}}{\longleftrightarrow} \forall X\,(\mathsf{Transmit}(b,a,X) \to \mathsf{Bel}_a(X)).$$

The agent b might not know or believe that a stands in this relation to her. If b believes this is the case this reads $\mathsf{Bel}_b(\text{Trust}(a,b))$. If a does not trust b, we say that a lacks trust in b, formally $\mathsf{Bel}_a(\neg\text{Trust}(a,b))$.

Confidentiality An important concept in computer security is the notion of *confidentiality.* A fact φ is *contingent secret* for a group of agents G, written $\text{CSecret}(G,\varphi)$, iff every agent not in the group does not possess φ, formally:

$$\text{CSecret}(G,\varphi) \overset{\text{def}}{\longleftrightarrow} \forall x(\mathsf{Agent}(x) \wedge x \notin G \to \neg\,\mathsf{Bel}_x\,(\varphi)).$$

Yet contingent secrecy does not reveal much of the true nature of secrecy, a sentence φ might be contingent secret by chance. It is more common that given facts are secret within a epoch, $\mathbb{E}_n^m\,\text{CSecret}(G,\varphi)$, that is, φ is confidential with respect to the group G from time n to time $n+m$. An even stronger notion is obtained by requiring that a given fact φ remains secret in the future, formalized by $\boxdot\text{CSecret}(G,\varphi)$.

Honesty An agent is *honest* if the agent believes everything it says:

$$\text{Honest}(a) \overset{\text{def}}{\longleftrightarrow} \forall X\,\forall y\,(\mathsf{Transmit}(a,y,X) \to \mathsf{Bel}_a(X)).$$

Honesty is an example of a *character.* A set of characters constitute a *personality*, in other words: the attitude of the agent.

3 Operational semantics for the security language

Agents communicate with other agents over a network. A message in the network consists of a message content m, the sender's name and the receivers name. If a and b are agent names, and m is the message content we write $\mathsf{msg}\ m\ \mathsf{from}\ a\ \mathsf{to}\ b$. The entities *agents* and *messages* are the only inhabitants in the network model. We introduce the *parallel operator* $\|$, and use the notation $o_1 \| o_2 \| \ldots \| o_n$, to express that the entities $o_1, o_2, \ldots, o_n$ coexist concurrently. A rewrite rule $t \longmapsto t'$ can be interpreted as a local transition rule allowing an instance of the term t to evolve into the corresponding instance of the pattern t'. Each rewrite rule describes how a part of a configuration can evolve in one transition step. A *configuration* is a snapshot of a dynamic system evolving. The parallel operator is an Abelian monoid over the set of configurations with an identity element (the empty configuration). The agent is a structure containing four slots:

$$\prec \underbrace{\mathsf{id}}_{\text{agent name}} \mid \underbrace{\mathsf{bel},}_{\text{set of sentences}} \underbrace{\mathsf{pers},}_{\text{personality}} \underbrace{\mathsf{in},\ \mathsf{out}}_{\text{buffers}} \succ$$

where id denotes the *identity* or *name* of the agent, while bel denotes its current *set of beliefs*. In the paper we also consider the interpretation of beliefs as a *bag* of formulas (multiple occurences are permitted), and show how bags and contraction of beliefs do not have any undesirable effects. The beliefs are interpreted extensionally, that is informally we say that $\mathsf{Bel}_{\mathsf{id}}(\varphi)$ if and only if $\varphi \in \mathsf{bel}$. The variable denoted in represents the *in-buffer* - messages waiting for processing in protocol, while out denotes the *out-buffer* - messages intended for transmission.

The *personality* of an agent is given by a set of *characters. Honesty*, defined in section 2.2, is an example of a character. Examples of characters that might be possessed by a given agent a include *dishonest* (a always tell the opposite of what a believes), *conscious* (a remembers each interaction), *forgetful* (a forgets each interaction), *verbose* (a tells everybody everything that a believes), *silent* (a is not sending any message) [9].

The personality of the agents determines which actions they prefer to do and which actions they can do. For convenience we assume that every agent is conscious and that attributes not used in a rule are omitted. The decision to send messages is a conscious act of mind, messages are added to the out-buffer of an agent, meaning that it is in a state of being ready for transmission. The rule SEND is given as follows:

$$\begin{aligned}
&\prec a \mid \mathsf{bel}, \mathsf{pers}, \mathsf{in}, \mathsf{out} \cup \{\mathsf{Transmit}(a, b, F)\} \succ \\
&\longmapsto \\
&\prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, b, F)\}, \mathsf{pers}, \mathsf{out} \succ \ \| \ \mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b \\
&\qquad\qquad \text{if } \mathsf{Honest}\ (a) \in \mathsf{pers} \text{ and } F \in \mathsf{bel}
\end{aligned}$$

In the rule for receiving messages, the agent b believes that he has received the message and b also memorizes that a is an agent, as given by the REC_1 rule:

$$\begin{aligned}
&\prec b \mid \mathsf{bel}, \mathsf{pers}, \mathsf{in}, \mathsf{out} \succ \ \| \ \mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b \\
&\longmapsto \\
&\prec b \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, b, F)\}, \mathsf{pers}, \mathsf{in} \cup \{\mathsf{Transmit}(a, b, F)\}, \mathsf{out} \succ
\end{aligned}$$

The in-buffer in represents messages ready for protocol execution. Similar rules can be constructed for the other personalities, described above.

