Università degli Studi di Bologna
DEIS

On the automatic verification of interaction protocols using $g-\mathcal{S}$CIFF

Marco Alberti Federico Chesani
Marco Gavanelli Evelina Lamma
Paola Mello Paolo Torroni

November 11, 2004

DEIS Technical Report no. DEIS-LIA-004-04 *LIA Series no. 72*

On the automatic verification of interaction protocols using $g-\mathcal{S}$CIFF

Marco Alberti [1] Federico Chesani [2] Marco Gavanelli [1]
Evelina Lamma [2] Paola Mello [1] Paolo Torroni [1]

[1] *Dipartimento di Ingegneria, Università di Ferrara*
Via Saragat, 1
44100 Ferrara, Italy
{malberti, mgavanelli, elamma}@ing.unife.it
[2] *DEIS, Università di Bologna*
V.le Risorgimento 2
40136 Bologna, Italy
{fchesani, pmello, ptorroni}@deis.unibo.it

November 11, 2004

Abstract. Interaction protocol verification has been in recent years intensively investigated within and outside multi-agent research. Many techniques and models have been proposed based on a number of assumptions and choices, among which are the kind of knowledge available and the kind of properties that are subject of verification.

We focus on interaction protocol verification in open multi-agent systems, where the internal architecture of the individual agents is not known a-priori. In such a setting, while it is not possible to predict the behaviour of individual agents, it is still possible to approach the interaction verification problem either by assuming that agents will indeed comply to the protocols, thus focussing on the study of properties of the interaction protocols, or by assuming that the agent behaviour, if not predictable, is at least observable, and then focussing on the run-time verification of compliant behaviour of interacting agents by reasoning on the messages they exchange with each other.

In this article, we build on previous work on logic-based specification and run-time verification of protocol compliance, and we extend the SOCS social framework towards verification of protocol properties. In doing so, we propose a unified formal and computational framework based on abductive logic programming which can be used to (*i*) design and specify interaction protocols that provably exhibit desirable properties and (*ii*) verify, at run-time, that agent interaction actually complies to the specified interaction protocols.

Keywords: *computational logic, logic programming, multi-agent communication, agent interaction, properties verification, SOCS, $\mathcal{S}CIFF$, g−$\mathcal{S}CIFF$*

1 Introduction

Verification is a critical step of software development. In its broader understanding, software verification is a process aimed at proving that, given some specifications, some properties hold. Such properties are generally divided into *safety* properties and *liveness* properties [Lamport, 1983], the former indicating that some undesired condition will not occur during the lifetime of a system, the latter that some desired condition will instead hold at some point. It is easy to find many examples of such properties, spanning from termination properties (like deadlock or loop safety) to security properties, involving disclosure of sensible data, to fairness properties like absence of starvation.

In recent years the attention of many researchers has focussed on multi-agent systems as a powerful and intuitive metaphor to model many aspects of distributed intelligent computation. Alongside the evolution and growth of multi-agent technology and its application to ever more complex scenarios, the problem of verification of multi-agent systems is faced with new challenges, involving new aspects whose correctness has to be verified.

Agent interaction verification is one such aspect which has been intensively studied under many perspectives. Among them we cite the application of temporal logic of knowledge to the formal verification of security protocols [Dixon *et al.*, 2004a; Dixon *et al.*, 2004b], work done on verification of epistemic properties via bounded and unbounded model checking [Kacprzak *et al.*, 2004; Raimondi and Lomuscio, 2004], verification of properties that hold if protocols are followed correctly or not [Lomuscio and Sergot, 2002], and translation of protocol insecurity problems into logic programming [Armando *et al.*, 2004].

In our work, we approach agent interaction verification from the perspective of an open environment, in which it is not possible to make any assumption on the nature and architecture of the autonomous agents interacting together. We assume instead that agents interact with each other by exchanging messages. In such an open system, a possible way for agents to engage in a conversation could be to choose some pre-defined patterns of interaction, i.e., protocols. If we assume that such protocols are formally defined in some way outside of the agents, there are two kinds of verification that we can consider. In the first place, we could start from the assumption that agents do behave according to such formally defined protocols. Protocols then become in a way a specification of the (externally observable) behaviour of agents. In this case, it would be important to verify if the specifications of protocols are such that some (safety or liveness) properties hold in case of com-

pliant behaviour of the interacting parties. We call this approach *static* verification of protocol properties.

We can also follow a different approach. We could drop the assumption that agents behave according to the protocol specification, but we assume that their exchanged messages are externally observable. In that case, the agent messages become events which are the object itself of verification. This can be done based on a collection of past events (exchanged messages), or at run-time, as agents interact with each other. We call this approach *dynamic* or *on-the-fly* verification of protocol compliance.

This work aims at integrating these two aspects of interaction verification into a unified framework. Ultimately, we aim to provide a formal machinery that can be used to

i (statically) design protocols that exhibit some desired properties and prevent some undesired condition to occur, under the assumption that agents do comply to the protocols, and

ii (dynamically) verify that agents are indeed complying to them.

In doing this, we build on previous work, in which we have proposed a formalism, called *Social Integrity Constraints* (ic_S) [Alberti *et al.*, 2004b], that can be used to specify agent interaction protocols. We used ic_S to model several kinds of protocols, ranging from low-level inter-process communication protocols such as the TCP-IP, to human-to-human interaction protocols based on e-mail exchange, including well known e-commerce protocols such as the NetBill and the English Auction protocols, and public key authentication protocols such as the Needham-Schroeder.

ic_S are the distinguishing feature of the SOCS social model, which together with the $\mathcal{S}$CIFF [Alberti *et al.*, 2003b] proof-procedure represents the main building blocks of the SOCS social framework. In such a framework, $\mathcal{S}$CIFF can be used to verify whether a collection of messages exchanged by agents conforms to a protocol specified through ic_S.

In this work, we extend the $\mathcal{S}$CIFF procedure in the direction of static verification of protocol properties. By doing so, we integrate the two aforementioned approaches by proposing a single formalism based on ic_S to specify agent interaction and two different procedures to perform static and dynamic verification.

The kind of properties that we consider for static verification are in general those that one can formally define by way of logical formulas about socially relevant events such as messages. The purpose of verification is to define whether, given a protocol specification and a formula, there exists a possible sequence of events which is compliant to the protocol and which verifies the formula.

This can be used in different ways. For instance, if the formula is an empty (always verified) formula, the existence of a compliant sequence of events can be

read as a proof that the protocol is "well-defined" (i.e., it is not impossible for agents to comply to it). If the formula represents instead the negation of a desired property, finding a compliant sequence of events which verifies the formula means to show that the protocol does not enforce the desired behaviour: for instance, in the context of a security protocol a compliant sequence of undesired events could be considered as a proof that the protocol is flawed.

For this purpose, in this article we enhance the $\mathcal{S}$CIFF in the direction of automatic generation of compliant sequences of events. We discuss the properties of such new proof-procedure ($g{-}\mathcal{S}$CIFF) in terms of correctness and guaranteed termination. Finally, we confront our approach with well-known problems taken from the literature, namely the NetBill protocol and the Needham-Shroeder protocol. We take these two protocols as examples of use of the SOCS social framework enhanced with $g{-}\mathcal{S}$CIFF, to show the potential of our approach.

The paper is structured as follows. In Section 2 we give the necessary background. In Section 3 we describe the $g{-}\mathcal{S}$CIFF and in Section 4 we discuss the two case studies. Related work and conclusion follow.

2 Background

Many existing techniques for verification of agent-based behaviour are based on formal logics, thus providing the clear semantic basis for the verification tasks. A well-known and widely used approach for automatic analysis of reactive systems is model checking [Clarke *et al.*, 2000; Merz, 2001]. The inputs to a model checker are usually a finite state description of the system to be analyzed, and a number of properties, often expressed as formulas of temporal logic that are expected to hold of the system. The model checker either confirms that the properties hold or reports that they are violated. In the latter case, it provides a counterexample: a run that violates the property. Such a run can provide valuable feedback and points to design errors. An alternative approach for proving properties is the proof theoretic one, which relies on the syntactic structure of properties and logic-based specifications. Usually, a proof is given by contradiction. The main drawback of this approach lies on the undecidability of first order logic, and its (exponential) complexity.

In the past years, a social framework called SOCS [SOCS-web, ; Alberti *et al.*, 2004b] has been developed aimed at specifying and verifying agent interaction, with a special focus on on-the-fly verification of protocol compliance by observation. This is the problem of deciding whether a pool of agents interacting together are behaving according to some specified protocols. The SOCS social framework consists of a social model and of a proof-procedure called $\mathcal{S}$CIFF. In the model, protocols are specified by way of "*Social Integrity Constraints*" and based on some additional information which we call "*Social Knowledge*". The idea is to consider the messages that are being exchanged by the agents and verify if they violate the constraints

defining the protocols. The $\mathcal{S}$CIFF proof-procedure is used to automatically verify the compliance of messages to protocols.

