Specification and Verification of Agent Interaction Using Abductive Reasoning

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

[1] DEIS - Dipartimento di Elettronica, Informatica e Sistemistica,
Facoltà di Ingegneria, Università di Bologna,
viale Risorgimento, 2, 40136 – Bologna, Italy
{fchesani, pmello, ptorroni}@deis.unibo.it
[2] DI - Dipartimento di Ingegneria,
Facoltà di Ingegneria, Università di Ferrara,
Via Saragat, 1, 44100 – Ferrara, Italy
{marco.gavanelli, marco.alberti, lme}@unife.it

Abstract. Amongst several fundamental aspects in multi-agent systems design, the definition of the agent interaction space is of the utmost importance. The specification of the agent interaction has several facets: syntax, semantics, and compliance verification.

In an open society, heterogenous agents can participate without showing any credentials. Accessing their internals or their knowledge bases is typically impossible, thus it is impossible to prove a priori that agents will indeed behave according to the society rules.

Within the SOCS (Societies Of ComputeeS) project, a language based on abductive semantics has been proposed as a mean to define interactions in open societies. The proposed language allows the designer to define open, extensible and not over-constrained protocols. Beside the definition language, a software tool has been developed with the purpose of verifying at execution time if the agents behave correctly with respect to the defined protocols.

This paper provides a tutorial overview of the theory and of the tools the SOCS project provided to design, define and test agent interaction protocols.

1 Introduction

Multi-Agent Systems (MAS) are recently emerging as a new programming paradigm. In the process of designing and developing a MAS, various facets of the system have to be studied and addressed: the architecture of the various agents, the interactions amongst the agents, the social organisation, the rules, the roles of the agents in the society.

According to Davidsson [27], there can be four types of societies:

Closed societies are predefined societies, in which no agent can enter. Only the designer of the society can create new agents in the society itself.

Semi-closed are societies in which agents cannot enter, but they can nominate or spawn representatives in the society.

Semi-open are societies in which there exists one agent taking the role of gatekeeper, which receives the requests for entering the society. A potential member applies at the gate, can provide some credentials, and can possibly be admitted in the society by the gatekeeper.

Open are societies in which any agent can enter without restriction.

The classification by Davidsson is based on rules for entering the society, as this is the most pressing issue. Leaving the society could be done with a leaving protocol (in semi-open or semi-closed societies), or, in some cases, it can be considered as a way to punish misbehaving agents: when an agent does not comply to the rules, it is ejected from the society. Note that there are no given protocols to abandon an open society: agents may leave at any time without restrictions.

Clearly, open societies are the most flexible, but can also be very unstable. The set of members is not fixed, nor even computable in general, as new agents may join anytime, and current members could leave without any notification. Also openness à la Davidsson implies heterogeneity: any agent may join, so they are not required to share concepts such as beliefs, intentions, knowledge bases, or architectures. Some agents may exhibit powerful reasoning capabilities, while others may only be able to react to stimuli with predefined patterns. Foreign agents can join the society without restrictions and profit from interacting with the agents in the society. On the other hand, malicious agents could enter and disrupt the harmonious evolution of the society, threatening the usability of the whole MAS.

Thus, mastering open societies in order to drive them to a coherent, useful global behaviour is a challenge. The SOCS project accepted this challenge and provided theory, methods, and tools to raise from anarchy without overrestricting the agents' freedom. The goal is to point out unwanted behaviour without accessing the agent's mind. The aim is to orchestrate the agents' actions toward the user's goals without obliging agents to follow predefined rails.

A basic requirement for a meaningful society is that there exists a language of commonly understood utterances in the society. It is not necessary that all the agents understand the whole language: agents may understand subsets of the language, depending on the roles they want to play in the society, and on the type of interactions they want to start. The meaningful sequences of utterances make up the *interaction protocol*: agents are supposed to follow such protocols in order to get a coherent societal evolution. The MAS designer defines such protocols in a given language. Coherently with the concept of open society, protocols should be defined not to be over-restrictive, but should only guide the agents towards a desired behaviour. Note that agents cannot be forced to follow such protocols. While in non-open societies there are proposals that inspect the agent's mind and possibly update it to obtain a desired behaviour [32], in an open society any agent could join. The agent's implementation remains unrevealed to the society, so to change its mind and impose a desired behaviour is unimaginable. Agents, as well as humans, might not follow the protocol: this is a fact of life. It might happen due to malicious behaviour, because of erroneous design, because of ignorance of the society rules, or because of incapability to keep pace with tight deadlines. But, although unavoidable, protocol violation must not be accepted supinely, or the system will soon degenerate to chaos.

Of utmost importance is then to check that agents do not violate the protocols. Such a test cannot be executed in advance in an open society: even if we knew all the participants, we would still be unable to foresee the behaviour of members without knowing their implementation and their current (mental) state. Knowing the internals of the agents is against the concept of open society and, indeed, against that of multi-agent system research itself. The applicable check of compliance can be performed on-line: the society does not check beforehand the implementation of the agents, nor their internal mental states, but can only observe their external behaviour.

The SOCS project is aimed at developing Multi-Agent Systems for open societies and addresses two basic issues: it developed a model of a single agent [25], and a model for the society [10].

In this paper, we give taste of the society model, developed in the three-year SOCS project. In SOCS, the society model can be defined through a logic language, evolution of the IFF [34], called $\mathcal{S}$CIFF (Social Constrained IFF). The $\mathcal{S}$CIFF language can be used to define declaratively the interaction space, i.e., both the utterances of the agents and the protocols in the devised society, in a uniform way.

A corresponding proof-procedure can be used to verify that the agents behave according to the protocols, and detect possible violations. The $\mathcal{S}$CIFF proof-procedure is sound and complete with respect to its declarative semantics. Finally, practical issues have been taken into account, leading to an implementation and the development of a full-fledged software tool. The tool, called SOCS-SI, runs the implementation of the $\mathcal{S}$CIFF proof-procedure and it has been developed and interfaced with popular MAS systems. An intuitive Graphical User Interface (GUI) lets the user inspect both the history of happened events and the internal state of the proof-procedure.

This tutorial will not go deeply in the theoretical issues concerning the $\mathcal{S}$CIFF proof-procedure, but it will provide examples to clarify the concepts, together with pointers to previous publications, reports, and downloadable software to let the reader investigate the various facets of the SOCS society model and experiment with the provided tools.

The rest of the paper is organised as follows. After the introduction of the necessary background, we define the $\mathcal{S}$CIFF language, with motivating examples to smoothly learn how to define interaction space and protocols with $\mathcal{S}$CIFF. We then define the declarative semantics in Section 4, and the $\mathcal{S}$CIFF proof-procedure, with the SOCS-SI tool in Section 5. Discussion, related work and conclusions follow.

2 Background

We assume the reader has a basic familiarity with logics and logic programming; a good introduction is the book by Lloyd [46]. As it will be clear soon, the $\mathcal{S}$CIFF proof-procedure is based on Abductive Logic Programming and on Constraint Logic Programming; we introduce the two concepts in an intuitive way, and provide pointers to the formal parts.