4 Making agents perform the deduction themselves

The agents can be strengthened to be limited theorem provers, denoted *local reasoners.* They can perform deductions during each transition step, and extract as much relevant information out of the belief set as possible. But how is this possible? Rewriting logic assumes that the specifications written are both terminating and confluent. It is not surprising that the main problem about local reasoners is to assure that they stop proving during each transition step. The only way to manage this is requiring that the agents are *logically focused*, that is, they minimize computationally bad strategies for deduction and maximize good strategies for obtaining information. The deduction engine for beliefs are based on the following axioms in epistemic logic:

$$\begin{array}{ll}
\mathsf{Bel}_a(\phi) \wedge \mathsf{Bel}_a(\phi) \to \mathsf{Bel}_a(\phi) & \text{contraction} \\
\mathsf{Bel}_a(\phi \wedge \psi) \to \mathsf{Bel}_a(\phi) \wedge \mathsf{Bel}_a(\psi) & \wedge\text{-distribution} \\
\mathsf{Bel}_a(\phi) \wedge \mathsf{Bel}_a(\phi \to \psi) \to \mathsf{Bel}_a(\psi) & \text{modus ponens } (K_{\text{Bel}}) \\
\mathsf{Bel}_a(\mathsf{Bel}_a(\phi)) \to \mathsf{Bel}_a(\phi) & \text{reflective awareness } (4^C_{\text{Bel}})
\end{array}$$

The first axiom is a tautology in propositional logic $\wedge$-contraction: in the context of epistemic logic it express that beliefs might be contracted. The second axiom states that beliefs distributes over conjunction, and is provable in any normal epistemic logic. The third axiom is the well-known K-axiom for epistemic logics, which expresses that beliefs are closed under modus ponens. The fourth axiom, 4^C_{Bel}, says that if an agent has reflective awareness of its beliefs, the agent also posses these beliefs. Reflective awareness is rarely if ever mentioned in contexts of epistemic logics. Local theorem proving by agents that are *eager to know*, gives a practical example of how useful this axiom can be. The deductions that involves trust relations to infer new beliefs, have the following pattern:

$$\dfrac{\overset{[\text{ Reflective awareness }]}{\mathsf{Bel}_a(\mathsf{Bel}_a(\varphi)) \to \mathsf{Bel}_a(\varphi)} \qquad \dfrac{\overset{[\text{ Fresh message }]}{\mathsf{Bel}_a(\mathsf{Transmit}(b,a,\varphi))} \qquad \dfrac{\overset{[\text{ Trust Axiom: Trust}(a,b)\text{]}}{\mathsf{Bel}_a(\forall X\,(\mathsf{Transmit}(b,a,X) \to \mathsf{Bel}_a(X)))}}{\mathsf{Bel}_a(\mathsf{Transmit}(b,a,\varphi) \to \mathsf{Bel}_a(\varphi))}\ \text{IA}}{\mathsf{Bel}_a(\mathsf{Bel}_a(\varphi))}}{\mathsf{Bel}_a(\varphi)}$$

Note that the previous deduction, formulated as natural deduction alike inferences, hides an important distinction between the external view, represented by each of the outermost belief operators Bel, and the internal deduction activity inside agents, represented by the arguments of the outermost operators. Hence as considered from inside, the agents might observe:

$$\dfrac{\overset{(3)}{\mathsf{Bel}_a(\varphi) \to \varphi} \qquad \dfrac{\overset{(1)}{\mathsf{Transmit}(b,a,\varphi)} \qquad \dfrac{\overset{(2)}{\forall X\,(\mathsf{Transmit}(b,a,X) \to \mathsf{Bel}_a(X))}}{\mathsf{Transmit}(b,a,\varphi) \to \mathsf{Bel}_a(\varphi)}\ \text{IA}}{\mathsf{Bel}_a(\varphi)}}{\varphi}$$

The agent a receives a message (1), checks whether it matches a trust relation (2), and if it does, performs modus ponens on the instantiated conditional, and removes the outermost belief (3). The label IA denotes the instantiation algorithm. The previous inferences does not capture local deductions faithfully. The deduction is performed top down. The instantiation of the universal quantifier is not performed blindly, an instantiation is triggered by the incoming message $\mathsf{Transmit}(b,a,\varphi)$: Each quantified sentence is matched with the received message in order to search for a potential instantiation. Reflection is part of the propositional fragment of the deduction engine, and is executed whenever possible.

4.1 Operational interpretation of epistemic deduction

The agents minimize their reasoning by trying to abandon superfluous information and avoid recursive traps for deductions: Idempotence of bel entails that multiple copies of a single sentence are avoided. Conjuncts are broken down when the outermost connective is a belief operator, since beliefs distributes over conjunction ($\wedge$). Note that the framework for local deduction breaks a holy principle in logic that was stressed by Frege: that an implication is not a causal nor temporal relation [6]. In contrast, a local deduction in the PROSA framework is triggered by a communication event and takes one time unit to execute:

$$\begin{array}{ll} \mathsf{Bel}_a(\varphi) \wedge \mathsf{Bel}_a(\varphi) \rightarrow \bigcirc(\mathsf{Bel}_a(\varphi)) & (\mathrm{contr}^{\mathrm{TIME}}) \\ \mathsf{Bel}_a(\varphi \wedge \psi) \rightarrow \bigcirc(\mathsf{Bel}_a(\varphi) \wedge \mathsf{Bel}_a(\psi)) & (\wedge\mathrm{distr}^{\mathrm{TIME}}) \end{array}$$

In the semantics idempotence of beliefs follows from general idempotence of every set. The distribution of Bel over conjunctions is interpreted as distribution on the highest level of the belief set of the agent:

$$\begin{array}{ll} \prec a \mid \mathsf{bel} \cup \{\varphi, \varphi\} \succ \;\longmapsto\; \prec a \mid \mathsf{bel} \cup \{\varphi\} \succ & (\mathrm{C}_\wedge) \\ \prec a \mid \mathsf{bel} \cup \{\varphi \wedge \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi\} \cup \{\psi\} \succ & (\mathrm{D}_\wedge) \end{array}$$