On-the-fly verification differs from what we can call "static" verification mainly because it assumes that a number of social events (messages) is given as an input to the verifier. Other techniques for static verification instead produce histories of events as an output, possibly as a counterexample.

This work is aimed at reconciling these two distinct approaches to interaction verification, proposing the Social Integrity Constraints as a formalism to be used for both static and for on-the-fly verification of agent interaction. The idea is to use the $\mathcal{S}$CIFF, suitably extended, to check whether a given property, expressed as a formula, is a logical consequence of the Social Integrity Constraints of a given society. For this purpose, we need to equip the proof procedure with the capability of generating compliant histories. In fact, to state that a property holds in a society amounts to checking whether the given property holds in *all* compliant histories.

In this way, we propose the SOCS approach as an alternative, proof-theoretic approach to interaction verification in a broader sense.

2.1 The SOCS social model

The SOCS social model [Alberti *et al.*, 2004b] was originally conceived to describe the knowledge about societies of agents in a declarative way. Such knowledge is mainly composed of two knowledge bases: a *static* one called $SOKB$, defining the society organizational and "normative" elements, and a *dynamic* one called $SEKB$, describing the "socially relevant" events, that have so far occurred. In this paper, such events will be messages, exchanged by agents. In addition to these two categories of knowledge, information about social *goals* is also maintained, while Social Integrity Constraints (ic_S) are collected into a set called $\mathcal{IC}_S$.

A society may evolve, as new events happen, giving rise to sequence of society instances, each one characterised by the static components (including $SOKB$, social goals and $\mathcal{IC}_S$) and the dynamic $SEKB$ (including *Happened events*, denoted by functor **H**, and *Expectations*, denoted by functors **E** and **EN**).

The collection of happened events is the history, **HAP**, of a society instance. Events are represented as ground atoms

$$\mathbf{H}(Event[, Time]).$$

Expectations can be

$$\mathbf{E}(Event[, Time]) \qquad \mathbf{EN}(Event[, Time])$$

for, respectively, positive and negative expectations. **E** is a positive expectation about an event (the society expects the event to happen) and **EN** is a negative

expectation, (the society expects the event not to happen). Explicit negation ($\neg$) can be applied to expectations.

The arguments of expectation atoms can be non-ground terms. Intuitively, if an $\mathbf{E}(X)$ atom is in the set **EXP** of expectations generated by the society, it indicates that an event $\mathbf{H}(Y) \in \mathbf{HAP}$ which unifies with it, X/Y, is expected. One such event will be enough to fulfill the expectation: thus, variables in an **E** atom are always existentially quantified, with scope the entire set of expectations, whereas the other variables, that occur only in **EN** atoms are universally quantified, expressing the idea that if $\mathbf{EN}(p(X))$ belongs to **EXP**, then $p(X)$ is expected not to happen for all possible values of X.

$SOKB$ is a logic program [Lloyd, 1987], i.e., a collection of clauses representing predicate definitions in which the body can contain expectations and CLP constraints [Jaffar and Maher, 1994]. An example of SOKB is shown in Fig. 6 below. The goal $\mathcal{G}$ of the society has the same syntax as the *Body* of a clause in the SOKB, and the variables are quantified accordingly.

ic_S are in the form of implications. We report here the characterizing part of their syntax:

$$
\begin{array}{rcl}
ic_S & ::= & \chi \rightarrow \phi \\
\chi & ::= & (HEvent|Expectation)\ [\wedge BodyLiteral]^\star \\
BodyLiteral & ::= & HEvent|Expectation|Literal|Constraint \\
\phi & ::= & HeadDisjunct\ [\ \vee HeadDisjunct\ ]^\star|false \\
HeadDisjunct & ::= & Expectation\ [\ \wedge (Expectation|Constraint)]^\star \\
Expectation & ::= & [\neg]\mathbf{E}(Event\ [,T]) \mid [\neg]\mathbf{EN}(Event\ [,T]) \\
HEvent & ::= & [\neg]\mathbf{H}(Event\ [,T])
\end{array}
\tag{1}
$$

Given an ic_S $\chi \rightarrow \phi$, χ is called the *body* (or the *condition*) and ϕ is called the *head* (or the *conclusion*).

Examples of ic_S can be found throughout the paper (see Fig. 7 and Fig. 5).

2.2 Declarative Semantics of the SOCS social model

The SOCS social model has been interpreted in terms of Abductive Logic Programming [Kakas *et al.*, 1998], and an abductive semantics has been proposed for it [Alberti *et al.*, 2003a].

An Abductive Logic Program (ALP, for short) [Kakas *et al.*, 1998] is a triple $\langle KB, \mathcal{A}, IC\rangle$ where KB is a logic program, (i.e., a set of clauses), $\mathcal{A}$ is a set of predicates that are not defined in KB and that are called *abducibles*, IC is a set of formulae called *Integrity Constraints*. An abductive explanation for a goal G is a set $\Delta \subseteq \mathcal{A}$ such that $KB \cup \Delta \models G$ and $KB \cup \Delta \models IC$, for some notion of entailment $\models$.

In our social model, the idea is to exploit abduction for defining expected behaviour of the agents inhabiting the society, and an abductive proof procedure such

as the $\mathcal{S}$CIFF to dynamically *generate* the expectations and perform the *compliance check*. By "compliance check" we mean the procedure of checking that the ic_S are not violated, together with the function of detecting fulfillment and violation of expectations.

We give semantics to a society instance by defining those sets of expectations which, together with the society's knowledge base and the happened events, imply an instance of the goal—if any—and *satisfy* the integrity constraints.

The declarative semantics of the SOCS social model relies on the notions of *instance* of a society, characterized by the set **HAP** of happened events. Note that we assume that such events are always ground.

Definition 1 (Society instance) *An* instance $\mathcal{S}_{\mathbf{HAP}}$ *of a society* $\mathcal{S}$ *is represented as an ALP, i.e., a triple* $\langle P, \mathcal{E}, \mathcal{IC}_S \rangle$ *where:*

- P *is the SOKB of* $\mathcal{S}$ *together with the history of happened events* **HAP***;*
- $\mathcal{E}$ *is the set of* abducible predicates*, namely* **E***,* **EN***,* $\neg$**E***,* $\neg$**EN***;*
- $\mathcal{IC}_S$ *are the social integrity constraints of* $\mathcal{S}$*.*

The declarative semantics relies on three-valued completion [Kunen, 1987], and is based on various levels of consistency.

Definition 2 ($\mathcal{IC}_S$-consistency) *Given a society instance* $\mathcal{S}_{\mathbf{HAP}}$*, an* $\mathcal{IC}_S$-consistent *set of social expectations* **EXP** *is a set of expectations such that:*

$$Comp(SOKB \cup \mathbf{EXP} \cup \mathbf{HAP}) \cup CET \models \mathcal{IC}_S \tag{2}$$

where the symbol $\models$ is understood as three-valued entailment, *Comp* stands for Completion and CET is the Clark Equality Theory.

$\mathcal{IC}_S$-consistent sets of expectations could be self-contradictory (e.g., both $\mathbf{E}(p)$ and $\neg\mathbf{E}(p)$ may belong to a $\mathcal{IC}_S$-consistent set). We will say that a set **EXP** is E-consistent if it does not contain both a positive and a negative expectation of the same event, and that it is $\neg$-consistent if it does not contain both an expectation and its explicit negation:

Definition 3 (E-consistency) *A set of social expectations* **EXP** *is* E-consistent *if and only if for each (ground) term* p*:*

$$\{\mathbf{E}(p), \mathbf{EN}(p)\} \not\subseteq \mathbf{EXP}$$

Definition 4 ($\neg$-consistency) *A set of social expectations* **EXP** *is* $\neg$-consistent *if and only if for each (ground) term* p*:*

$$\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \mathbf{EXP} \qquad \textit{and} \qquad \{\mathbf{EN}(p), \neg\mathbf{EN}(p)\} \not\subseteq \mathbf{EXP}.$$

Given a society instance, a set of expectations is called *admissible* if and only if it satisfies Definitions 2, 3 and 4, i.e., if it is $\mathcal{IC}_S$-, E- and $\neg$-consistent.

The generation of expectations provides the ground for verification of compliant agent behaviour. We identify the notion of compliance with that of fulfillment of expectations.

Definition 5 (Fulfillment) *Given a society instance $\mathcal{S}_{\mathbf{HAP}}$, a set of social expectations* **EXP** *is* fulfilled *if and only if for all (ground) terms p:*

$$\mathbf{HAP} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \rightarrow \neg\mathbf{H}(p)\} \nvdash \mathit{false} \tag{3}$$

Symmetrically, we define violation:

Definition 6 (Violation) *Given a society instance $\mathcal{S}_{\mathbf{HAP}}$, a set of social expectations* **EXP** *is* violated *if and only if there exists a (ground) term p such that:*

$$\mathbf{HAP} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \rightarrow \neg\mathbf{H}(p)\} \vdash \mathit{false} \tag{4}$$

Finally, we give the notion of goal achievement.