2.1 Abduction

Abduction is a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge, that is handled by labelling some pieces of information as "abducibles". Abducibles can be viewed as possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base. The abduction process is typically applied when looking for an explanation about some observation. Starting from some observed facts, possible causes are hypothesised (they are abduced). Then it is possible to confirm the hypotheses by performing some additional observation: for example, the scientist postulates some theory, and then develops new experiments to confirm (or disconfirm) such theory. Another common application of abduction is *diagnosis*: the physician, by observing the symptoms, formulates some alternative hypothesis about the disease. The physician tries to find more facts by prescribing a patient another test, that will possibly support a smaller set of explanations. Some of the previously made hypotheses could be discarded because they are now incompatible with the new facts, or because some pairs of explanations cannot be assumed at the same time.

Formally, an *abductive logic program* (ALP) [40] is a triple $\langle P, Ab, IC\rangle$ where:

- P is a (normal) logic program, i.e., a set of clauses of the form $A_0 \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$, where $m, n \geq 0$, each A_i $(i = 1, \ldots, m+n)$ is an atom, and all variables are implicitly universally quantified with scope the clause. A_0 is called the *head* and $A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$ is called the *body* of any such clause;
- Ab is a set of *abducible predicates*, p, such that p is a predicate in the language of P which does not occur in the head of any clause of P;
- IC is a set of integrity constraints, that is, a set of formulae in the language of P.

Given an abductive logic program $\langle P, Ab, IC\rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (the *abductive explanation*), with $\Delta \subseteq Ab$, and which, together with P, entails G, and satisfies IC:

$$P \cup \Delta \models G \tag{1}$$

$$P \cup \Delta \models IC \tag{2}$$

The notion of entailment $\models$ depends on the semantics associated with the logic program P.

Several abductive proof procedures can be found in the literature (like the Kakas-Mancarella [41], limited to ground literals, SLDNFA [28], that can abduce literals with existentially quantified variables, ACLP [42] and $\mathcal{A}$-system [43], that integrate constraints, to cite some). The $\mathcal{S}$CIFF proof procedure, upon which the *SOCS-SI* application relies (see Section 5) is an extension of the if-and-only-if (IFF) abuctive proof procedure [34]. The integrity constraints, in the IFF proof-procedure, are expressed as a set of implications of the form:

$$B_1 \wedge \ldots \wedge B_n \rightarrow A_1 \vee \ldots \vee A_m$$

where all variables are universally quantified, A_i and B_i are atoms (can be abducibles or defined predicates), but they cannot be the negation of an atom.

2.2 Constraint Logic Programming

Constraint Logic Programming [37, 38] (CLP) is a class of programming languages that extend logic programming by giving an interpretation to some of the symbols. In classical Logic Programming, the symbols are not interpreted, so the term 2+3 does not mean 5, but simply a structure whose functor is + and whose terms are 2 and 3. Unification performs a syntactical operation, and does not provide any interpretation, so the term 5 will not unify with the term 3+2, and the goal 5=3+2 simply fails.

In Constraint Logic Programming, a subset of the terms and atoms are given a standard interpretation: the symbol 5 stands for the number *five* and the symbol + represents the addition operation. Unification is extended, and treated as a *constraint*. For example, the goal $5 = A + 3$ succeeds in CLP, providing the answer $A = 2$. This behaviour is obtained by identifying syntactically the set of interpreted atoms, called *constraints*, and inserting them into a *constraint store* instead of applying resolution. The constraints in the store are then evaluated by a *constraint solver*, that detects possible failures and infers new constraints.

Each language of the CLP class is identified by a *domain*, representing the set of values that a variable subject to constraints can assume, the set of constraints, the set of interpreted symbols. For example, CLP(R) [39] is the instance of CLP that works on the reals; this means that a variable in CLP(R) can have a real value, and it can be subject to constraints on the reals. Current implementations typically employ the simplex algorithm as constraint solver.

CLP(FD) is the specialisation of CLP on the Finite Domains [30]. Variables are initially assigned a domain through the predicate $Variable :: Domain$. For instance $X :: [red, green, blue]$ states that X can take only the values *red*, *green* or *blue*. On numeric values, CLP(FD) languages typically interpret the symbols $<$, $\leq$, $=$, $\neq$, etc., plus the usual operations $+$, $-$, $*$, $/$. In CLP(FD), imposing constraints typically deletes inconsistent values from the domains of the variables; for example, if $A :: [0..10]$, $B :: [1..5]$, $A < B$ would remove the values that cannot satisfy the imposed constraint, in this case the values greater than 4 from the domain of A. When a domain becomes empty, there cannot be an assignment for the corresponding variable, so the system fails. Various languages and efficient solvers have been developed [30, 53]. Such languages have been successfully used for hard combinatorial problems, such as scheduling [24], planning [22], bioinformatics [47], and many others. These solvers typically deal only with problems that contain existentially quantified variables.

3 The $\mathcal{S}$CIFF Language

We will now give the syntax of the $\mathcal{S}$CIFF language, together with examples to clarify the various components. We first introduce the concept of happened event, that is the basic link between the society and the agents. Then, we introduce the concept of expectation, that is used to describe the correct evolution of the society. We define the Social Knowledge Base (Section 3.1) and the Social Integrity Constraints (Section 3.2), that are used to relate happened events and expected behaviour, and in particular can be used to define the interaction protocols that are valid in the society.

We will use, as a running example, an auction scenario; we can envisage the following utterances:

openauction(Item, Type) opens an auction for an $Item$, specifying the $Type$ of auction, possibly with its own specific parameters;
bid(Item, Price) propose to buy the $Item$ for the proposed $Price$;
answer(win/lose, Item, Price) communicate if a bid wins or loses the $Item$ for the price $Price$;
deliver(Item) provide the $Item$;
pay(Item, Price) pay the $Price$ for the $Item$;

The language for defining the society is based on computational logics, and is used to:

– Describe the events generated by agents in the society. Happened events are represented with the atom $\mathbf{H}(Description, Time)$, where $Description$ is a term describing the type of event, its parameters, etc., and $Time$ is an integer identifying the instant in which the event happened in the society. The collection **HAP** of all events happened in the society is called the *history*.
– Define the expected behaviour of agents.
– Relate the current history with the expected behaviour.

The expected behaviour is a conjunction of literals $[\neg]\mathbf{E}(Description, Time)$ and $[\neg]\mathbf{EN}(Description, Time)$.

– $\mathbf{E}(Description, Time)$ declares that an event matching with $Description$ is expected to happen in the given $Time$. Note that $Time$ could be a variable, possibly subject to CLP constraints, which may restrict the instants in which the event is expected to happen. This can be useful to express deadlines, time intervals, scheduling constraints, and any type of constraints existing in the adopted CLP language (possibly, user-defined). For instance:

$$\mathbf{E}(tell(luke, mark, answer(A, pen, 1), auction_1), T), T < 10$$

could mean that agent Luke is supposed to tell Mark an answer regarding its bid of 1€ for a pen, within time 10, in the context of $auction_1$. *Description* can be a term, possibly with variables, which can be possibly constrained. We often use the term $tell(Sender, Receiver, Content, DialogueId)$ to indicate communicative acts, however the formalism is open to any type of term. All the variables occurring in a literal **E** are existentially quantified: as soon as an action matching the expectation is performed, the expectation is fulfilled.
– $\mathbf{EN}(Description, Time)$ states that all matching events are violating the protocol: they are expected not to happen in order to fulfill the correct social evolution. Again, *Time* can be a (possibly constrained) variable and *Description* a term involving variables. Variables in **EN** are universally quantified (unless they also occur in **E** literals), expressing that all the matching events are forbidden in a compliant interaction. If a variable is shared between **E** and **EN**, it will be quantified existentially, as in

$$\begin{aligned}(\exists_{Auctioneer,Bidder,T_1}\forall_{T_2})\ & \mathbf{E}(tell(Auctioneer, Bidder, win, D), T_1),\\ & \mathbf{EN}(tell(Auctioneer, Bidder, lose, D), T_2), T_2 > T_1\end{aligned}$$

meaning that any auctioneer should tell any bidder that he wins the auction, and afterwards the same auctioneer should not tell the same bidder that he loses in the context of the same dialogue.