Removing superfluous information does not make the agent smarter. However, new information can be obtained by using modus ponens on implications closed under the belief operator in addition to new messages received by the agent. Formalized modus ponens $\mathrm{K}_{\mathrm{Bel}}$ has been presented as one of the guiding axioms. By modifying the axiom slightly using propositional logic and introduce the next operator, it gets the shape of the sentence ($\mathrm{K}_{\mathrm{Bel}}^{\mathrm{TIME}}$):

$$\mathsf{Bel}_a(\varphi) \wedge \mathsf{Bel}_a(\varphi \rightarrow \psi) \rightarrow \bigcirc(\mathsf{Bel}_a(\varphi) \wedge \mathsf{Bel}_a(\psi)) \qquad (\mathrm{K}_{\mathrm{Bel}}^{\mathrm{TIME}})$$

Now we have stated explicitly what the target beliefs of the agent should be. But this is still not enough. If the agent a trusts an agent b with respect to the sentence φ, this would read: $\mathsf{Bel}_a(\mathsf{Transmit}(b, a, \varphi) \rightarrow \mathsf{Bel}_a(\varphi))$. But axiom ($\mathrm{K}_{\mathrm{Bel}}^{\mathrm{TIME}}$) only gives $\mathsf{Bel}_a(\mathsf{Transmit}(b, a, \varphi)) \overset{\text{next}}{\Longrightarrow} \mathsf{Bel}_a(\ \mathsf{Bel}_a(\varphi))$, which is not strictly informative. It is more appropriate to claim that a trusts b should entail: $\mathsf{Bel}_a(\mathsf{Transmit}(b, a, \varphi)) \rightarrow \mathsf{Bel}_a(\varphi)$. Hence the timed version of reflective awareness of beliefs, given by

$$\mathsf{Bel}_a(\mathsf{Bel}_a(\varphi)) \rightarrow \bigcirc\ \mathsf{Bel}_a\ (\varphi) \qquad (4_{\mathrm{bel}}^{C\ \mathrm{TIME}}),$$

would be sufficient to solve the problem. In the mapping of $\mathrm{K}_{\mathrm{Bel}}^{\mathrm{TIME}}$ to its corresponding rule, we remove the implication $\varphi \rightarrow \psi$ to avoid obvious non-terminating execution. Thus *economical modus ponens* (emp) and *reflective awareness* (refa) are characterized by the two rules:

$$\begin{array}{ll} \prec a \mid \mathsf{bel} \cup \{\varphi, \varphi \rightarrow \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi, \psi\} \succ & (\mathrm{emp}) \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Bel}_a(\phi)\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\phi\} \succ & (\mathrm{refa}) \end{array}$$

Let T_1 denote the previous theory $T_1 = \{\mathrm{SEND}, \mathrm{REC}_1, \mathrm{C}_\wedge, \mathrm{D}_\wedge, \mathrm{emp}, \mathrm{refa}\}$. Then

Theorem 1. *Any specification over $\mathscr{L}_S$ interpreted in T_1 is terminating.*

Proof. Suppose that S is an arbitrary specification, containing n agents. We prove the theorem by induction on the number of messages in the output buffer. Without loss of generality we might suppose that S does not contain messages in the network, since if S' contained m messages we assume that they are received by the agents in question by asserting that $S' \longmapsto^* S$. Messages where the receiver is not contained in S, do not play any role for the argument, since these messages are not processed. Let $\mathscr{A}$ denote the set of agents in S, and let $\mathrm{Out}(S)$ denote the function that measures the total number of messages in the out-buffers of the agents in S

$$\mathrm{Out}(S) = \sum_{\prec a \,|\, \mathsf{bel},\mathsf{out} \succ \in \mathscr{A}} |\,\mathsf{out}\,|.$$

Induction basis: Then there are no messages in any out-buffer, hence $\mathrm{Out}(S) = 0$. Every agent a in S has initially a finite set of sentences: $\mathsf{bel} = \{\varphi \mid \mathsf{Bel}_a(\varphi)\}$, $|\mathsf{bel}| = n$. A measure for termination (weight function) can be defined by summation of the syntactic complexity of the belief set, as given by the function:

$$\mathrm{sdeg}(\mathsf{bel}) = \sum_{\varphi \in \mathsf{bel}} \deg(\varphi) \times 2^{\deg(\varphi)}.$$

Obviously $\mathrm{sdeg}(\mathsf{bel}) \geqslant 0$. Since $\mathrm{Out}(S) = 0$, there are no messages to be processed, thus the belief set is never extended with new beliefs about transmissions. Each of the epistemic deduction rules R in T_1 reduces the weighting of bel:

$$\prec a \,|\, \mathsf{bel}, \mathsf{out} \succ \;\|\, \mathscr{C} \longmapsto^R \prec a \,|\, \mathsf{bel}', \mathsf{out} \succ \;\|\, \mathscr{C} \implies \mathrm{sdeg}(\mathsf{bel}) > \mathrm{sdeg}(\mathsf{bel}').$$

Consider (emp): $\prec a \mid \mathsf{bel} \cup \{\varphi, \varphi \to \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi, \psi\} \succ$. We must show that $\mathrm{sdeg}(\mathsf{bel} \cup \{\varphi, \varphi \to \psi\}) > \mathrm{sdeg}(\mathsf{bel} \cup \{\varphi, \psi\})$, which is established by showing that $\mathrm{sdeg}(\{\varphi, \varphi \to \psi\}) > \mathrm{sdeg}(\{\varphi, \psi\})$. Then we have that $\mathrm{sdeg}(\{\varphi, \varphi \to \psi\})$ $= \deg(\varphi) \times 2^{\deg(\varphi)} + (\max(\deg(\varphi), \deg(\psi)) + 1) \times 2^{\max(\deg(\varphi),\deg(\psi))+1}$, and also that $\mathrm{sdeg}(\{\varphi, \psi\}) = \deg(\varphi) \times 2^{\deg(\varphi)} + \deg(\psi) \times 2^{\deg(\psi)}$. The largest value of $\mathrm{sdeg}(\{\varphi, \psi\})$ occurs when $\deg(\varphi) = \deg(\psi)$, suppose therefore that the latter is the case, and $\deg(\varphi) = m$. Then $\mathrm{sdeg}(\{\varphi, \varphi \to \psi\}) = m \times 2^m + (m + 1) \times 2^{m+1} = m \times 2^m + 2^{m+1} + m \times 2^{m+1}$ and $\mathrm{sdeg}(\{\varphi, \psi\}) = m \times 2^m + m \times 2^m = m \times 2^{m+1}$, which proves the result. The verifications for the other rules are similar and are therefore omitted.