Definition 7 (Goal achievement) *Given an instance of a society, $\mathcal{S}_{\mathbf{HAP}}$, and a ground goal G, we say that G is* achieved *(and we write $\mathcal{S}_{\mathbf{HAP}} \vdash_{\mathbf{EXP}} G$) iff there exists an admissible and fulfilled set of social expectations* **EXP**, *such that:*

$$Comp(SOKB \cup \mathbf{HAP} \cup \mathbf{EXP}) \cup CET \models G \tag{5}$$

In such a case, we write:

$$\mathcal{S}_{\mathbf{HAP}} \vdash_{\mathbf{EXP}} G$$

2.3 Operational Framework

The $\mathcal{S}$CIFF proof procedure is inspired to the IFF proof procedure [Fung and Kowalski, 1997]. As well as the IFF, it is based on a transition system, which rewrites a formula into another one, until no more rewriting transitions can be applied. Each of the transitions generates one or more children from a node. $\mathcal{S}$CIFF is an extension of the IFF proof-procedure in that it is designed to cope with (*i*) universally quantified variables in abducibles, (*ii*) dynamically incoming events, (*iii*) consistency, fulfillment and violations, and (*iv*) CLP-like constraints.

Each node of the proof procedure is represented by a tuple

$$T \equiv \langle R, CS, PSIC, \mathbf{PEXP}, \mathbf{HAP}, \mathbf{FULF}, \mathbf{VIOL} \rangle$$

where

- R is a conjunction (that replaces the *R*esolvent in SLD resolution); initially set to the goal G, the conjuncts can be atoms or disjunctions (of conjunctions of atoms)
- CS is the constraint store (as in Constraint Logic Programming [Jaffar and Maher, 1994])
- $PSIC$ is a set of implications, representing *partially solved social integrity constraints*
- **PEXP** is the set of pending expectations
- **HAP** is the history of happened events
- **FULF** is a set of fulfilled expectations
- **VIOL** is a set of violated expectations

2.3.1 Initial Node and Success

A derivation D is a sequence of nodes $T_0 \rightarrow T_1 \rightarrow \cdots \rightarrow T_{n-1} \rightarrow T_n$. Given a goal G, a set of integrity constraints $\mathcal{IC}_S$, and an initial history $\mathbf{HAP}^i$ (that is typically empty) the first node is: $T_0 \equiv \langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$ i.e., the conjunction R is initially the query ($R_0 = \{G\}$) and the partially solved integrity constraints $PSIC$ is the whole set of social integrity constraints ($PSIC_0 = \mathcal{IC}_S$). The other nodes $T_j, j > 0$, are obtained by applying one of the transitions of the proof-procedure, until no further transition can be applied (we call this last condition *quiescence*).

Let us now give the definition of successful derivation.

Definition 8 *Given an initial history* $\mathbf{HAP}^i$ *that evolves toward a final history* $\mathbf{HAP}^f$ *(with* $\mathbf{HAP}^f \supseteq \mathbf{HAP}^i$*), and a society instance* $\mathcal{S}_{\mathbf{HAP}^i}$*, there exists a* successful derivation *for a goal G iff the proof tree with root node*

$$\langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$$

has at least one leaf node

$$\langle \emptyset, CS, PSIC, \mathbf{PEXP}, \mathbf{HAP}^f, \mathbf{FULF}, \emptyset \rangle$$

where CS is consistent (i.e., there exists a ground variable assignment such that all the constraints are satisfied), and **PEXP** *contains only negative literals* $\neg\mathbf{E}$ *and* $\neg\mathbf{EN}$.

In such a case, we write:

$$\mathcal{S}_{\mathbf{HAP}^i} \vdash_{\mathbf{EXP}}^{\mathbf{HAP}^f} G$$

From each non-failure leaf node N, answers can be extracted in a very similar way to the IFF proof procedure. Answers of the $\mathcal{S}$CIFF proof procedure, called *expectation answers*, are composed of an answer substitution and a set of abduced expectations. First, an answer substitution σ' is computed such that (i) σ' replaces all variables in N that are not universally quantified by a ground term, and (ii) σ' satisfies all the constraints in the store CS_N.

Definition 9 *Given a non-failure node N, let σ' be the answer substitution extracted from N.*

Let $\sigma = \sigma'|_{vars(G)}$ be the restriction of σ' to the variables occurring in the initial goal G. Let $\mathbf{EXP}_N = (\mathbf{FULF}_N \cup \mathbf{PEXP}_N)\sigma'$. The pair $(\mathbf{EXP}_N, \sigma)$ is the expectation answer *obtained from the node N.*

+ def simbolo $\vdash$

2.3.2 Transitions

$\mathcal{S}$CIFF transitions are based on those of the IFF proof procedure, augmented with those of CLP [Jaffar and Maher, 1994], and with specific transitions accommodating the concepts of fulfillment, dynamically growing history and consistency of the set of expectations with respect to the given definitions (Definitions 2, 3, and 4).

The $\mathcal{S}$CIFF proof procedure starts with a formula (that replaces the concept of *resolvent* in logic programming) built as a conjunction of the initial query and the ICs. Then it repeatedly applies one of the following *inference rules*:

unfolding replaces *resolution*: given a node with a definite atom, it replaces it with one of its definitions;

propagation propagates ic_S: given a node containing $A \wedge B \rightarrow C$ and an atom A' that unifies with A, it replaces the implication with $(A = A') \wedge B \rightarrow C$;

splitting distributes conjunctions and disjunctions, making the final formula in a sum-of-products form;

case analysis if the body of an ic_S contains $A = A'$, case analysis nondeterministically tries $A = A'$ or $A \neq A'$,

factoring tries to reuse a previously made hypothesis;

rewrite rules for equality use the inferences in the Clark Equality Theory (CET);

logical simplifications try to simplify a formula through equivalences like $[A \wedge false] \leftrightarrow false$, $[true \rightarrow A] \leftrightarrow A$, etc;

CLP-like transitions contain the CLP transitions [Jaffar and Maher, 1994] of *Constrain* (moves a constraint from R to the constraint store CS), *Infer* (infers new constraints given the current state of CS) and *Consistent* (checks if the constraint store is satisfiable);

happening inserts a new event to maintain the history **HAP** and deals with non-happening of events;

consistency, fulfillment and violation rule out nodes that are either inconsistent with respect to the declarative semantics or contain violations.

Due to lack of space, we do not give more detail on the $\mathcal{S}$CIFF transitions, but the interested reader can refer to [Alberti *et al.*, 2003b] for an exhaustive description of them.

2.4 Properties of the $\mathcal{S}$CIFF proof-procedure

The $\mathcal{S}$CIFF has been proven sound with respect to the declarative semantics:[1]

Theorem 1 (Soundness). *Given a society instance* $\mathcal{S}_{\mathbf{HAP}^f}$ *, if*

$$\mathcal{S}_{\mathbf{HAP}^i} \vdash^{\mathbf{HAP}^f}_{\mathbf{EXP}} G$$

with expectation answer $(\mathbf{EXP} \cup \mathbf{FULF}, \sigma)$*, then*

$$\mathcal{S}_{\mathbf{HAP}^f} \models_{\mathbf{EXP}\sigma} G\sigma$$

A result of termination has also been proved, when the knowledge of the society is *Acyclic*. The notion of acyclicity for an abductive logic program was stated by Xanthakos [Xanthakos, 2003], and is basically the extension of the notion of acyclic logic program used for the proof of termination of SLD resolution extended for integrity constraints.

Theorem 2 (Termination of $\mathcal{S}$CIFF) *Let G be a goal and $\mathcal{S}$ a society, such that $SOKB$, $\mathcal{IC}_S$ and G are acyclic with respect to some level mapping, and G and all implications in $\mathcal{IC}_S$ bounded w.r.t. the level-mapping. Then, every $\mathcal{S}$CIFF derivation for G for each instance of $\mathcal{S}$ is finite.*

Proof. Trivial extension of the proof of termination of the IFF proof-procedure [Xanthakos, 2003]. □

[1] The proof of soundness is contained in an annex of Deliverable D12 of the SOCS project [SOCS-web,]. It can be downloaded from the URL: `http://lia.deis.unibo.it/research/SOCS/6401-b1-FullProofMain.pdf`

3 Automatic verification of interaction protocols using $g{-}\mathcal{S}$CIFF

Intuitively, a protocol property is any formula that holds true for all the histories admitted by the protocol. Thus, given a property P of a history, we are interested in proving (or disproving) that the property holds for the protocol, without restricting its scope to only some specific history.