The current history and the set of current expectations are related through the rules of the society, that can be defined in the $\mathcal{S}$CIFF language. Such language consists of a *Social Knowledge Base* and a set of *Social Integrity Constraints*, defined in the following sections.

3.1 The Social Knowledge Base

The Social Knowledge Base represents the pre-built, compile-time knowledge of the society. It is a set of rules that provide causal consequences of agents' behaviour. It provides properties that hold in the society when given conditions are met. For reasons that will be clear soon, the conditions are described by means of *expectations*, i.e., atoms describing the expected behaviour of the whole MAS.

We first give some motivating examples, then give the formal meaning and the scope rules. We can say that we have full occupation of the agents if none of them is idle, in any time:

$$full\_occupation :- \mathbf{EN}(idle(Agent), T). \tag{3}$$

meaning that

$$full\_occupation \leftarrow [\forall_{Agent,T}\ \mathbf{EN}(idle(Agent), T)].$$

We can say that an agent is busy if it is never idle:

$$busy(Agent) :- \mathbf{EN}(idle(Agent), T).$$

i.e.,

$$\forall_{Agent}\ busy(Agent) \leftarrow [\forall_T\ \mathbf{EN}(idle(Agent), T)].$$

An agent in a society could be fairly served if it gets at least one resource within some given time limit:

$$\begin{array}{l} fairly(Agent) :- \mathbf{E}(get(Agent, R), T), resource(R), T \leq 10. \\ resource(printer). \\ resource(window). \\ \ldots \end{array} \tag{4}$$

where the first clause means:

$$\forall_{Agent}\ fairly(Agent) \leftarrow [\exists_{R,T} \mathbf{E}(get(Agent, R), T), resource(R), T \leq 10],$$

or, equivalently,

$$\forall_{Agent,T,R}\ fairly(Agent) \leftarrow \mathbf{E}(get(Agent, R), T), resource(R), T \leq 10.$$

Formally, the Social Knowledge base is a set of clauses (i.e., implications in the form $Head \leftarrow Body$) that can contain, in the body, expectations, literals or constraints. Variables are all quantified universally with the following scope rules. Variables that occur only in **EN** literals and constraints are quantified universally with the body as scope (this is coherent with the intuitive meaning of Eq. 3: in order to have full occupation, there should be no agent which is idle in any time). All other variables are quantified universally with the clause as scope (as in Eq. 4, in which one resource R is enough).

Note that the given clauses can also be interpreted in an abductive fashion to derive the expected behaviour given that we want a fair society. Stated otherwise, there could be a goal of the society (fairness, in this example), and expectations could be abduced describing the behaviour of the agents in a fair society. Then expectations could be communicated to the agents in order to guide them towards the desired behaviour. The generated expectations can then be matched on-line with the history to check if the current evolution of the society indeed provides the requested feature.

3.2 The Social Integrity Constraints

Social Integrity Constraints are a set of implications that relate the current history with the expected behaviour. They can involve the various elements in the $\mathcal{S}$CIFF language, namely happened events, expectations, CLP constraints and predicates defined in the Social KB. Their syntax is given by the following grammar (where Literal and Term have the usual meaning as in Logic Programming [46] and Constraint is an atom in the language of constraints [37]):

$$
\begin{aligned}
\mathcal{IC}_S &::= [ic_S]^\star \\
ic_S &::= Body \rightarrow Head \\
Body &::= (EventLiteral \mid ExpLiteral)\ [\ \wedge\ BodyLiteral\ ]^\star \\
BodyLiteral &::= EventLiteral \mid ExpLiteral \mid Literal \mid Constraint \\
Head &::= HeadDisjunct\ [\ \vee HeadDisjunct\ ]^\star \mid \mathit{false} \\
HeadDisjunct &::= ExpLiteral\ [\ \wedge (ExpLiteral \mid Constraint)]^\star \\
EventLiteral &::= [\neg]\mathbf{H}(term, T) \\
ExpLiteral &::= [\neg]\mathbf{E}(term, T) \mid [\neg]\mathbf{EN}(term, T)
\end{aligned}
$$

Social Integrity Constraints are the perfect tool to define both the semantics of the basic utterances and the interaction protocol in a uniform way.

Semantics of Communication Acts. When designing the interaction, we have to define:

- the set of communication acts commonly understood in the society
- the meaning of such communication acts.

Various works propose a semantics for communication acts. One of the most popular is the FIPA [33] proposal, based on the BDI (Beliefs, Desires, Intentions) model [48]. The semantics of the so-called *speech acts* is based on the Beliefs, Desires and Intentions of the agents. For instance, if agent A *informs* agent B about X, this means that A wants B to believe X. Intuitively, A is also implicitly stating that it believes X. Formally, speech acts are modeled in terms of *feasibility conditions* and *rational effects*, expressed through BDI logic formulas [58].

In open societies, as argued earlier, one cannot access mental states of the agents, so checking that an utterance is compliant with its semantics is impossible from the society viewpoint. We prefer a semantics based on observable events in the environment, and, in particular, which actions the agents perform. Hence, instead of mentalistic approaches, we prefer social approaches. One of the most successful is the semantics based on commitments [57, 29]; intuitively, by performing a communicative act, an agent implicitly commits to the truth of some statement, or to perform some further action. In the $\mathcal{S}$CIFF language, commitments are easily represented through expectations. In the auction example, with *openauction* an agent commits to renounce owning an *Item* in exchange for money. In the $\mathcal{S}$CIFF language, this means that when the auctioneer opens an auction, it knows that it will be *expected* to deliver the item, in case there is some bid which is declared as winning:

$$
\begin{aligned}
&\mathbf{H}(tell(A, \_, openauction(Item, \_), D), T_{open}) \wedge \\
&\mathbf{H}(tell(B, A, bid(Item, Price), D), T_{bid}) \wedge \\
&\mathbf{H}(tell(A, B, answer(win, Item, Price), D), T_{win}) \\
\rightarrow\ &\mathbf{E}(tell(A, B, deliver(Item), D), T_{deliv}) \wedge \\
&T_{deliv} < T_{win} + T_{deliver\_deadline}
\end{aligned} \tag{5}
$$

We use the underscore for an unnamed variable (à la Prolog). Note that in an open society bidders may join the auction without invitation, so it is not important that the bidder was also addressee of the *openauction* message. The winning bidder might have obtained the information about the auction by another agent, from a blackboard, or advertisement.