Consider the induction step: Suppose that the specification S that contains n output messages, is extended into a specification $S^\star$ such that one agent a has one extra output message to process, in other words $\mathrm{Out}(S^\star) = n + 1$. Suppose that this message is $m = \mathsf{Transmit}(a, b, F)$. By induction hypothesis S is terminating. Suppose without loss of generality that we emulate the terminating rewrites of S inside $S^\star$, such that $S^\star \longmapsto^* S''$ and the only possible rewrite in S'' is the transmission of m. If there exists an agent b that can receive m, and a can send m, then:

$$S^\star \longmapsto^* S'' \longmapsto^{\mathrm{SEND}} S_1 \longmapsto^{\mathrm{REC}} S_2,$$

otherwise S'' itself is a final state. In S_2, the agent b has received the message. By induction hypothesis the only possible deductions may be performed using message m. Suppose further that b trusts a, otherwise S_2 is already a final state. Then by at most $2^{\deg(F)} - 1$ applications of the epistemic deduction rules, a final state $S_2 \longmapsto^* S_3$ is reached. □

4.2 Discussion of the standard epistemic axioms

The local deduction engine balances on a thin line of termination. Consider the standard axioms of epistemic logic: Consider the standard axioms of epistemic logic are the *epistemic modus ponens* (K_{bel}), *consistency* (D_{bel}), *positive introspection* (4_{bel}) and *negative introspection* (5_{bel}):

$$\begin{array}{lr}
\mathsf{Bel}_a(\varphi) \wedge \mathsf{Bel}_a(\varphi \to \psi) \to \mathsf{Bel}_a(\psi) & (K_{\text{bel}}) \\
\mathsf{Bel}_a(\phi) \to \neg \mathsf{Bel}_a(\neg\phi) & (D_{\text{bel}}) \\
\mathsf{Bel}_a(\varphi) \to \mathsf{Bel}_a(\mathsf{Bel}_a(\varphi)) & (4_{\text{bel}}) \\
\neg \mathsf{Bel}_a(\phi) \to \mathsf{Bel}_a(\neg \mathsf{Bel}_a(\phi)) & (5_{\text{bel}})
\end{array}$$

The axiom D_{bel} expresses that the agent a's belief set is consistent, in other words $D_{\text{bel}} \to \neg(\mathsf{Bel}_a(\phi) \wedge \mathsf{Bel}_a(\neg\phi))$. The axiom of consistency is not included as a guiding axioms, since agents can have contradictory beliefs. A typical example is when an agent a trusts two agents b and c, but receives contradictory assertions from b and c. The negative introspection axiom (5_{bel}) is not coherent with the requirement that agents act only based on explicit beliefs. Axiom (5_{bel}), and any operation interpretation is abandoned since an agent should certainly *not* have explicit beliefs about everything it does not believe.

Observation 1 *The distribution axiom for implication and the axiom of positive introspection both cause non-terminating specifications.*

Proof. The operational interpretation thus gives $\mathsf{Bel}_a(\varphi) \to \bigcirc \mathsf{Bel}_a(\mathsf{Bel}_a(\varphi))$ and $\mathsf{Bel}_a(\varphi) \wedge \mathsf{Bel}_a(\varphi \to \psi) \to \bigcirc \mathsf{Bel}_a(\psi)$. The reason why they both are non-terminating, follows directly by considering the rules interpreting the axioms:

$$\begin{array}{lr}
\prec a \mid \mathsf{bel} \cup \{\varphi, \varphi \to \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi, \varphi \to \psi, \psi\} \succ & \text{(smp)} \\
\prec a \mid \mathsf{bel} \cup \{\phi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\mathsf{Bel}_a(\phi)\} \succ & \text{(pos)}
\end{array}$$

In case of (smp), the persistence of the implication $\varphi \to \psi$, result in an infinite sequence of applications of (smp), if φ is possessed by the agent. Non-termination arises from (4_{bel}) if the agent possesses one single belief φ, since sentences $\mathsf{Bel}_a(\ldots\mathsf{Bel}_a(\varphi))$ of arbitrary nesting depth will be deduced. □

Reflective awareness works opposite of the classical axioms of epistemic logic ([10],[5]), the axioms of solipsisms, positive and negative introspection. Together with economical modus ponens, reflective awareness *focus* the agent's beliefs.

4.3 Weak second order instantiation

The single agent properties and the trust relation were formulated as second order sentences. Thus to be able to deduce sentences from the properties, those sentences must be instantiated. Each time a new sentence enters a's beliefs, the sentences that contains second order quantifiers are instantiated and the potential antecedents are matched towards the recently added belief. Several of the security properties described in this paper are sentences on the normal form $\mathbf{Q}(F \to G)$, where $\mathbf{Q}$ is a quantifier prefix consisting of first or second order universal quantifiers. For convenience we say that the quantifier prefix might take only three forms, either $\mathbf{Q} = (\forall X)(\forall y_1)\ldots(\forall y_n)$,

$\mathbf{Q} = (\forall X)$, or $\mathbf{Q} = (\forall y_1)\ldots(\forall y_n)$. We write $\mathbf{Q}^1$ if there are only first order or no quantification in the prefix. Let $\boldsymbol{y}$ denote a sequence (possibly empty) of first order variables, $\boldsymbol{y} = y_1, \ldots, y_n$. Then we can characterize particularly simple classes of sentences, the *nice second order*, the *second order simple*, and the *simple quantified* sentences:

Definition 3. *The* nice second order *sentences $\mathscr{L}_{2O}$, is the least set such that:*

$$\begin{array}{rl} (i) & \mathsf{Agent}(a) \in \mathscr{L}_{2O} \\ (ii) & \textit{If } \psi \in \mathscr{L}_S \textit{ then } \mathsf{Bel}_a(\psi) \in \mathscr{L}_{2O} \textit{ and } \mathsf{Transmit}(a, b, \psi) \in \mathscr{L}_{2O} \\ (iii) & \textit{If } F(X, \boldsymbol{y}),\ G(X, \boldsymbol{y}) \in \mathscr{L}_{2O}, \textit{ then } \forall X \forall \boldsymbol{y}(F(X, \boldsymbol{y}) \to G(X, \boldsymbol{y})) \in \mathscr{L}_{2O} \\ (iv) & \textit{If } F(x, \boldsymbol{y}),\ G(x, \boldsymbol{y}) \in \mathscr{L}_{2O}, \textit{ then } \forall x \forall \boldsymbol{y}(F(x, \boldsymbol{y}) \to G(x, \boldsymbol{y})) \in \mathscr{L}_{2O} \end{array}$$

A sentence φ is *second order simple*, denoted $\varphi \in \mathscr{L}_\forall$, if φ is a closed sentence and $\varphi \in \mathscr{L}_{2O}$. If $\varphi \in \mathscr{L}_\forall$ and $\varphi = \mathbf{Q}(F \to G)$, where $\mathbf{Q}$ is a quantifier prefix, then φ is a *simple quantified* sentence. Obviously $\mathscr{L}_\forall \subseteq \mathscr{L}_{2O} \subseteq \mathscr{L}_S$. The rule for receiving messages is then changed, such that the agent extracts as much logical information from the message and the belief-set as possible, as in the rule REC$_2$:

$$\begin{array}{l} \prec b \mid \mathsf{bel}, \mathsf{pers}, \mathsf{in}, \mathsf{out} \succ \ \| \ \mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b \\ \longmapsto \\ \prec b \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, b, F)\} \cup \{\mathbb{e}(F, \mathsf{bel})\}, \mathsf{pers}, \mathsf{in} \cup \{\mathsf{Transmit}(a, b, F)\}, \mathsf{out} \succ \end{array}$$

Thus define $T_2 = (T_1 - \{\mathrm{REC}_1\}) \cup \{\mathrm{REC}_2\}$. The rewriting theory T_2 is designed in order to be able to perform correct deductions of trust according to the inference schemes introduced in the beginning of this section. If F is a sentence and bel is a set of beliefs, then the function $\mathbb{e}(F, \mathsf{bel})$ returns the instantiations of universally quantified sentences in bel, such that F matches the antecedents in bel. The boolean function $\mathbb{M}(F', F)$ decides if F' may match F: that is, F' match F iff they are equal except that some of the terms or some of the sentences in F' correspond to agent variables and sentence variables in F. The function $\mathbb{S}(F', F)$ performs the matching of the terms in F' with variables in F, resulting in a set of matching pairs. The next example illustrates a successful matching:

$$\begin{array}{l} \mathbb{S}(\mathsf{Transmit}(\mathrm{Alice}, \mathrm{Bob}, \mathsf{Bel}_{\mathrm{Carol}}(\mathsf{Agent}(\mathrm{Bob}))), \mathsf{Transmit}(x, y, \mathsf{Bel}_{\mathrm{Carol}}(X))) \\ = \{\langle \mathrm{Alice}, x\rangle, \langle \mathrm{Bob}, y\rangle, \langle \mathsf{Agent}(\mathrm{Bob}), X\rangle\} \end{array}$$

Definition 4. *The* logical expansion *of a set of sentences S with respect to a given sentence F, denoted $\mathbb{e}(F, S)$, is defined by:*

$$\begin{array}{rl} (i) & \mathbb{e}(F, \varnothing) = \{F\} \\ (ii) & \mathbb{e}(F, \{\mathbf{Q}(\varphi \to \psi)\} \cup S) = \{\mathrm{SUB}(\mathbb{S}(F, \varphi), \varphi \to \psi)\} \cup \mathbb{e}(F, S) \\ & \textit{if } \mathbf{Q}(\varphi \to \psi) \textit{ is a simple quantified sentence, and } \mathbb{M}(F, \varphi), \\ & \textit{else } \mathbb{e}(F, \{\varphi\} \cup S) = \mathbb{e}(F, S). \end{array}$$

The REC$_2$: rule contains a potential recursion trap $\{\mathbb{e}(F, \mathsf{bel})\}$, hence it is not obvious that the deduction terminates. Fortunately we can prove that

Theorem 2. *Second order universal instantiation is terminating for specifications interpreted in T_2.*

Proof. Suppose that a is an agent in a configuration $\mathscr{C}$, that just received a message $\mathsf{Transmit}(b, a, \varphi)$. We must prove that the logical expansion of this new fact leads to a finite local derivation in the rules T_2 by the agent a. Let $\{\mathsf{Bel}_a(\varphi) \mid \varphi \in \mathsf{bel}\} = \Gamma \cup \Delta$, where $\Gamma \in \mathscr{L}_S - \mathscr{L}_\forall$ and $\Delta \in \mathscr{L}_\forall$. Thus we have $\Delta = \{\mathbf{Q}_i\,(\varphi_i \to \psi_i) \mid 0 \leqslant i \leqslant n\}$. Then the new event can only match a subset $\Delta' \subseteq \Delta$, such that each element in Δ' is of the form

$$\mathbf{Q}\;(\mathsf{Transmit}(t_1, t_2, \varphi') \to \psi),$$

such that $\mathbb{M}(\mathsf{Transmit}(b, a, \varphi), \mathsf{Transmit}(t_1, t_2, \varphi'))$. Suppose for convenience that $|\Delta'| = k \geqslant 1$. Logical expansion $\oplus(\mathsf{Transmit}(b, a, \varphi), \{\mathsf{Bel}_a(\varphi) \mid \varphi \in \mathsf{bel}\})$ gives that additional k implications which are augmented to a's beliefs. Hence by m applications of (emp), the consequences $\psi_1^C, \ldots, \psi_k^C$ are augmented to a's beliefs. This gives a situation where only the epistemic deduction rules from T_2 may be applied, hence by theorem 1, the deduction terminates. □