Proving a protocol property $\mathcal{P}$ amounts to proving that in all histories compliant to the protocol, the property holds, that is

$$\left.\begin{array}{l}\forall_{\mathbf{HAP}} SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC}_S \\ \mathbf{EXP} \text{ is fulfilled, } \neg, \mathbf{E}\text{-consistent}\end{array}\right\} \Rightarrow SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{P}$$

One way to generate such a proof is to generate, and test, each compliant history. An alternative method, also adopted by model-checking, is to negate the property and try to generate one compliant history that entails the property $\neg\mathcal{P}$: if such a history does not exist, then $\mathcal{P}$ holds in each compliant history; otherwise, the generated history is a counterexample proving an instance in which the property $\mathcal{P}$ does not hold, that can be useful for debugging and improving the protocol.

The task of automatically proving a property, amounts then to stating it in the syntax of a goal of the $\mathcal{S}$CIFF proof-procedure, and extending the proof-procedure to generate compliant histories. To this purpose, we define in the next section the $g{-}\mathcal{S}$CIFF proof-procedure as an extension of $\mathcal{S}$CIFF.

Notice also that generating a compliant history is also useful, when defining the society, even if no goal is given. In such a case, we can test if the society is *well-defined*, i.e., if there exists at least one history that satisfies the protocol.

3.1 The $g{-}\mathcal{S}$CIFF proof-procedure

Most protocols, even very simple ones, can have an infinite number of compliant histories. One can restrict himself to only finite universe situations, where the number of involved agents is finite and known a-priori, the set of possible utterances is fixed, and the language is function-free. In such cases, the number of compliant histories is finite, although they can be many of them, and one can think to generate all the compliant histories, possibly pruning the search space by means of some efficient technique.

We decided to adopt a different viewpoint: atoms in the history will contain variables, that can possibly concisely express a number of different instantiations. Thus, the atoms generated by $g{-}\mathcal{S}$CIFF will have the same syntax as the happened events of the (non-generative) $\mathcal{S}$CIFF proof-procedure, but will not be required to be ground. Variables in **H** atoms will be considered existentially quantified, as we search for at least one compliant history violating the property.

Since **H** literals will be generated, we need more flexibility than in the original $\mathcal{S}$CIFF. In the abductive framework we adopted, it is quite natural to map them onto abducible literals. In fact, from a declarative viewpoint, such a choice is rather harmless, as in all the formulas that define the declarative semantics, (in particular, Def. 2 and 7), the history (**HAP**) occurs on the left hand side of the entailment symbol, just like the sets of the abducible atoms (**EXP**).

In the operational semantics, the transition *Happening* (which inserts a new event in the current history, see Sec. 2.3.2) is no longer needed (as this is not on-the-fly verification), and it is replaced by the following transition.

Fulfiller. Given a node N_k in which

- $\mathbf{PEXP}_k = \mathbf{PEXP}' \cup \{\mathbf{E}(E,T)\}$

and transitions of fulfillment are not applicable, transition *Fulfiller* is applicable and generates a node N_{k+1} identical to N_k except:

- $\mathbf{PEXP}_{k+1} = \mathbf{PEXP}'$
- $\mathbf{FULF}_{k+1} = \mathbf{FULF}_k \cup \{\mathbf{E}(E,T)\}$
- $\mathbf{HAP}_{k+1} = \mathbf{HAP}_k \cup \{\mathbf{H}(E,T)\}$

i.e., a new event is inserted in the history fulfilling the expectation.

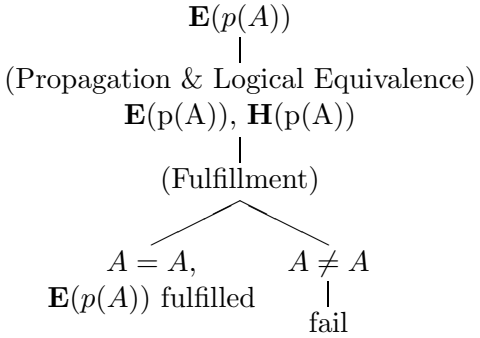
We will use the symbol $\overset{g}{\vdash}$ to indicate a derivation in the generative version of the $\mathcal{S}$CIFF proof-procedure. The generative version, as the on-the-fly one, takes as input an initial history $\mathbf{HAP}^i$ (that will be typically empty) and provides a possibly extended history $\mathbf{HAP}^f$:

$$\mathcal{S}_{\mathbf{HAP}^i} \overset{g}{\vdash}^{\mathbf{HAP}^f}_{\mathbf{EXP}} G$$

Note that we would have obtained a semantically equivalent result by imposing the integrity constraint

$$\mathbf{E}(X,T) \rightarrow \mathbf{H}(X,T) \tag{6}$$

In fact, the derivation tree for a single abduced atom $\mathbf{E}(p(A))$, would have been the following:



Despite their semantical equivalence, transition *Fulfiller* is more efficient, as it avoids generating the right failure branch.

3.2 Formal results

The formal properties of $g{-}\mathcal{S}$CIFF are a consequence of those of $\mathcal{S}$CIFF.

Theorem 3 (Soundness of $g{-}\mathcal{S}$CIFF). *Given a society instance $\mathcal{S}_{\mathbf{HAP}^f}$, if*

$$\mathcal{S}_{\mathbf{HAP}^i} \overset{g^{\mathbf{HAP}^f}}{\vdash_{\mathbf{EXP}}} G$$

with expectation answer $(\mathbf{EXP}, \sigma)$, then

$$\mathcal{S}_{\mathbf{HAP}^f} \models_{\mathbf{EXP}\sigma} G\sigma$$

Proof. As previously noted, transition *Fulfiller* retains the same behaviour of a sequence of $\mathcal{S}$CIFF transitions, so the $g{-}\mathcal{S}$CIFF is sound with respect to a society with added the ic_S (6). Adding the ic_S (6) does not change the declarative semantics of a society, since it is already included in Definition 5. □

Concerning termination, $g{-}\mathcal{S}$CIFF guarantees termination if the society *together* with the ic_S (6) is acyclic. However, it turns out that some of the protocols that can be expressed through an acyclic society knowledge may become cyclic when adding rule (6), and consequently the proof of termination is no longer valid. In such a case, we can rely on the idea of iterative deepening, as many other approaches do (one of them is bounded model checking). For instance, in the context of verification of a security protocol against possible attacks, if we implement iterative deepening we search for attacks of increasing length. We first focus only on attacks of length 1, and if none exists, we look for attacks of length 2, and so on. This method is, of course, unable to prove that a protocol is secure, but it can prove that no attack exists up to any given length n (provided that we have enough time). However, other protocols will retain the acyclicity even with respect to the society knowledge base of (6), so in such cases we have a greater expressive power than bounded methods.

4 Case studies

In previous work [Alberti *et al.*, 2004a; Alberti *et al.*, 2004c], we have shown the application of $\mathcal{S}$CIFF to the on-the-fly verification of compliance.

In this section, we exemplify the use of $g{-}\mathcal{S}$CIFF for the static verification of properties, focusing on two well known interaction protocols: the NetBill transaction protocol and the Needham-Schroeder security protocol.

4.1 Verifying the NetBill protocol

NetBill (see [Cox *et al.*, 1995]) is a security and transaction protocol optimized for the selling and delivery of low-priced information goods, like software or journal articles. The protocol rules transactions between two agents: *merchant* and *customer*. A NetBill server is used to deal with financial issues such as those related to credit card accounts of customer and merchant.

In this paper, we focus on the type of the NetBill protocol designed for non zero-priced goods, and do not consider the variants that deal with zero-priced goods.

A typical protocol run is composed of three phases:

1. *price negotiation.* The customer requests a quote for a good identified by *PrId* (`priceRequest(PrId)`), and the merchant replies with (`priceQuote(PrId,Quote)`).

2. *good delivery.* The customer requests the good (`goodRequest(PrId,Quote)`) and the merchant delivers it in an encrypted format (`deliver(crypt(PrId,Key),Quote)`).

3. *payment.* The customer issues an Electronic Payment Order (EPO) to the merchant, for the amount agreed for the good (`payment(epo(C,crypt(PrId,K),Quote))`); the merchant appends the decryption key for the good to the EPO, signs the pair and forwards it to the NetBill server (`endorsedEPO(epo(C,crypt(PrId,K),Quote),M)`); the NetBill server deals with the actual money transfer and returns the result to the merchant (`signedResult(C,PrID,Price,K)`), who will, in her turn, send a receipt for the good and the decryption key to the customer (`receipt(PrId,Price,K)`).

The customer can withdraw from the transaction until she has issued the *EPO* message; the merchant until she has issued the *endorsedEPO* message.

NetBill protocol specification based on ic_S. The NetBill protocol is implemented in the $\mathcal{S}$CIFF framework by means of ic_S. Conceptually, ic_S are of two types:

- *backward integrity constraints* (Fig. 1), i.e., integrity constraints that state that if some set of event happens, then some other set of event is expected to have happened before.