Analogously, the bidder commits to pay in exchange for the item by declaring its *bid*:

$$\begin{aligned}
&\mathbf{H}(tell(B, A, bid(Item, Price), D), T_{bid}) \wedge \\
&\mathbf{H}(tell(A, B, answer(win, Item, Price), D), T_{win}) \wedge \\
&\mathbf{H}(tell(A, B, deliver(Item), D), T_{deliv}) \\
\rightarrow\ &\mathbf{E}(tell(B, A, pay(Item, Price), D), T_{pay}) \wedge \\
&T_{pay} < T_{deliv} + T_{pay\_deadline}
\end{aligned} \tag{6}$$

Note that these definitions are independent of the type of auction, which is defined by the protocol. The concept of expectation is not limited to represent the semantics of communicative acts, and, in particular, is not limited to express commitments, as we will see in the following.

Definition of the Protocol. Many works in the literature represent interaction protocols with Finite State Automata (FSA) [21]. The sequence of correct interaction moves can be interpreted as a phrase in the language recognised by the FSA. Clearly, FSA can recognise only regular languages, so there is a limit in the expressivity of the language for defining the protocol. On the other hand, the simple representation allows for powerful reasoning: proving properties of a protocol described as a FSA is probably easier than using a more sophisticated language. Model checking techniques, for example, have been used for this purpose by analysing protocols described as a FSA. Especially in the field of security protocol analysis, model checking-based techniques have been shown to be extremely successful [23].

Also, representing protocols as a graph means that every interaction which is not explicitly represented in the graph is considered *non compliant*. We believe that in open societies agents should be as free as possible: free to discuss in small groups with a language that is not recognised by the society, free to take shortcuts in long interaction runs (especially in presence of tight deadlines). A "whitelist" of allowed interaction moves is probably the best solution in the instance of security protocols; but in general it might be too restrictive.

The $\mathcal{S}$CIFF language gives the user more expressivity in the definition of the protocol: while in FSA an action can be only required or forbidden, in $\mathcal{S}$CIFF some actions are required (**E**), some are forbidden (**EN**) and all the others are *possible*. The *possible* state of an action provides the agent the freedom degrees to take shortcuts and to do actions not explicitly considered by the protocol designer. Uniformly to the semantics of communicative acts, the protocol can again be defined by means of Social Integrity Constraints.

Various protocols have a common core, which specialise into subtypes in different situations, with different properties. For example, the common concept of an auction can be implemented in a variety of ways, and in the real world various flavours of auctions are successfully employed (English, Dutch [54], first-price sealed bid, Vickrey, reverse, combinatorial [50, 35], just to name a few). On the other hand, all auction protocols share some core elements. From an engineering viewpoint, one could first try to define the common core, then refine the general protocol to obtain the desired specific features.

In the auction scenario, we can write rules that hold for all types of auctions, such as:

Before placing bids, there must have been an OpenAuction

$$\begin{aligned} & \mathbf{H}(tell(B, A, bid(Item, Price), D), T_{bid}) \\ \rightarrow\ & \mathbf{E}(tell(A, \_, openauction(Item, \_), D), T_{open}) \\ & \wedge\ T_{open} < T_{bid} \end{aligned} \tag{7}$$

The auctioneer should reply to all bids

$$\begin{aligned} & \mathbf{H}(tell(A, \_, openauction(Item, \_), D), T_{open})\ \wedge \\ & \mathbf{H}(tell(B, A, bid(Item, Price), D), T_{bid}) \\ \rightarrow\ & \mathbf{E}(tell(A, B, answer(Answer, Item, Price), D), T_{answer})\ \wedge \\ & Answer :: [win, lose] \end{aligned} \tag{8}$$

The auctioneer should not give contradicting answers

$$\begin{aligned} & \mathbf{H}(tell(A, B, answer(Answer_1, Item, Price), D), T_1) \\ \rightarrow\ & \mathbf{EN}(tell(A, B, answer(Answer_2, Item, Price), D), T_2)\ \wedge \\ & Answer_1 \neq Answer_2 \end{aligned} \tag{9}$$

Other rules specify the type of auction. One of the most used is the *English auction.* In an English auction, bids are increasing in value: a first bidder declares publicly its bid, then other bids can be placed, of increasing value. When no more bids are placed, the good is assigned to the last bid (which is also the highest). In order to decide that no other bids will occur, there exists a timeout τ: in human auctions, after each bid the auctioneer counts typically up to three and then declares the item sold.

The previous core of auction protocols can be easily specialised to the English auction instance by adding more Social Integrity Constraints, which refine the general auction protocol schema. In an English auction bids are in increasing order, so bidders should not place a bid which is lower than the previous ones:

$$\begin{aligned} & \mathbf{H}(tell(A, \_, openauction(Item, english(\tau)), D), T_{open}) \wedge \\ & \mathbf{H}(tell(Bidder_1, A, bid(Item, Price_1), D), T_1) \\ \rightarrow\ & \mathbf{EN}(tell(Bidder_2, A, bid(Item, Price_2), D), T_2)\ \wedge \\ & T_2 > T_1 \wedge Price_2 \leq Price_1 \end{aligned} \tag{10}$$

After a bid has been placed, the auctioneer waits for τ time units; either a better bid is placed within this time, or the auctioneer should declare the last bid as winning:

$$\begin{aligned} & \mathbf{H}(tell(A, \_, openauction(Item, english(\tau)), D), T_{open})\ \wedge \\ & \mathbf{H}(tell(B_1, A, bid(Item, Price_1), D), T_1) \\ \rightarrow\ & \mathbf{E}(tell(B_2, A, bid(Item, Price_2), D), T_2)\ \wedge \\ & Price_2 > Price_1\ \wedge\ T_2 < T_1 + \tau \\ \vee\ & \mathbf{E}(tell(A, B_1, answer(win, Item, Price_1), D), T_{win})\ \wedge \\ & T_{win} = T_1 + \tau \end{aligned} \tag{11}$$

It is well known that the English auction protocol might not terminate, so if there is a deadline, other auction protocols are used. The Dutch auction is used when the goods must be sold quickly. The Dutch auction follows a protocol which is nearly opposite to the English: the proposals are from the auctioneer, and they decrease in time. The auctioneer starts proposing a (very high) price. If one bidder accepts it, it wins the auction. If, within τ time units, no bidder replies, the auctioneer proposes a lower price.

In this case, we can exchange the order of the primitives *answer* and *bid*. The two utterances retain their meaning: $answer(win, Item, Price)$ still means that a bid for the *Item* and with the given *Price* wins, while $bid(Item, Price)$ means that the bidder would pay the *Price* for the *Item*. While retaining the original meaning, we can change the protocol: first the auctioneer declares a possible winning price, then the bidders place their bids.

Again, we refine the generic auction given by the semantics of the communication acts (5-6) together with the auction core (7-9) adding more Social Integrity Constraints specific for the Dutch auction.