Confidentiality is *not* a simple quantified sentence. This is not a big problem since most security policy interactions can be performed by wrapping such sentences inside beliefs or transmissions. Consider a scenario where Alice trusts Bob, and Bob wants to enforce Alice to believe that this trust relation is a secret: $\mathsf{Bel}_{\mathrm{Alice}}(\mathrm{Trust}(\mathrm{Alice}, \mathrm{Bob}))$ (A_1), and

$$\mathsf{Transmit}(\mathrm{Bob}, \mathrm{Alice}, \mathrm{CSecret}(\{\mathrm{Alice}, \mathrm{Bob}\}, \mathrm{Trust}(\mathrm{Alice}, \mathrm{Bob}))). \qquad (A_2)$$

Both sentences (A_1) and (A_2) are simply quantified sentences. When Alice receives (A_2), hence $\mathsf{Bel}_{\mathrm{Alice}}(A_2)$, she can perform the required second order instantiation according to Definition 4, hence we can conclude that
$\mathsf{Bel}_{\mathrm{Alice}}(\mathrm{CSecret}(\{\mathrm{Alice}, \mathrm{Bob}\}, \mathrm{Trust}(\mathrm{Alice}, \mathrm{Bob})))$.

4.4 Recursive second order instantiation

When the agent a performs an instantiation, new consequences might occur in a's mind. Let us consider the naive agent John. John makes himself a disciple of every agent y that claims to John that y is honest, that is, John establishes a trust relation towards every such agent y (clause B_1). John also thinks that every agent that he trust, trusts him as well (clause B_2).

$$\mathsf{Bel}_{\mathrm{John}}(\forall y(\mathsf{Transmit}(y, \mathrm{John}, \mathrm{Honest}(y)) \to \mathsf{Bel}_{\mathrm{John}}(\mathrm{Trust}(\mathrm{John}, y)))) \qquad (B_1)$$
$$\mathsf{Bel}_{\mathrm{John}}(\forall y(\mathsf{Bel}_{\mathrm{John}}(\mathrm{Trust}(\mathrm{John}, y)) \to \mathsf{Bel}_{\mathrm{John}}(\mathrm{Trust}(y, \mathrm{John})))) \qquad (B_2)$$

The sentences B_1 and B_2 indicate two kinds of weaknesses of the rules presented so-far, due to *eager reflection* and *recursive expansion.* Consider first eager reflection: The antecedent in B_2, $\mathsf{Bel}_{\mathrm{John}}(\mathrm{Trust}(\mathrm{John}, y))$ might not match any sentence in John's beliefs, because the required match $\mathsf{Bel}_{\mathrm{John}}(\mathsf{Bel}_{\mathrm{John}}(\mathrm{Trust}(\mathrm{John}, y)))$ might have been reduced by reflection. We therefore introduce an axiom removing the outermost belief, denoted *synchronous reflection* (SR):

$$\mathsf{Bel}_a(\mathsf{Bel}_a(\varphi) \to \psi) \to \mathsf{Bel}_a(\varphi \to \psi) \qquad (\mathrm{SR})$$

The axiom is given an operational interpretation by

$$\mathsf{Bel}_a(\mathsf{Bel}_a(\varphi) \to \psi) \to \bigcirc\; \mathsf{Bel}_a\;(\varphi \to \psi) \qquad (\mathrm{SR}^{\mathrm{TIME}})$$

The corresponding rule to be added to the rewrite theory, is denoted (ar):

$$\prec a \mid \mathsf{bel} \cup \{\mathsf{Bel}_a(\varphi) \to \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi \to \psi\} \succ \qquad \text{(ar)}$$

While (SR) and (SR^{TIME}) could be a bit difficult to justify, the rewrite (ar) has a natural interpretation: The antecedent belief in the term to be rewritten is just unnecessary. Does $T_1 \cup \{\text{ar}\}$ and $T_2 \cup \{\text{ar}\}$ terminate? Fortunately the answer is yes:

Theorem 3. *Both $T_1 \cup \{\text{ar}\}$ and $T_2 \cup \{\text{ar}\}$ give terminating specifications.*

Proof. The principal job is to establish termination of $T_1 \cup \{\text{ar}\}$, since termination of $T_2 \cup \{\text{ar}\}$ follows from the former, by a proof similar to theorem 2. The definition of the ordering is straightforward: Define first the function $\#\text{Refl}(\Phi)$, that counts the number of occurrences of reflection schemes $\mathsf{Bel}_a(\varphi) \to \varphi$ inside a formula Φ, by obvious recursion on $\deg(\Phi)$. Then we define

$$\text{rdeg}(\mathsf{bel}) = \sum_{\varphi \in \mathsf{bel}} \#\text{Refl}(\varphi) \quad \text{and} \quad \text{SR}(\mathsf{bel}) = \text{sdeg}(\mathsf{bel}) + \text{rdeg}(\mathsf{bel}).$$