  For instance, the first backward integrity constraints imposes that, if `M` has sent a `priceQuote` message to `C`, stating that `M`'s quote for the good identified by `PrId` is `Quote`, in the interaction identified by `Id`, then `C` is expected to have sent to `M` a `priceRequest` message for the same good, in the same interaction, at an earlier time;

```
H(tell(M,C,priceQuote(PrId,Quote),Id),T)
--->
E(tell(C,M,priceRequest(PrId),Id),T2) /\ T2 < T.

H(tell(C,M,goodRequest(PrId,Quote),Id),T)
--->
E(tell(M,C,priceQuote(PrId,Quote),Id),Tpri) /\ Tpri < T.

H(tell(M,C,goodDelivery(crypt(PrId,K),Quote),Id),T)
--->
E(tell(C,M,goodRequest(PrId,Quote),Id),Treq) /\ Treq < T.

H(tell(C,M,payment(C,crypt(PrId,K),Quote),Id),T)
--->
E(tell(M,C,goodDelivery(crypt(PrId,K),Quote),Id),Tdel) /\ Tdel <
T.

H(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign)

/\ M != netbill
--->
E(tell(M,netbill,endorsedEPO(epo(C,PrId,Quote),K,M),Id),T) /\ T
< Tsign.

H(tell(M,C,receipt(PrId,Quote,K),Id),Ts)
--->
E(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign) /\
Tsign < Ts.
```

Figure 1. *NetBill protocol: backward integrity constraints*

- *forward integrity constraints* (Fig. 2), i.e., constraints that state that if some conjunction of event has happened, then some other set of event is expected to happen in the future.

  For instance, the first forward integrity constraint in Fig. 2 imposes that an `endorsedEPO` message from `M` to the *netbill* server be followed by a `signedResult` message, with the corresponding parameters.

  We only impose forward constraints from the `endorsedEPO` message onwards, because both parties (merchant and customer) can withdraw from the transaction at the previous steps.

```
H(tell(M,netbill,endorsedEPO(epo(C,PrId,Quote),K,M),Id),T)
--->
E(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign) /\ T <
Tsign.

H(tell(netbill,M,signedResult(C,PrId,Quote,K),Id),Tsign)
--->
E(tell(M,C,receipt(PrId,Quote,K),Id),Ts) /\ Tsign < Ts.
```

Figure 2. *NetBill protocol: forward integrity constraints*

4.1.1 Verification of NetBill properties

In this section, we show how a simple property of the NetBill protocol can be expressed, and verified, with $\mathcal{S}$CIFF.

We want to verify the following property: *the merchant receives the payment for a good G if and only if the customer receives the good G*, as long as the protocol is respected.

Since the $\mathcal{S}$CIFF deals with (communicative) events and not with the states of the agents, we need to express the properties in terms of happened events. To this purpose, we can assume that merchant has received the payment once the NetBill server has issued the *signedResult* message, and that the the customer has received the good if she has received the encrypted good (with a *deliver* message) and the encryption key (with a *receipt* message).

Thus, the property we want to verify can be espressed as

$$\begin{aligned} & H(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ \iff & H(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \wedge H(tell(M, C, receipt(PrId, Quote, K), Id), Ts) \end{aligned} \tag{7}$$

whose negation is

$$\begin{aligned} & (\neg H(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge H(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \wedge H(tell(M, C, receipt(PrId, Quote, K), Id), Ts)) \\ & \vee \\ & (H(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \neg H(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T) \\ & \vee \\ & (H(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign) \\ & \wedge \neg H(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T)) \end{aligned} \tag{8}$$

In other words, an history that entails Eq. (8) is a counterexample of the property that we want to prove. In order to search for such a history, we define the $\mathcal{S}$CIFF goal as follows:

$$
\begin{aligned}
g \leftarrow & EN(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\
& E(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T), \\
& E(tell(M, C, receipt(PrId, Quote, K), Id), Ts)). \\
g \leftarrow & E(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\
& EN(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T). \\
g \leftarrow & E(tell(netbill, M, signedResult(C, PrId, Quote, K), Id), Tsign), \\
& EN(tell(M, C, goodDelivery(crypt(PrId, K), Quote), Id), T))
\end{aligned}
\tag{9}
$$

and run $\mathcal{S}$CIFF, with the *fulfiller*, and the integrity constraints that define the NetBill protocol.

The result of the call is a failure. This suggests that there is no history that entails the negation of the property while respecting the protocol, i.e., the property is likely to hold if the protocol is respected. However, yet no guarantee can be given unless $g-\mathcal{S}$CIFF is proven complete.

If we remove the second forward integrity constraints (which imposes that a `signedResult` message be followed by a `receipt` message), then the following history is generated:

```
h(tell(_E,_F,priceRequest(_D),_C),_M),
h(tell(_F,_E,priceQuote(_D,_B),_C),_L),
h(tell(_E,_F,goodRequest(_D,_B),_C),_K),
h(tell(_F,_E,goodDelivery(crypt(_D,_A),_B),_C),_J),
h(tell(_E,_F,payment(_E,crypt(_D,_A),_B),_C),_I),
h(tell(_F,netbill,endorsedEPO(epo(_E,_D,_B),_A,_F),_C),_H),
h(tell(netbill,_F,signedResult(_E,_D,_B,_A),_C),_G),
_I<_H, _H<_G,
_L>_M, _K>_L, _I>_J, _J>_K,
```

The *receipt* event is missing, which would violate the integrity constraint that has been removed.

In this way, a protocol designer can make sure that an integrity constraint is not redundant with respect to a desired property of the protocol.

4.2 Verifying the Needham-Schroeder Protocol

The Needham-Schroeder authentication protocol [Needham and Schroeder, 1978] consists of seven steps, but – as other authors have previously done – we focus on a

(1) $A \rightarrow B : \{N_A, A\}_{pub\_key(B)}$

(2) $B \rightarrow A : \{N_A, N_B\}_{pub\_key(A)}$

(3) $A \rightarrow B : \{N_B\}_{pub\_key(B)}$

Figure 3. *The Needham-Schroeder protocol (simplified version)*

simplified version consisting of only three steps, which are the kernel of the protocol. The aim of the protocol is to establish a connection between two agents, A and B, in which the agent A who initiates the protocol is authenticated with B who responds to A. In support of the authentication procedure, agents rely on the well-known public key encryption technology. Thanks to such a technology, an agent is able to generate two keys, a public key which is made available to the other agents, and a private key which must remain undisclosed. A sequence of bytes encrypted using the public key can be decrypted only by using the corresponding private key. The idea of the protocol is to challenge the fellow agents in a communication session (conversation), to make sure that he is actually the holder of the private key associated with his public key.

During the authentication phase, agents can generate some special items of data called *nonces*. Therefore, during the conversation, if agent A sends a nonce N_A generated by himself to B, and if A sends N_A encrypted with the public key of B, only B will be able to decrypt N_A and send it back to A. A will then know that the agent to whom he sent N_A is actually the holder of B's private key.

The three messages of the protocol that we consider in this paper are those listed in Figure 3.

Basically, with the simplified version, we assume that all the agents know the public key of the other agents.

By message (1) A challenges B to decrypt his nonce N_A encrypted using B's public key. By message (2) B responds to A's challenge, by attaching to N_A a new nonce N_B, which he generated himself, and encrypting the whole set of two nonces using A's public key, thus challenging A to decrypt N_B and prove to be the holder of A's private key. By message (3) A responds to B's challenge.

Formalizing agent's authentication In the idea of the Needham-Schroeder protocol, an agent trusts the identity of the agent with whom he is communicating by associating his name with his public key and receiving back a nonce that he forged, encrypted in his own public key. If we had to define the idea of an agent B 'trusting' that he is communicating with A, we could do it by using a combination of messages in which an agents responds to a challenge posed by another agent and successfully

decrypts a nonce.

Definition 10 *We say that B trusts that the agent X he is communicating with is A,*[2] *and we write $trust_B(X, A)$ once two messages have been exchanged at times T_1 and T_2, $T_1 < T_2$, having the following sender, recipient, and content:*

$$(T_1)\ B \rightarrow X : \{N_B, \dots\}_{pub\_key(A)}$$

$$(T_2)\ X \rightarrow B : \{N_B, \dots\}_{pub\_key(B)}$$

where N_B is a nonce generated by B.

Note that B is unable to judge whether N_A is a nonce actually generated by X or not, therefore no condition is posed on the origin of such nonce.

Symmetrically, we can consider, from A's viewpoint, messages (1) and (2) as those that prove the identity of B. We therefore implement Def. 10 in Def. 11, where messages are expressed using the notation of the $\mathcal{S}$CIFF, namely as events which are part of some "history" **HAP**. The content of messages will be composed of three parts, the first showing the public key used to encrypt it, the second and third containing agent names or nonces or nothing (in particular, the last part may be empty).