In the Dutch auction, we must ensure that only one (valid) bid is placed; after the first bid is placed all other bids are illegal:

$$\begin{array}{l} \mathbf{H}(tell(A,\_,openauction(Item,dutch(\tau)),D),T_{open}) \wedge \\ \mathbf{H}(tell(A,\_,answer(win,Item,Price),D),T_a) \wedge \\ \mathbf{H}(tell(B_1,A,bid(Item,Price),D),T_1) \\ \rightarrow \mathbf{EN}(tell(\_,\_,bid(Item,\_),D),T_2) \ \wedge T_2 > T_1 \end{array} \tag{12}$$

Moreover, either a bid has been placed within τ time units, or the auctioneer should propose a new (lower) price:

$$\begin{array}{l} \mathbf{H}(tell(A,\_,openauction(Item,dutch(\tau)),D),T_{open}) \wedge \\ \mathbf{H}(tell(A,\_,answer(win,Item,Price_i),D),T_i) \\ \rightarrow \mathbf{E}(tell(B,A,bid(Item,Price),D),T_{bid}) \ \wedge T_{bid} < T_i + \tau \\ \vee \ \mathbf{E}(tell(A,\_,answer(win,Item,Price_{i+1}),D),T_{i+1}) \wedge \\ T_{i+1} = T_i + \tau \ \wedge \ Price_{i+1} < Price_i \end{array} \tag{13}$$

Note that in this way the protocol is not overconstrained by a fixed sequence of communicative acts. Many freedom degrees are left to the agents, that may exploit them to converge faster to an agreement. For example, the auctioneer may start with a high price, bidders place their bids even if they do not match the price proposed by the auctioneer, and the auctioneer could choose one of them. Infinitely many hybrid auctions flavours could arise in an interaction. Of course, if this is not the intended meaning, and avoiding this double negotiation is necessary, the designer can refine the specification by adding more Social Integrity Constraints to avoid unwanted paths: *in the time interval between two answers, bidders can bid only the proposed price*, i.e., they cannot bid other prices:

$$\begin{array}{l} \mathbf{H}(tell(A,\_,openauction(Item,dutch(\tau)),D),T_{open}) \wedge \\ \mathbf{H}(tell(A,\_,answer(win,Item,Price_i),D),T_i) \\ \rightarrow \mathbf{EN}(tell(B,A,bid(Item,Price),D),T_{bid}) \wedge \\ Price \neq Price_i \wedge T_i < T_{bid} < T_i + \tau \end{array} \tag{14}$$

In this way, the bidders can place only bids whose prices match the prices proposed by the auctioneer.

4 Declarative Semantics

The SOCS social model is interpreted in terms of an *Abductive Logic Program (ALP)*. The idea is to exploit abduction for defining expected behaviour of the agents inhabiting the society, and an abductive proof-procedure to dynamically *generate* the expectations and perform the *compliance check*.

Classical abduction does not contemplate changes in the knowledge bases, while in a society the set of happened events dynamically grows. For this reason, we give abductive semantics to a society by associating an ALP to each possible history. We call *society instance* the grounding of a society on a given history:

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

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

We give semantics to a society instance by defining those sets $\mathbf{EXP}$ (Δ in the abductive framework) 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. Equations 1 and 2 can be rewritten as:

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models G \tag{15}$$

$$SOKB \cup \mathbf{HAP} \cup \mathbf{EXP} \models \mathcal{IC}_S \tag{16}$$

Moreover, we require the set $\mathbf{EXP}$ to be also

$\neg$-consistent: for any p, $\mathbf{EXP}$ cannot include $\{\mathbf{E}(p), \neg\mathbf{E}(p)\}$ or $\{\mathbf{EN}(p), \neg\mathbf{EN}(p)\}$ (which implements explicit negation), and

E-consistent: for any p, $\mathbf{EXP}$ cannot include $\{\mathbf{E}(p), \mathbf{EN}(p)\}$ (an event cannot be both expected to happen and expected not to happen);

At this point it is possible to define the concepts of fulfillment and violation of a set $\mathbf{EXP}$ of social expectations. Fulfillment requires all the $\mathbf{E}$ expectations to have a matching happened event, and all $\mathbf{EN}$ expectations not to have a matching $\mathbf{H}$ event in the history:

Definition 2. *Given a society instance $\mathcal{S}_{\mathbf{HAP}}$, a set of social expectations* $\mathbf{EXP}$ *that is $\neg-$consistent and $\mathbf{E}-$consistent, is* fulfilled *if and only if for all (ground) terms p:*

$$\mathbf{HAP} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \rightarrow \neg\mathbf{H}(p)\} \not\models \mathit{false} \tag{17}$$

Symmetrically, we define violation as follows:

Definition 3. *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:*

$$\mathbf{HAP} \cup \mathbf{EXP} \cup \{\mathbf{E}(p) \rightarrow \mathbf{H}(p)\} \cup \{\mathbf{EN}(p) \rightarrow \neg\mathbf{H}(p)\} \models \mathit{false} \tag{18}$$

5 The *SOCS-SI* Tool for Compliance Checking

The *SOCS-SI* (SOCS-Society Infrastructure) application check the compliance of a given agent interaction with a given protocol definition. It uses the $\mathcal{S}$CIFF proof-procedure to perform the abductive reasoning, and it provides integration with multi-agent platforms. The $\mathcal{S}$CIFF is the logical "engine": by performing the abduction process, it generates the expectations (represented as abducibles) and verifies if they are fulfilled or violated.

While the *SOCS-SI* software heavily relies on the $\mathcal{S}$CIFF proof-procedure, this can be used instead as a stand-alone application. In fact, $\mathcal{S}$CIFF is a stand-alone abductive proof-procedure, that has been exploited for agent interaction compliance checking, but that can be used also to perform general abductive reasoning.

5.1 The $\mathcal{S}$CIFF Proof-Procedure

The operational semantics of the $\mathcal{S}$CIFF language is an abductive proof-procedure, i.e., it computes the set Δ introduced in Section 2.1. It is an extension of the IFF proof-procedure, but it also provides the following additional features:

- abduces atoms with variables universally quantified;
- deals with CLP constraints, also imposed as quantifier restrictions on universally quantified variables;
- is more dynamic, in fact new events may arrive, and the proof-procedure dynamically takes them into consideration in the knowledge base;
- has the new concepts, related to on-line verification, of *fulfillment* and *violation.*

As its ancestor IFF, the $\mathcal{S}$CIFF is a transition system that rewrites logic formulae into equivalent logic formulae. Each formula is a *Node* of the proof-tree, and it can be rewritten by one of the transitions into one or more nodes, logically in disjunction (so building an or-tree). The elements in a node are arranged as follows:

$$N \equiv \langle R, CS, PSIC, \mathbf{PEXP}, \mathbf{HAP}, \mathbf{FULF}, \mathbf{VIOL} \rangle \tag{19}$$

where R is the resolvent, CS is the constraint store (as in CLP), PSIC is a set of implications (initially the set of all integrity constraints), **HAP** is the current history, **PEXP**, **FULF**, and **VIOL** are, respectively, the set of pending, fulfilled, and violated expectations. Reporting the transitions of the $\mathcal{S}$CIFF proof-procedure is beyond the scope of this paper, but the interested reader can refer to previous publications for more details [13].

$\mathcal{S}$CIFF has been implemented in SICStus Prolog [53], exploiting its CHR library for defining the rewriting rules, and its CLP(FD) engine (suitably extended to deal with universally quantified variables) as underlying constraint solver.