The proof then follows the same line of argument as theorem 1, an induction on the size of the output buffers in a specification S. Consider the (ar) rule. There are two cases: either $\deg(\mathsf{Bel}_a(\varphi)) \leqslant \deg(\psi)$ or $\deg(\mathsf{Bel}_a(\varphi)) > \deg(\psi)$. In the former case we observe $\text{sdeg}(\{\mathsf{Bel}_a(\varphi) \to \psi\}) = \text{sdeg}(\{\varphi \to \psi\})$ but also $\text{rdeg}(\{\varphi \to \psi\}) = \text{rdeg}(\{\mathsf{Bel}_a(\varphi) \to \psi\}) - 1$. In the latter case, we have that
$\text{sdeg}(\{\varphi \to \psi\}) = \text{sdeg}(\mathsf{Bel}_a(\varphi)) - 1$ and $\text{rdeg}(\{\varphi \to \psi\}) = \text{rdeg}(\{\mathsf{Bel}_a(\varphi) \to \psi\}) - 1$, hence $\text{SR}(\mathsf{bel} \cup \{\mathsf{Bel}_a(\varphi) \to \psi\}) > \text{SR}(\mathsf{bel} \cup \{\varphi \to \psi\})$. The verification that the remaining rules reduce $\text{SR}(\mathsf{bel})$, is straightforward: In each case we have $\text{rdeg}(\mathsf{bel}) = \text{rdeg}(\mathsf{bel}')$ but $\text{sdeg}(\mathsf{bel}) > \text{sdeg}(\mathsf{bel}')$ for any old rule R of the form $\prec a \mid \mathsf{bel} \succ \longmapsto^{R} \prec a \mid \mathsf{bel}' \succ$. □

Yet the problem with the naive disciple is not solved by the epistemic inference in propositional logic, we must get behind the quantifier barrier:

$$\mathsf{Bel}_a(\mathbf{Q}\,(\mathsf{Bel}_a(\varphi) \to \psi)) \to \mathsf{Bel}_a(\mathbf{Q}\,(\varphi \to \psi)) \qquad (\text{SR}_\forall)$$

Hence the corresponding rewrite rule is given by the following:

$$\prec a \mid \mathsf{bel} \cup \{\mathbf{Q}\,(\mathsf{Bel}_a(\varphi) \to \psi)\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\mathbf{Q}\,(\varphi \to \psi)\} \succ \qquad (\text{ar}_\forall)$$

Corollary 1. *Both $T_1 \cup \{\text{ar}, \text{ar}_\forall\}$ and $T_2 \cup \{\text{ar}, \text{ar}_\forall\}$ are terminating.*

Consider recursive expansion: John is only capable of performing his correct deductions if the logical expansion is performed recursively over the new sentences produced. New sentences occur in the mind of the agent through communication (REC_2) or by deduction. Suppose that John receives a message from Alice of the required kind: $\mathsf{Transmit}(\text{Alice}, \text{John}, \text{Honest}(\text{Alice}))$, or written explicitly:

$$\mathsf{Transmit}(\text{Alice}, \text{John}, \forall X \forall z (\mathsf{Transmit}(\text{Alice}, z, X) \to \mathsf{Bel}_{\text{Alice}}(X))).$$

If John applies the old deduction rules presented in the beginning of section 4.1, then John is not able to deduce Trust(Alice, John). One instantiation is certainly not enough, the instantiation must be performed recursively on the deduced formulas. Hence the epistemic deduction rules in the rewrite theory T_2 are replaced by the rules

$$\begin{array}{ll}
\prec a \mid \mathsf{bel} \cup \{\varphi \wedge \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi, \psi\} \cup \mathrm{e}(\varphi, \mathsf{bel}) \cup \mathrm{e}(\psi, \mathsf{bel}) \succ & (\mathrm{D}^{\wedge}_{\mathrm{SO}}) \\
\qquad \text{if } \varphi \notin \mathsf{bel} \text{ or } \psi \notin \mathsf{bel} & \\
\prec a \mid \mathsf{bel} \cup \{\varphi, \varphi \to \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi, \psi\} \cup \mathrm{e}(\psi, \mathsf{bel}) \succ \text{ if } \psi \notin \mathsf{bel} & (\mathrm{emp}_{\mathrm{SO}}) \\
\prec a \mid \mathsf{bel} \cup \{\mathsf{Bel}_a(\varphi)\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi\} \cup \mathrm{e}(\varphi, \mathsf{bel}) \succ \text{ if } \varphi \notin \mathsf{bel} & (\mathrm{refa}_{\mathrm{SO}})
\end{array}$$

The idea behind the rules $(\mathrm{D}^{\wedge}_{\mathrm{SO}}, \mathrm{emp}_{\mathrm{SO}}, \mathrm{refa}_{\mathrm{SO}})$ is that each expand the freshly created sentences to check whether they might trigger any interesting instantiation of a second order simple sentence. It is important that the previous deductions performed by the agents are preserved within the second order extension. Hence contraction $(\mathrm{C}_{\wedge})$ is included and the remaining axioms are interpreted by

$$\begin{array}{ll}
\prec a \mid \mathsf{bel} \cup \{\varphi \wedge \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \succ \text{ if } \varphi \in \mathsf{bel} \text{ and } \psi \in \mathsf{bel} & (\mathrm{D}^{\wedge}_{\mathrm{PL}}) \\
\prec a \mid \mathsf{bel} \cup \{\varphi, \varphi \to \psi\} \succ \longmapsto \prec a \mid \mathsf{bel} \cup \{\varphi\} \succ \text{ if } \psi \in \mathsf{bel} & (\mathrm{emp}_{\mathrm{PL}}) \\
\prec a \mid \mathsf{bel} \cup \{\mathsf{Bel}_a(\varphi)\} \succ \longmapsto \prec a \mid \mathsf{bel} \succ \text{ if } \varphi \in \mathsf{bel} & (\mathrm{refa}_{\mathrm{PL}})
\end{array}$$