Definition 11 *Let A, B and X be agents, K_A and K_B respectively A's and B's public key, N_B a nonce produced by B, and let $\mathbf{HAP}_1$ and $\mathbf{HAP}_2$ be two sets of events each composed of two elements, namely:*

$\mathbf{HAP}_1 = \{$
$\mathbf{H}(send(B, X, content(key(K_A), agent(B), nonce(N_B))), T_1),$
$\mathbf{H}(send(X, B, content(key(K_B), nonce(N_B), nonce(...))), T_2)$
$\}$, *and*
$\mathbf{HAP}_2 = \{$
$\mathbf{H}(send(B, X, content(key(K_A), nonce(...), nonce(N_B))), T_1),$
$\mathbf{H}(send(X, B, content(key(K_B), nonce(N_B), empty(0))), T_2)$
$\}$.

Then, $trust_B(X, A)$ holds if and only if $\mathbf{HAP}_1 \subseteq \mathbf{HAP}$ or $\mathbf{HAP}_2 \subseteq \mathbf{HAP}$.

Lowe's attack on the protocol It turns out that (at least) one situation may occur in which B trusts that he is proving his own identity to another agent A, by following this protocol, but in fact a third agent I (standing for *intruder*) manages to successfully pretend that he is A and authenticate himself as A with B. This attack was suggested by Lowe [Lowe, 1996], and it consists of the messages listed in Figure 4.

[2] We restrict ourselves to only one communication session, all the definitions will therefore have as a scope the session.

(1) $A \rightarrow I : \{N_A, A\}_{pub\_key(I)}$

(2) $I(A) \rightarrow B : \{N_A, A\}_{pub\_key(B)}$

(3) $B \rightarrow I(A) : \{N_A, N_B\}_{pub\_key(A)}$

(4) $I \rightarrow A : \{N_A, N_B\}_{pub\_key(A)}$

(5) $A \rightarrow I : \{N_B\}_{pub\_key(I)}$

(6) $I(A) \rightarrow B : \{N_B\}_{pub\_key(B)}$

Figure 4. *Lowe's attack on the Needham-Schroeder protocol*

It is a nesting of the Needham-Schroeder protocol, in which A happens to start a conversation with I, thus transmitting him his nonce N_A. Instead of answering to the challenge with a new nonce N_I, I exploits the information contained in A's request for authentication (namely, its nonce) to handcraft a message to send to B. Such a message (2) will be encrypted using B's public key, and will contain A's name along with A's nonce N_A. I therefore sends this message pretending that he is A (we use the notation $I(A)$ for this purpose). I will reply to the challenge contained in message (2) by generating a nonce N_B and encrypting N_A and N_B together using A's public key. Since I is unable to decrypt a message encrypt with another agent's public key, I simply forwards B's message to A. This is understood by A as the continuation of the protocol initiated with message (1). In this way, I manages to receive back from A the nonce N_B encrypted using his own public key (message 5), and to respond to B with message (6).

Messages (1), (4) and (5) represent a conversation between A and I, while (2), (3), and (6) represent a conversation between $I(A)$ and B; both conversations happen to be compliant to the protocol. But, as we have seen, a combination of two compliant conversations generates a situation in which an agent authenticates himself with an identity which is not his own.

It is important to stress that in the attack proposed by Lowe it is never the case that an intruder manages to guess a nonce or a private key. In particular, initially only agent A knows the content of its own nonce N_A and only B knows the content of its own nonce N_B, and an agent knows the content of a nonce if either he initially knows it or if it is sent to him encrypted in his own public key.

However, we consider the protocol under the assumption that the communication channel is insecure:

1. when an agent sends a message to another agent, the sender has no way to know if the message has been received or not;

2. when an agent receives a message, there is no way to be sure about the sender, unless this information is somehow coded into the payload;

3. the content of a message could be compromised someway.

From the perspective of messages that can be produced and sent by agents, we can express these assumptions by way of ic_S. In particular, we can say that an agent X can send to another agent Y a message containing a nonce N_X which he does not initially know only if one of the following two cases hold: either (i) X received N_X from another agent, encrypted in X's own public key, or (ii) X received a message containing N_X and encrypted with a public key K_Y, in which case X can forward exactly the same message, without operating any modification on it.

Such integrity constraints about the impossibility to guess a nonce are shown in Fig. 5. In order to maintain relevant information about the ownership of public keys and nonces, we define a number of predicates in the Social Organization Knowledge Base, ash shown in Fig. 6.

Let us now proceed to the presentation of results about verification of the Needham-Schroeder authentication protocol. We will focus our attention on one specific property: that is, given two agents B and X, and a sequence of messages which comply to the protocol, it is never the case that B trusts that the agent X he is communicating with is A ($trust_B(X, A)$) while X is not A but instead some other agent $I(A)$.

Needham-Schroeder protocol specification based on ic_S We start by specifying the Needham-Schroeder protocol. The relevant ic_S are shown in Fig. 7. The first ic_S of Fig. 7 expresses that if a message is sent from X to B, containing the name of some agent A and some nonce N_A, encrypted together with some public key K_B:

$$\mathbf{H}(send(X, B, content(key(K_B), agent(A), nonce(N_A))), T1) \in \mathbf{HAP},$$

then a message is expected to be sent at a later time (and by some deadline T_{Max}) from B to X , containing the original nonce N_A and a new nonce N_B, encrypted with the public key of A:

$$\mathbf{E}(send(B, X, content(key(K_A), nonce(N_A), nonce(NB))), T2)$$

is therefore generated and put into **EXP**.

The second ic_S of Fig. 7 expresses that if a message of the protocol is sent from X to B, containing the name of some agent A and some nonce N_A, encrypted together with some public key K_B:

$$\mathbf{H}(send(X, B, content(key(K_B), agent(A), nonce(N_A))), T1) \in \mathbf{HAP},$$

```
    H( send( X, Y, content( key( KY), agent( W), nonce( NX))), T1)
    /\ X!=Y /\ notIsNonce( X, NX)
--->
    E( send( V, X, content( key( KX), agent( V), nonce( NX))), T0)
    /\ isAgent( V) /\ X!=V /\ isPublicKey( X, KX) /\ isNonce( V, NX)
    /\ T0 < T1 /\ T0 > 0
\/
    ...

    H( send( X, Y, content( key( KY), nonce( NX), nonce( NY))), T1)
    /\ X!=Y /\ notIsNonce( X, NX)
--->
    E( send( Z, X, content( key( KX), agent( V), nonce( NX))), T0)
    /\ isAgent( V) /\ isAgent( Z) /\ X!=V /\ Z!=X /\ isPublicKey( X, KX)
    /\ T0 < T1 /\ T0 > 0
\/
    ...

    H( send( X, Y, content( key( KY), nonce( NX), empty( 0))), T1)
    /\ X!=Y /\ notIsNonce( X, NX)
--->
    E( send( Y, X, content( key( KX), nonce( NW), nonce( NX))), T0)
    /\ isPublicKey( X, KX) /\ isNonce( NW) /\ NW!=NX
    /\ T0 < T1 /\ T0 > 0
\/
    E( send( Z, X, content( key( KX), nonce( NX), empty( 0))), T0)
    /\ isPublicKey( X, KX) /\ isAgent( Z) /\ X!=Z /\ Y!=Z
    /\ T0 < T1 /\ T0 > 0
\/
    ...
```

Figure 5. *Social Integrity Constraints expressing that an agent cannot guess the content of a* nonce

```
isPublicKey( PK) :-
   isPublicKey( _, PK).

isPublicKey( i, ki).
isPublicKey( b, kb).
isPublicKey( a, ka).

isMaxTime( 7).

isAgent( i).
isAgent( a).
isAgent( b).
```

```
isNonce( N) :-
    isNonce( _, N).

isNonce( A, N) :-
   checkIfNonce( A, N).

notIsNonce( A, NB) :-
   \+( checkIfNonce( A, NB)).

checkIfNonce( b, nb).
checkIfNonce( a, na).
```

Figure 6. *SOKB*

```
   H( send( X, B, content( key( KB), agent( A), nonce( NA))), T1)
--->
   E( send( B, X, content( key( KA), nonce( NA), nonce( NB))), T2)
   /\ isPublicKey( A, KA) /\ isNonce( NB) /\ NA!=NB
   /\ isMaxTime( TMax) /\ T2 > T1 /\ T2 < TMax.

   H( send( X, B, content( key( KB), agent( A), nonce( NA))), T1)
   /\ H( send( B, X, content( key( KA), nonce( NA), nonce( NB))), T2)
   /\ T2 > T1
--->
   E( send( X, B, content( key( KB), nonce( NB), empty( 0))), T3)
   /\ isMaxTime( TMax) /\ T3 > T2 /\ T3 < TMax.
```

Figure 7. *Social Integrity Constraints defining the Needham-Schroeder protocol*

and a message is sent at a later time from B to X, containing the original nonce N_A and a new nonce N_B, encrypted with the public key of A:

$$\mathbf{H}(send(B, X, content(key(K_A), nonce(N_A), nonce(N_B))), T2) \in \mathbf{HAP},$$

then a third message is expected to be sent from X to B, containing N_B, and encrypted with the public key of B:

$$\mathbf{E}(send(X, B, content(key(KB), nonce(NB), empty(0))), T3)$$

is therefore generated and put into **EXP**.