5.2 The SOCS-SI Tool

SOCS-SI is a software tool that uses the $\mathcal{S}$CIFF proof procedure to check if an agent interaction is compliant with a given protocol definition. It is a full-fledged system, able to interface with multi-agent system like JADE [36] and PROSOCS [55], as well as the standard e-mail system (to verify interactions happening between human agents), and simple text files containing the log of the interaction. It provides a Graphic User Interface (GUI), that allows the user to observe the interaction in the form of the exchanged messages, to view the list of participants to the interaction, and to inspect the set of expectations generated by the proof-procedure: this set represents the expected behaviour at the society level.

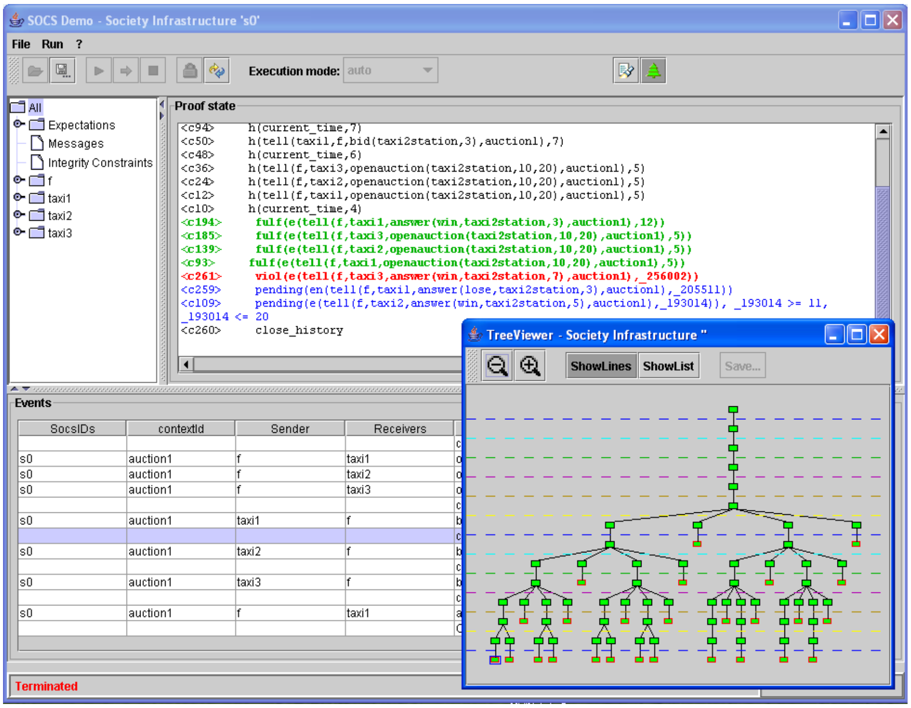
Through *SOCS-SI*, it is possible to access a tree-view of the computation of the $\mathcal{S}$CIFF proof-procedure (Figure 1); interestingly, the shown tree bears both an operational and a logical interpretation. The operational interpretation is an intuitive graphical form of a log-file, showing the most significant computational steps, useful for debugging purposes. The logical meaning is an or-tree (the branches of the tree are connected by logical disjunction) of the possible derivations timed by the incoming events. For each incoming event that enriches the knowledge base, the frontier of the explored proof-tree (which is a logical disjunction) is shown. The user can inspect each of the nodes, and see in the main window the state of the computation, i.e., the tuple given in Eq. 19.

SOCS-SI takes as input three types of information:

- The source of events, i.e. the multi agent system that is going to be observed.
- A file containing the Social Knowledge Base, as specified in Section 3.1.
- One or more files containing the specification of the protocol by means of Social Integrity Constraints (as discussed in Section 3.2).

SOCS-SI can be easily extended to support other multi-agent platforms, by simply adding interface modules, and selecting them as event sources. More details on *SOCS-SI*, on the output it generates and how to support new agent platforms can be found in [8].

SOCS-SI can be used at both design-time and run-time. The protocol designer can use *SOCS-SI* to support the development of a correct protocol: once this has been defined using the $\mathcal{S}$CIFF language, it is possible to check if the protocol does indeed allow only the desired interactions, and it excludes the wrong ones. Agent dialogues can be simulated by specifying on a log file the exchanged messages, and *SOCS-SI* can check the compliance of these interactions w.r.t. the protocol specification. Moreover, *SOCS-SI* provides a detailed view of the expectations



Proof state

```
<c94>     h(current_time,7)
<c50>     h(tell(taxi1,f,bid(taxi2station,3),auction1),7)
<c48>     h(current_time,6)
<c36>     h(tell(f,taxi3,openauction(taxi2station,10,20),auction1),5)
<c24>     h(tell(f,taxi2,openauction(taxi2station,10,20),auction1),5)
<c12>     h(tell(f,taxi1,openauction(taxi2station,10,20),auction1),5)
<c10>     h(current_time,4)
<c194>     fulf(e(tell(f,taxi1,answer(win,taxi2station,3),auction1),12))
<c185>     fulf(e(tell(f,taxi3,openauction(taxi2station,10,20),auction1),5))
<c139>     fulf(e(tell(f,taxi2,openauction(taxi2station,10,20),auction1),5))
<c93>     fulf(e(tell(f,taxi1,openauction(taxi2station,10,20),auction1),5))
<c261>     viol(e(tell(f,taxi3,answer(win,taxi2station,7),auction1),_256002))
<c259>     pending(en(tell(f,taxi1,answer(lose,taxi2station,3),auction1),_205511))
<c109>     pending(e(tell(f,taxi2,answer(win,taxi2station,5),auction1),_193014)), _193014 >= 11,
_193014 <= 20
<c260>     close_history
```

Events

SocsIDs	contextId	Sender	Receivers	
				c
s0	auction1	f	taxi1	o
s0	auction1	f	taxi2	o
s0	auction1	f	taxi3	o
				c
s0	auction1	taxi1	f	b
				c
s0	auction1	taxi2	f	b
				c
s0	auction1	taxi3	f	b
				c
s0	auction1	f	taxi1	a
				C

Terminated

Fig. 1. The Logic or-Tree

generated at every step of the interaction and, in case of violations, indicates also the set of the possible causes.

Thanks to its integration with various agent platforms, *SOCS-SI* can be used at runtime to detect violations to the society protocols: in such cases proper measures can be taken against the culprit (e.g. excluding the culprit agent from the society).

6 Discussion

A number of papers describe in detail various aspects of the $\mathcal{S}$CIFF proof-procedure; the details cannot be given here because of lack of space. In previous publications, the interested reader can find the definition of the general framework [15, 11, 10, 9], language and declarative semantics [14], the operational semantics [12, 13], and the implementation [8, 6]. The proofs of soundness, completeness and termination of the $\mathcal{S}$CIFF proof-procedure can be downloaded from the $\mathcal{S}$CIFF web page [51]. The concept of expectation, developed in the SOCS project, has been compared with that of obligation of deontic logic [17].

A plethora of different protocols has been tested with $\mathcal{S}$CIFF and *SOCS-SI*, including various flavours of auctions (English, First Price Sealed Bid, Combinatorial Auctions [5]), resource sharing [16], e-commerce protocols (NetBill [15]), high level protocols (FIPA) and low-level ones (TCP/IP). The proof-procedure and the *SOCS-SI* application have been tested thoroughly; the systems have been subject to stress testing, varying the number of interacting agents and the exchanged messages [3].