The rules $(\mathrm{D}^{\wedge}_{\mathrm{PL}}, \mathrm{emp}_{\mathrm{PL}}, \mathrm{refa}_{\mathrm{PL}})$ are motivated by the need to resolve delayed inferences: In the interpretation of "beliefs as bag", the latter rules condense the application of two rules in one: $\mathrm{D}^{\wedge}_{\mathrm{PL}} = (\mathrm{D}_{\wedge}, \mathrm{C}_{\wedge})$. The full second order rewrite theory including quantification T_{SO} is given by the rules

$$T_{\mathrm{SO}} = \{\mathrm{SEND}, \mathrm{REC}_2, \mathrm{C}, \mathrm{D}^{\wedge}_{\mathrm{PL}}, \mathrm{emp}_{\mathrm{PL}}, \mathrm{refa}_{\mathrm{PL}}, \mathrm{D}^{\wedge}_{\mathrm{SO}}, \mathrm{emp}_{\mathrm{SO}}, \mathrm{refa}_{\mathrm{SO}}\}.$$

The requirement that each deduced sentence is not contained in bel, as in case of $\mathrm{D}^{\wedge}_{\mathrm{SO}}$, $\mathrm{emp}_{\mathrm{SO}}$, and $\mathrm{refa}_{\mathrm{SO}}$ is necessary since without the requirements the rules would be trivially non-terminating. Unfortunately most specifications will not terminate, even for fair executions:

Observation 2 *There are specifications interpreted in T_{SO}, such that second order universal instantiation is non-terminating.*

Proof. Suppose that a, b, and c denote three agents in a configuration $\mathscr{C}$. Assume furthermore that c believes that a trusts c, that c recently sent a message to a, and finally that c believes that a and b have reflective beliefs about each other. The assumptions are specified as follows:

$$\begin{array}{l}
(1)\ \mathsf{Bel}_c(\mathrm{Trust}(a, c)) \\
(2)\ \mathsf{Bel}_c(\mathsf{Transmit}(c, a, \varphi)) \\
(3)\ \mathsf{Bel}_c(\forall X(\mathsf{Bel}_a(X) \to \mathsf{Bel}_b(\mathsf{Bel}_a(X)))) \\
(4)\ \mathsf{Bel}_c(\forall X(\mathsf{Bel}_b(X) \to \mathsf{Bel}_a(\mathsf{Bel}_b(X)))).
\end{array}$$

Then any fair execution of $\mathscr{C}$ will be non-terminating: From (1) and (2) by expansion and $\mathrm{emp}_{\mathrm{SO}}$, we have $\mathsf{Bel}_c(\mathsf{Bel}_a(\varphi))$ and expansion of $\mathsf{Bel}_a(\varphi)$ gives the instance $\mathsf{Bel}_b(\mathsf{Bel}_a(\varphi))$ from (3). Yet another expansion according to $\mathrm{emp}_{\mathrm{SO}}$ on (4) gives $\mathsf{Bel}_a(\mathsf{Bel}_b(\mathsf{Bel}_a(\varphi)))$, and so on. Since execution is required to be fair, each expansion can be performed eventually. □

The epistemic axiom 4_{bel} that might give non-terminating specification is definable by the second order formula $\forall X(\mathsf{Bel}_a(X) \to \mathsf{Bel}_a(\mathsf{Bel}_a(X)))$. If we assume that executions might be *unfair*, it could happen that 4_{bel} is always applied. Then there could be infinitely many instantiations of the $\forall X$ quantifier, resulting in an infinite deduction.

5 Conclusion

Modal logic is commonly regarded as powerful with respect to expressibility, but unfeasible from a practical point of view. Our case study about local deductions of trust supports this: standard axioms have been excluded and a couple of new reflection axioms was added to the theory. We have shown how agents can perform logical inferences and universal instantiations, based on their trust relations. Yet, in order to be able to perform inferences based on more intricate security properties, a recursive instantiation algorithm is required. Unfortunately, it is difficult to know whether a set of beliefs is closed under finite deductions or whether it is non-terminating in the general case. Although the current implementation covers several standard examples, there are several open problems: The two most important are extend the logic to include multi-agent reasoning and find the exact border of termination for local theorem provers.

References

1. Abadi, M., Burrows, M., Lampson, B., and Plotkin, G. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.
2. Bieber, P. and Cuppens, F. Expressions of Confidentiality Policies with Deontic Logic. In *Deontic Logic in Computer Science: Normative System Specification*, pages 103–123. John Wiley and Sons Ltd, 1993.
3. Burrows, B., Abadi, M., and Needham, R. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
4. M. Cohen and M. Dam. Logical omniscience in the semantics of BAN logic. In *Proc. FCS'05*, 2005.
5. Moses Y. Fagin R., Halpern J.Y. and Vardi. *Reasoning about knowledge.* The MIT Press, 1995.
6. Frege, G. *Begriffschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Halle: L. Nebert, 1879. Available in several translations.
7. Glasgow, J. and Macewen, G. A Logic for Reasoning About Security. *ACM Transactions on Computer Systems*, 10(3):226 – 264, August 1992.
8. Hagalisletto, A. M. Attacks are Protocols Too. In *Proc. of The Second Int. Conf. on Availability, Reliability and Security (IEEE ARES 2007)*, pages 1197 – 1206.
9. Hagalisletto, A. M. and Haugsand, J.. A Formal Language for Specifying Security Properties. In *Proc. for the Workshop on Specification and Automated Processing of Security Requirements - SAPS'04*. Austrian Computer Society, 2004.
10. Hintikka, J. *Knowledge and Belief.* Cornell University Press, Ithaca, 1962.
11. Lowe, G. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proc. of the Second Int. Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 147–166. Springer-Verlag, 1996.

12. Paulson, L. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.
13. Fagin R. and Halpern J.Y. Belief, Awareness and Limited Reasoning. *Artificial Intelligence*, 34:39–76, 1988.
14. Shapiro, S. *Foundations without Foundationalism: A Case Study for Second Order Logic.* Oxford Science Publication, 1991.