Generation of compliant histories A first result that we obtain by running the $g{-}\mathcal{S}$CIFF is that, given as a social goal the expectation about some event, the proof-procedure is able to generate a compliant history which includes such event.

For instance, given the goal *g1* representing the start of a protocol run between a and i:

$$g1 \leftarrow \mathbf{E}(send(a, i, content(key(ki), agent(a), nonce(na))), 1),$$

and given a deadline of 6 "time units" to the completion of the protocol, the execution of the proof returns the following compliant history:
$\mathbf{HAP}_{g1} = \{$
$\mathbf{H}(send(a, i, content(key(ki), agent(a), nonce(na))), 1),$
$\mathbf{H}(send(i, a, content(key(ka), nonce(na), nonce(nb))), T_A), T_A \in [2..5],$
$\mathbf{H}(send(a, i, content(key(ki), nonce(nb), empty(0))), T_B), T_B \in [3..6], T_B > T_A$
$\},$
while given the goal *g2*, representing the last step of a protocol run between i and b:

$$g2 \leftarrow \mathbf{E}(send(i, b, content(key(kb), nonce(nb), empty(0))), 6),$$

and again a range of 6 "time units" to the completion of the protocol (from time 1 to time 6), it is possible to obtain a compliant history such as the following:
$\mathbf{HAP}_{g2} = \{$
$\mathbf{H}(send(a, i, content(key(ki), agent(a), nonce(na))), T_C), T_C \in [1..3],$
$\mathbf{H}(send(i, a, content(key(ka), nonce(na), nonce(nb))), T_D), T_D \in [2..5], T_D > T_C$
$\mathbf{H}(send(a, i, content(key(ki), nonce(nb), empty(0))), T_E), T_E \in [3..6], T_E > T_D$
$\mathbf{H}(send(i, b, content(key(kb), agent(i), nonce(na))), T_B), T_B \in [2..4], T_B > T_C$
$\mathbf{H}(send(b, i, content(key(ki), nonce(na), nonce(nb))), T_A), T_A \in [3..5], T_A > T_B$
$\mathbf{H}(send(i, b, content(key(kb), nonce(nb), empty(0))), 6)$
$\}.$

It is worthwhile noticing that $\mathbf{HAP}_{g2}$ contains two possibly interleaved communication sessions (one between a and i and another between i and a) which do not represent an attack to the protocol. In fact, it is not the case that $trust_X(Y, W)$ and $\neg trust_W(Y, X)$, for all X, Y and W. What happens is, i uses to communicate with b the content of the nonce na obtained from a (but does not pretend to be himself a). This is in fact perfectly allowed by the protocol and does not contradict the assumptions on the generation of nonces specified by the constraints of Fig. 5.

Generation of Lowe's attack A second important result, which shows how the $g{-}\mathcal{S}$CIFF can effectively be used for protocol verification, is the generation of Lowe's attack. The property that we want to disprove is $\mathcal{P}_{trust}$ defined as $trust_B(X, A) \rightarrow X = A$, i.e., if B trusts that he is communicating with A, then he is indeed communicating with A. We obtain a problem which is symmetric in the variables A, B, and X. In order to check if we have a solution we can ground $\mathcal{P}_{trust}$ and define its negation $\neg\mathcal{P}_{trust}$ as a goal, $g3$, where we choose to assign to A, B, and X the values a, b and i:

$g3 \leftarrow isNonce(NA), NA \neq nb,$
$\mathbf{E}(send(b, i, content(key(ka), nonce(NA), nonce(nb))), 3),$
$\mathbf{E}(send(i, b, content(key(kb), nonce(nb), empty(0))), 6).$

Besides defining $g3$ for three specific agents, we also assign definite time points (3 and 6) in order to improve the efficiency of the proof.

Running the $g{-}\mathcal{S}$CIFF on $g3$ results in a compliant history:

$\mathbf{HAP}_{g3} = \{$
$h(send(a, i, content(key(ki), agent(a), nonce(na))), 1),$
$h(send(i, b, content(key(kb), agent(a), nonce(na))), 2),$
$h(send(b, i, content(key(ka), nonce(na), nonce(nb))), 3),$
$h(send(i, a, content(key(ka), nonce(na), nonce(nb))), 4),$
$h(send(a, i, content(key(ki), nonce(nb), empty(0))), 5),$
$h(send(i, b, content(key(kb), nonce(nb), empty(0))), 6)$
$\},$

which is indeed Lowe's attack on the protocol. $\mathbf{HAP}_{gL}$ represents a counterexample of the property $\mathcal{P}_{trust}$.

5 Related Work

In recent years the provability of properties for communication protocols has received a lot of attention; this holds even more for security protocols, since they are used for exchanging sensitive information or for electronic payments. Various techniques have been adopted to the task of automatic verification of properties, and many, in particular, have been applied to the case study of the Needham-Shroeder, or its evolution proposed by Lowe.

One way to prove/disprove the security of the protocols is the cryptographic approach, whose security definitions are based on complexity theory. Such an approach have been used for proofs by hand [Goldwasser *et al.*, 1989] or, more recently, automatically [Backes and Pfitzmann, 2003]. Theorem provers, such as Isabelle/HOL [Nipkow *et al.*, 2002] have also been applied to such a task, also together with tools for graphically representing and defining the protocols [von Oheimb and Lotz, 2002]. Another viewpoint is to embody a possible intruder and plan for an attack [Aiello and Massacci, 2002]: if a planner succeeds in developing such a plan, then the protocol is, clearly, flawed.

Dixon et al. [2004a], specify security protocols in $KL_{(n)}$, a language for representing the *Temporal Logic of Knowledge*. $KL_{(n)}$ contains both a linear-time temporal logic and a modal connective for representing what the various agents know. They use clausal resolution to automatically prove properties of security protocols. As an example, they give the specifications of the Needham-Schroeder protocol in $KL_{(n)}$, then they show how to apply clausal resolution for proving formal properties. In a technical report [Dixon *et al.*, 2004b], they define a proof system based on resolution

rules in a sequent style notation, and a temporal resolution algorithm, then applied to the mentioned protocol. They then show the derivation of some properties of the protocol, for example, that an intruder does not know the sensitive information exchanged by other agents following the protocol. Finally, they show Lowe's attack on the protocol, and state that, if they do not assume that an agent A can send a message to an agent B only encrypted with B's public key, the previous properties (on the intruder's ignorance) cannot be proven. Raimondi and Lomuscio [Raimondi and Lomuscio, 2004] also use a temporal logic enriched with epistemic connectives for representing the agents' knowledge, but exploit efficient data structures (namely, Ordered Binary Decision Diagrams) to improve the efficiency. The $g-\mathcal{S}$CIFF presented in this article, borrowing the idea from model-checking, is able not only to prove properties expressed in the $\mathcal{S}$CIFF language, but can also provide a counterexample. In our example, $g-\mathcal{S}$CIFF was both able to automatically prove that the Needham-Shroeder protocol is flawed, and to generate the Lowe's attack.

Armando et al [Armando *et al.*, 2004] compile a security program into a logic programming program with choice lp-rules with answer set semantics. They search for attacks of length k, for increasing values of k, and they are able to derive the flaws of various flawed security protocols in a library. They model explicitly the capabilities of the intruder, while we take the opposite viewpoint: we explicitly state what the intruder cannot do (like decrypting a message without having the key, or guessing the key or the nonces of an agent), without explicitly limiting the abilities of the intruder.

Among other approaches to security protocols verification we cite those developed using hereditary Harrop formulas [Delzanno, 2001], process-algebraic languages [Pang, 2002], model-checking with pre-configuration [Kim *et al.*, 2004], proof-theory [Delzanno and Etalle, 2001].

6 Conclusion

In this article, we approached agent interaction verification from the perspective of an open environment, where agents interact with each other by message exchange. Building on previous work, we showed and integrated two aspects of agent interaction verification: agent compliance to protocols, and verification of protocol properties, based on a unified formalism (ic_S) to specify agent interaction. For this purpose, we defined the $g-\mathcal{S}$CIFF proof-procedure, which is an enhanced version of the $\mathcal{S}$CIFF proof-procedure in the direction of automatic generation of compliant sequences of events. We studied the properties of the $g-\mathcal{S}$CIFF in terms of correctness and guaranteed termination, and we discussed two case studies to show how the outcomes of the $g-\mathcal{S}$CIFF can be used to prove that some protocol is "well-defined" or as a counterexample to disprove some desired protocol property.