SOCS-SI and $\mathcal{S}$CIFF can be downloaded from the web [1, 51].

7 Related Work

Opposite to mentalistic approaches [58], that give semantics to communication through the mental states of the agents, social approaches propose to focus on observable acts [57, 29]. The works on $\mathcal{S}$CIFF and SOCS-SI take the second view, and indeed belongs to such research stream. While other works [57] are based on temporal logics, we adopted a constraint solver, that is able to efficiently deal with scheduling constraints and to express a variety of real-life concepts, such as deadlines.

The idea of expected behaviour can be considered related to *deontic logic* [59]; however, our claim is that we do not need the full power of the standard deontic logic, but only constraints on events that are expected to happen or not to happen. We do not use deontic operators, but instead we map them into predicates (**E** for positive and **EN** for negative expectations).

Our work is very close for the objective and methodology to the notable work on computational societies presented and developed in the context of the ALFEBIITE project [18], and the work by Singh [60] where a social semantics is exemplified by using a commitment-based approach. With this work we share the same view of an open society as that of [20].

Artikis et al. [20] present a theoretical framework for providing executable specifications of particular kinds of multi-agent systems, called open computational societies, and present a formal framework for specifying, animating and ultimately reasoning about and verifying the properties of open computational societies: systems where the behaviour of the members and their interactions cannot be predicted in advance. Differently from [20], we do not explicitly represent the institutional power of the members and the concept of valid action. Permitted are all social events that do not determine a violation, i.e., all events that are not explicitly forbidden are allowed, and this implements a sort of "open world assumption" at a society level. Permission, when it needs to be explicitly expressed, is mapped into the negation of a negative expectation: $\neg\mathbf{EN}(\ldots)$.

The semantics of our model can be directly mapped in an abductive framework, where expectations can be confirmed (fulfilled) or disconfirmed (violated) by the history of the happened social events.

Sadri et al. [49] propose a framework for agent negotiation based on dialogue. The dialogue of agents is defined in a two-part setting as an ordered sequence of communication primitives. The generation of dialogues results from an abductive reasoning process taking place inside each agent during the *think* phase of its life cycle (the cycle being inspired by [44]). Our work shares the view of integrity constraints that provide new abducible atoms, but in our case the abducibles are *expectations* of the society about the future behavior of the agents, while in [49] they are used as communication primitives.

Many abductive proof procedures have been proposed in the past; the reader can refer to the exhaustive survey by Kakas *et al.* [40]. The $\mathcal{S}$CIFF proof-procedure is mostly related to the IFF [34], which it extends in several directions, as explained in the paper.

Other proof procedures deal with constraints; in particular ACLP [42] and the $\mathcal{A}$-system [43] deeply focus on efficiency issues. Both use integrity constraints in the form of denials, instead of forward rules, and both only abduce existentially quantified atoms, which makes the $\mathcal{S}$CIFF in this sense more expressive.

The integration of the IFF with constraints has been explored, both theoretically [45], and in an implementation [31]. These works, however, do not deal with confirmation of hypotheses and universally quantified variables in abducibles.

Abdual [19] is a system for performing abduction from extended logic programs plus constraints adopting the well-founded semantics, but also capturing 2-valued generalized stable models. It handles only ground negated literals, and it relies on tabled evaluation.

8 Conclusions and Future Work

In this paper, we presented a tutorial overview of the methods and tools the SOCS project provided for defining the interaction space in an agent society. The reader interested in the theory can find the foundations of the $\mathcal{S}$CIFF language and proof-procedure in the given references. The practitioner interested in applying the tools can download the implementation of the proof-procedure and apply it to the check of compliance of interaction protocols, or to general abductive tasks. The *SOCS-SI* tool can be easily adapted to interact with popular multi agent systems, or with human communication tools, such as the e-mail exchange.

Current work follows multiple threads. A first thread is aimed at applying the developed tools to new applications, beside the check of compliance to protocols. Experiments are currently conducted in planning with the abductive event calculus [52], a classical application of abductive proof-procedures. Other applications involve checking protocols in other environments besides agents, like giving medical guidelines [4].

A second thread focuses on the evolution and optimisation of the proof-procedure. The aim is to reduce the branching factor of the $\mathcal{S}$CIFF proof-procedure by identifying a priori branches that will fail and whose exploration can be skipped. This goal could be obtained through powerful constraint propagation, or by encapsulating knowledge given by the experienced user on the application domain.

The third thread widens the properties the $\mathcal{S}$CIFF proof-procedure is able to prove. Besides on-line protocol conformance, the $\mathcal{S}$CIFF proof-procedure could also prove properties a-priori, by considering as input only the protocol (and not the history). The software engineering task of developing new protocols could be assisted by a tool that proves properties of the protocol. Such an approach has been widely used for detecting flawedness of security protocols [23]. Our aim is to extend the $\mathcal{S}$CIFF proof-procedure to also prove protocol properties, given as negated goals. The proof-procedure could find counterexamples if the proposed property is not entailed by the protocol definition, similarly to model checking in security protocols. The first experiments are very encouraging, as the $\mathcal{S}$CIFF proof-procedure was able to find attacks of flawed security protocols [7], although we believe that $\mathcal{S}$CIFF is better suited to prove properties of other protocols, such as e-commerce ones.

Finally, extensions of the framework could be considered, like communicating the expectations to the agents, or advertising to possible members the rules that should be followed in the society. Such rules would implicitly provide the accepted common language understood in the society.

References

1. SOCS-SI. `http://lia.deis.unibo.it/Research/socs_si/`.
2. C. Priami and P. Quaglia, editors, *Global Computing: IST/FET International Workshop*, volume 3267 of *LNAI*. Springer-Verlag, 2005.
3. M. Alberti and F. Chesani. The computational behaviour of the SCIFF abductive proof procedure and the SOCS-SI system. *Intelligenza Artificiale*, II(3):45–51, 2005.
4. M. Alberti, F. Chesani, A. Ciampolini, P. Mello, M. Montali, S. Storari, and P. Torroni. Protocol specification and verification by using computational logic. In *In Proceedings of Workshop dagli Oggetti agli Agenti (WOA'05)*, November 2005.
5. M. Alberti, F. Chesani, M. Gavanelli, A. Guerri, E. Lamma, P. Mello, and P. Torroni. Expressing interaction in combinatorial auction through social integrity constraints. *Intelligenza Artificiale*, II(1):22–29, 2005.
6. M. Alberti, F. Chesani, M. Gavanelli, and E. Lamma. The CHR-based implementation of a system for generation and confirmation of hypotheses. In A. Wolf, T. Frühwirth, and M. Meister, editors, *19th Workshop on (Constraint) Logic Programming*, pages 111–122, University of Ulm, Germany, 2005.
7. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Security protocols verification in abductive logic programming: a case study. In O. Dikenelli, M.P. Gleizes, and A. Ricci, editors, *Proceedings of ESAW'05*, LNAI. Springer Verlag. to appear.
8. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based tool. In Trappl [56], pages 570–575. Extended version to appear in Applied Artificial Intelligence.
9. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A logic based approach to interaction design in open multi-agent systems. In *Proceedings of WETICE-2004*, pages 387–392. IEEE Press, June 14–16 2004.
10. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. The SOCS computational logic approach for the specification and verification of agent societies. In Priami and Quaglia [2], pages 324–339.
11. M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A social ACL semantics by deontic constraints. In V. Mařík, J. Müller, and M. Pěchouček, editors, *CEEMAS 2003*, volume 2691 of *LNAI*, pages 204–213. Springer-Verlag, 2003.

12. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Abduction with hypotheses confirmation. In F. Giunchiglia, editor, *IJCAI-05*, pages 1545–1546.
13. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. The $\mathcal{S}$CIFF abductive proof-procedure. In S. Bandini and S. Manzoni, editors, *AI*IA 2005*, volume 3673 of *LNAI*, pages 135–147. Springer Verlag.
14. 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*, volume 2829 of *LNAI*, pages 287–299. Springer-Verlag, 2003.
15. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. *Electronic Notes in Theoretical Computer Science*, 85(2), 2003.
16. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Modeling interactions using *Social Integrity Constraints*: A resource sharing case study. In J.A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Declarative Agent Languages and Technologies*, volume 2990 of *LNAI*, pages 243–262. Springer-Verlag, 2004.
17. M. Alberti, E. Lamma, M. Gavanelli, P. Mello, G. Sartor, and P. Torroni. Mapping deontic operators to abductive expectations. *Computational and Mathematical Organization Theory.* To appear.
18. ALFEBIITE: A Logical Framework for Ethical Behaviour between Infohabitants in the Information Trading Economy of the universal information ecosystem. IST-1999-10298, 1999. Home Page: `http://www.iis.ee.ic.ac.uk/~alfebiite/`.
19. J. Alferes, L. M. Pereira, and T. Swift. Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Theory and Practice of Logic Programming*, 4:383–428, July 2004.
20. A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In Castelfranchi and Lewis Johnson [26], pages 1053–1061.
21. M. Barbuceanu and M.S. Fox. Cool: A language for describing coordination in multi-agent systems. In V. Lesser, editor, *Proceedings of the First Intl. Conference on Multi-Agent Systems*, pages 17–25. AAAI Press/The MIT Press, 1995.
22. R. Barruffi, M. Milano, and R. Montanari. Planning for security management. *IEEE Intelligent Systems*, 16(1):74–80, 2001.
23. D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In E. Snekkenes and D. Gollmann, editors, *Computer Security - ESORICS 2003*, volume 2808 of *LNCS*, pages 253–270. Springer-Verlag, 2003.
24. F. Bosi and M. Milano. Enhancing CLP branch and bound techniques for scheduling problems. *Software Practice & Experience*, 31(1):17–42, 2001.
25. A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, F. Toni, and G. Terreni. The KGP model of agency: Computational model and prototype implementation. In Priami and Quaglia [2], pages 340–367.
26. C. Castelfranchi and W. Lewis Johnson, editors. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)*, Bologna, Italy, 2002. ACM Press.
27. P. Davidsson. Categories of artificial societies. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agents World II*, volume 2203 of *LNAI*, pages 1–9. Springer-Verlag, 2001.
28. M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.
29. V. Dignum, J. J. Meyer, and H. Weigand. Towards an organizational model for agent societies using contracts. In Castelfranchi and Lewis Johnson [26], pages 694–695.

30. M. Dincbas, P. van Hentenryck, H. Simonis, and A. Aggoun. The constraint logic programming language CHIP. In *Proceedings of the 2nd International Conference on 5th Generation Computer Systems*, pages 693–702, Tokyo, Japan, 1988.
31. U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. In J.J. Alferes and J.A. Leite, editors, *JELIA 2004*, volume 3229 of *LNAI*, pages 31–43. Springer-Verlag.
32. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In G. Gottlob and T. Walsh, editors, *IJCAI-03.* Morgan Kaufmann.
33. FIPA: Foundation for Intelligent Physical Agents. `http://www.fipa.org/`.
34. 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.
35. A. Guerri and M. Milano. Exploring CP-IP based techniques for the bid evaluation in combinatorial auctions. In F. Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003*, volume 2833 of *LNCS*, pages 863–867. Springer-Verlag.
36. Java Agent DEvelopment framework. `http://sharon.cselt.it/projects/jade/`.
37. J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.
38. J. Jaffar, M.J. Maher, K. Marriott, and P.J. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.
39. J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.
40. 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.
41. A.C. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of PRICAI-90*, pages 438–443. Ohmsha Ltd., 1990.
42. A.C. Kakas, A. Michael, and C. Mourlas. ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, 44(1-3):129–177, July 2000.
43. A.C. Kakas, B. van Nuffelen, and M. Denecker. $\mathcal{A}$-System: Problem solving through abduction. In B. Nebel, editor, *IJCAI-01*, pages 591–596, Seattle, Washington, USA, August 2001. Morgan Kaufmann.
44. R.A. Kowalski and F. Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25(3/4):391–419, 1999.
45. R.A. Kowalski, F. Toni, and G. Wetzel. Executing suspended logic programs. *Fundamenta Informaticae*, 34:203–224, 1998.
46. J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2nd extended edition, 1987.
47. A. Dal Palù, A. Dovier, and E. Pontelli. Heuristics, optimizations, and parallelism for protein structure prediction in CLP(FD). In P. Barahona and A.P. Felty, editors, *Proc. of Principles and Practice of Declarative Programming*, pages 230–241. ACM, 2005.
48. A. Rao and M. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of KR'92*, pages 439–449, 1992.
49. F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In S. Greco and N. Leone, editors, *Proceedings of JELIA'02*, volume 2424 of *LNCS*, pages 419–431. Springer-Verlag, September 2002.

50. T. Sandholm. Algorithm for optimal winner determination in combinatorial auction. *Artificial Intelligence*, 135(1-2):1–54, 2002.
51. The $\mathcal{S}$*CIFF* abductive proof procedure. `http://lia.deis.unibo.it/Research/sciff/`.
52. M. Shanahan. The event calculus explained. In M. Wooldridge and M.M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, volume 1600 of *LNCS*, pages 409–430. Springer, 1999.
53. SICStus prolog user manual, release 3.11.0, 2003. `http://www.sics.se/sicstus/`.
54. C. Sierra and P. Noriega. Agent-mediated interaction. From auctions to negotiation and argumentation. In M. d'Inverno, M. Luck, M. Fisher, and C. Preist, editors, *Foundations and Applications of Multi-Agent Systems*, volume 2403 of *LNCS*, pages 27–48. Springer-Verlag, 2002.
55. K. Stathis, A.C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In Trappl [56], pages 523–528. Extended version to appear in Applied Artificial Intelligence.
56. R. Trappl, editor. *Proceedings of the 17th European Meeting on Cybernetics and Systems Research, Symposium AT2AI-4*. Vienna, Austria, April 13-16 2004.
57. M. Venkatraman and M.P. Singh. Verifying compliance with commitment protocols. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.
58. M. Wooldridge. *Introduction to Multi-Agent Systems*. John Wiley & Sons, Ltd., 2002.
59. G.H. Wright. Deontic logic. *Mind*, 60:1–15, 1951.
60. P. Yolum and M.P. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In Castelfranchi and Lewis Johnson [26], pages 527–534.