The main contribution of this work has to be found (i) in the definition of the

$g-\mathcal{S}$CIFF, which represents per se an alternative to other existing technologies for protocol verification, and (*ii*) in the integration of two aspects of verification which have so far, to the best of our knowledge, remained separate. Thanks to such integration, it will be possible to verify the specifications of interaction protocols at design time, and to directly use such specifications at run-time, for dynamic verification of compliance.

The $g-\mathcal{S}$CIFF has been implemented as an extension of the $\mathcal{S}$CIFF, using Constraint Handling Rules [Frühwirth, 1998] and SICStus Prolog [SICStus, 2003]. The experimental results discussed in this paper show the feasibility of the approach and the expressivity and readability of the specifications needed to perform the verification. Using the declarative language of ic_S it is possible to define well-known interaction protocols in only a few lines. By relying on a constraint solver, we are able to reason about variables ranging over domains, and to express concepts such a deadlines and ordering relationships.

Indeed, the counterpart of such an expressive framework is its computational complexity. In the future, we intend to work on the complexity aspects of the $g-\mathcal{S}$CIFF and to improve the efficiency of its implementation, by testing different search strategies.

Acknowledgements

This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-32530 SOCS project, within the Global Computing proactive initiative, and by the MIUR COFIN 40% projects *Sviluppo e verifica di sistemi multiagente basati sulla logica*, and *La Gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici*.

References

[Aiello and Massacci, 2002] Luigia Carlucci Aiello and Fabio Massacci. Planning attacks to security protocols: Case studies in logic programming. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, volume 2407 of *Lecture Notes in Computer Science*, pages 533–560. Springer, 2002.

[Alberti *et al.*, 2003a] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. An Abductive Interpretation for Open Societies. In A. Cappelli and F. Turini, editors, *AI*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa*, volume

2829 of *Lecture Notes in Artificial Intelligence*, pages 287–299. Springer-Verlag, September 23–26 2003.

[Alberti *et al.*, 2003b] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Specification and verification of interaction protocols: a computational logic approach based on abduction. Technical Report CS-2003-03, Dipartimento di Ingegneria di Ferrara, Ferrara, Italy, 2003. Available at `http://www.ing.unife.it/aree_ricerca/informazione/cs/technical_reports`.

[Alberti *et al.*, 2004a] M. Alberti, D. Daolio, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interaction protocols in a logic-based system. In Hisham M. Haddad, Andrea Omicini, and Roger L. Wainwright, editors, *Proceedings of the 19th Annual ACM Symposium on Applied Computing (SAC 2004). Special Track on Agents, Interactions, Mobility, and Systems (AIMS)*, pages 72–78, Nicosia, Cyprus, March 14–17 2004. ACM Press.

[Alberti *et al.*, 2004b] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The SOCS computational logic approach for the specification and verification of agent societies. In *Global Computing Workshop, Rovereto, Italy, March 2004*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2004. to appear.

[Alberti *et al.*, 2004c] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Modeling interactions using *Social Integrity Constraints*: A resource sharing case study. In João Alexandre Leite, Andrea Omicini, Leon Sterling, and Paolo Torroni, editors, *Declarative Agent Languages and Technologies*, volume 2990 of *Lecture Notes in Artificial Intelligence*, pages 243–262. Springer-Verlag, May 2004. First International Workshop, DALT 2003. Melbourne, Australia, July 2003. Revised Selected and Invited Papers.

[Armando *et al.*, 2004] Alessandro Armando, Luca Compagna, and Yuliya Lierler. Automatic compilation of protocol insecurity problems into logic programming. In José Júlio Alferes and João Alexandre Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, volume 3229 of *Lecture Notes in Computer Science*, pages 617–627. Springer, 2004.

[Backes and Pfitzmann, 2003] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In Paritosh K. Pandya and Jaikumar Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings*, volume 2914 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003.

[Clarke *et al.*, 2000] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000.

[Cox *et al.*, 1995] B. Cox, J.C. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, New York, July 1995.

[Delzanno and Etalle, 2001] Giorgio Delzanno and Sandro Etalle. Proof theory, transformations, and logic programming for debugging security protocols. In A. Pettorossi, editor, *Logic Based Program Synthesis and Transformation : 11th International Workshop, (LOPSTR 2001). Selected papers.*, volume 2372 of *Lecture Notes in Computer Science*, pages 76–90, Paphos, Cyprus, November 2001. Springer Verlag.

[Delzanno, 2001] Giorgio Delzanno. Specifying and debugging security protocols via hereditary Harrop formulas and λProlog - a case-study -. In Herbert Kuchen and Kazunori Ueda, editors, *Functional and Logic Programming, 5th International Symposium, FLOPS 2001, Tokyo, Japan, March 7-9, 2001, Proceedings*, volume 2024 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2001.

[Dixon *et al.*, 2004a] Claire Dixon, Mari-Carmen Fernández Gago, Michael Fisher, and Wiebe van der Hoek. Using temporal logics of knowledge in the formal verification of security protocols. In *Procedings of the Eleventh International Workshop on Temporal Representation and Reasoning (TIME'04)*, 2004.

[Dixon *et al.*, 2004b] Claire Dixon, Mari-Carmen Fernández Gago, Michael Fisher, and Wiebe van der Hoek. Using temporal logics of knowledge in the formal verification of security protocols. Technical Report ULCS-03-022, University of Liverpool, Department of Computer Science, Liverpool, UK, 2004. `http://www.csc.liv.ac.uk/research/techreports/`.

[Frühwirth, 1998] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3):95–138, October 1998.

[Fung and Kowalski, 1997] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, November 1997.

[Goldwasser *et al.*, 1989] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.

[Jaffar and Maher, 1994] J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[Kacprzak *et al.*, 2004] Magdalena Kacprzak, Alessio Lomuscio, and Wojciech Penczek. Verification of multiagent systems via unbounded model checking. In N. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2004)*, pages 638–645, Columbia University, New York City, July 2004.

[Kakas *et al.*, 1998] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.

[Kim *et al.*, 2004] Kyoil Kim, Jacob A. Abraham, and Jayanta Bhadra. Model checking of security protocols with pre-configuration. In Kijoon Chae and Moti Yung, editors, *Information Security Applications, 4th International Workshop, WISA 2003, Jeju Island, Korea, August 25-27, 2003, Revised Papers*, volume 2908 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.

[Kunen, 1987] K. Kunen. Negation in logic programming. In *Journal of Logic Programming*, volume 4, pages 289–308, 1987.

[Lamport, 1983] L. Lamport. What Good Is Temporal Logic? In R. E. A. Mason, editor, *Information Processing*, volume 83, pages 657–668. Elsevier Science Publishers, 1983.

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2nd extended edition, 1987.

[Lomuscio and Sergot, 2002] Alessio Lomuscio and Marek J. Sergot. The bit transmission problem revisited. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part II*, pages 946–947, Bologna, Italy, July 15–19 2002. ACM Press.

[Lowe, 1996] G. Lowe. Breaking and fixing the needham-shroeder public-key protocol using csp and fdr. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems: Second International Workshop, TACAS'96*, volume 1055 of *Lecture Notes in Artificial Intelligence*, pages 147–166. Springer-Verlag, 1996.

[Merz, 2001] S. Merz. Model checking: A tutorial overview. In F. Cassez, C. Jard, B. Rozoy, and M.D.Ryan, editors, *Modeling and Verification of Parallel Processes*, number 2067 in Lecture Notes in Computer Science, pages 3–38. Springer-Verlag, 2001.

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

[Nipkow *et al.*, 2002] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[Pang, 2002] Jun Pang. Analysis of a security protocol in μCRL. In Chris George and Huaikou Miao, editors, *Formal Methods and Software Engineering, 4th International Conference on Formal Engineering Methods, ICFEM 2002 Shanghai, China, October 21-25, 2002, Proceedings*, volume 2495 of *Lecture Notes in Computer Science*, pages 396–400. Springer, 2002.

[Raimondi and Lomuscio, 2004] Franco Raimondi and Alessio Lomuscio. Verification of multiagent systems via ordered binary decision diagrams: An algorithm and its implementation. In N. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2004)*, pages 630–637, Columbia University, New York City, July 2004.

[SICStus, 2003] SICStus prolog user manual, release 3.11.0, October 2003. `http://www.sics.se/isl/sicstus/`.

[SOCS-web,] Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. IST-2001-32530. Home Page: `http://lia.deis.unibo.it/Research/SOCS/`.

[von Oheimb and Lotz, 2002] David von Oheimb and Volkmar Lotz. Formal security analysis with interacting state machines. In Dieter Gollmann, Günter Karjoth, and Michael Waidner, editors, *Computer Security - ESORICS 2002, 7th European Symposium on Research in Computer Security, Zurich, Switzerland, October 14-16, 2002, Proceedings*, volume 2502 of *Lecture Notes in Computer Science*, pages 212–229. Springer, 2002.

[Xanthakos, 2003] I. Xanthakos. *Semantic Integration of Information by Abduction*. PhD thesis, Imperial College London, 2003.