Università degli Studi di Bologna
DEIS

Verifiable Agent Interaction in Abductive Logic Programming: the $\mathcal{S}$CIFF proof-procedure

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

March 2006

DEIS Technical Report no. DEIS-LIA-06-001 *LIA Series no. 75*

Verifiable Agent Interaction in Abductive Logic Programming: the $\mathcal{S}$CIFF proof-procedure

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
{marco . alberti, marco . gavanelli, evelina . lamma}@unife.it
[2]*DEIS, Università di Bologna*
V.le Risorgimento 2
40136 Bologna, Italy
{federico . chesani, paola . mello, paolo . torroni}@unibo.it

March 2006

Abstract. $\mathcal{S}$CIFF is a new abductive logic programming proof-procedure for reasoning with expectations in dynamic environments. $\mathcal{S}$CIFF is also the main component of a framework thought to specify and verify interaction in open agent societies. In this paper we present the declarative and operational semantics of $\mathcal{S}$CIFF, its termination, soundness and completeness results, and some sample applications to demonstrate its use in the multi-agent domain.

Keywords: *Abductive Logic Programming, Proof-procedures, Agent Interaction Protocols, Declarative Semantics, Formal Properties, $\mathcal{S}$CIFF, SOCS (SOcieties of ComputeeS), IFF Proof-procedure*

Contents

1 Introduction

This article describes a declarative and operational framework, grounded on Abductive Logic Programming (ALP) [63], that can be used in a number of situations involving reasoning with incomplete knowledge and dynamic occurrence of events.

The main motivations of such a work originate from the domain of agent interaction. In order to appreciate them, one must consider that in such a domain the definition of agent communication languages and protocols, their semantic characterization, and their verification, are key issues with many points still open, notwithstanding interaction being one of the characterizing and most widely studied aspects of agency [101]. In order to address these issues, not surprisingly, multifarious logics have been adopted, extended and developed: modal logics, mainly, in the domain of rational agents, and a number of fragments of temporal logics in the domain of open societies and social commitments.

Our work mainly focusses on the specification and verification of interaction in open agent societies. It advocates the use of Computational Logic for agent applications, and it demonstrates ALP to be a viable tool for the specification, development and implementation of operational frameworks for agents and their application in a number of reference scenarios. The outcomes of our research activity, carried out in the context of a EU IST project named SOCS (Societies Of ComputeeS) [92], completed in June 2005, are a declarative semantic characterization of open agent societies in terms of abduction, and an operational framework centered around an ALP proof-procedure called $\mathcal{S}$CIFF.

$\mathcal{S}$CIFF is an extension of the well-known IFF abuctive proof-procedure defined by Fung and Kowalski [56]. IFF has inspired conspicuous work in logic-based agent systems [68, 36, 82], paving the way for the definition of a reference agent architecture that combines rationality with reactivity and other components like planning, temporal reasoning and deliberation, to cite some [67, 72]. Important as it has been in its seminal role, some limitations have made the use of IFF impractical in many scenarios, and an agent system based on IFF and at the same time dealing with typical agent settings, involving communication, constraints, deadlines, and the occurrence of events has never been implemented.

To overcome such limitations, again in the context of the SOCS project, work has been done to extend IFF by constraint solving giving rise to the CIFF (IFF with Constraints) proof-procedure [45]. The introduction of constraints made it possible to implement core agent reasoning functionalities, such as temporal reasoning and abductive event calculus-based planning [74], and to demonstrate the feasibility of the approach in simple scenarios [2].

1.1 Agent societies: interaction specification and verification

We think of Multi-Agent Systems (MAS) as societies of computational entities. The counterpart of individual agent design and implementation is a number of issues related to agent interaction, at the social level. They include:

(a) definition of an agent interaction model. What is a suitable representation of messages and protocols? What is the meaning given to agent communication exchange, and what is the role of protocols in this?

(b) social goals. How to define the goal of agent interaction formally, in a specific context? Does the occurrence of a certain sequence of messages mean that a goal has been achieved?

(c) evaluation of the agent social behaviour. How to define the "correct" behaviour of agents interacting in a social environment? In relation to protocols, is it possible to tell out the agent interaction actions that are compliant to some defined protocol from those that are not? Can this be done dynamically, while agents are interacting?

(d) projection of the future states of agent interaction. Given some rules of interaction, which can be coded in terms of protocols, what is the expected behaviour of the MAS? Will it be able to develop a consistent sequence of events, or will it necessarily violate some protocols? Is there a way at all for agents to engage in compliant interaction, given some state of affairs and possibly a goal?

An important point to make is that to answer these questions we do not need to acquire information nor to make assumptions about the agent internal architecture. Agents could be computational entities, reactive systems, peers in a distributed computer system, even human actors. This detachment of agent architectural design information from interaction definition and study is called "social approach" to agent interaction, and it is advocated by milestones of the MAS literature, such as work by Castelfranchi [26], Singh [91], Yolum and Singh [104], and Fornara and Colombetti [53]. Other very influential approaches have been proposed, which focus on the relationship of agent interaction with agent internal reasoning and mental states, leading to a completely different semantic interpretation of agent communication, rooted in the theory of speech acts [32]. We will not delve here on this this hoary debate. We will focus instead on how to address the issues above with a computational logic-based and operational framework. We will show a semantics for agent interaction based on the concept of *social expectation*. Such a semantics provides a simple, intuitive and general, high-level abductive interpretation for agent societies [12], in which we define the concept of *social goal* and *social integrity constraints* (ICs). We will not only provide a declarative specification language for agent interaction, but we will also present a fully-fledged operational framework centered around the $\mathcal{S}$CIFF proof-procedure, whose properties of termination, soundness and completeness with respect to its declarative semantics, make it a suitable means for tackling verification of agent interaction.

We define compliance to protocols by matching of "socially relevant" events with social expectations. Such a unification is checked dynamically, as agents execute their programs, and as events of other kind occur in the environment. We explicitly represent timing of events, which makes it possible for such a framework to cater for sequences, deadlines and concurrent events.

We propose $\mathcal{S}$CIFF as a general framework, that can be applied to many situations involving reasoning with incomplete knowledge and dynamic occurrence of events, not necessarily related to agent interaction. Nevertheless, the main motivations of the research behind it come from the MAS domain. We will now briefly show, by examples, what typical situations we shall meet in such a domain. This will explain some choices we have made, and why we have decided to propose such a powerful formalism, rather than resorting to off-the-shelf solutions, based on existing proof-procedures or other logic-based operational tools. We will keep this part short, since the focus of this article, and its original contribution, is the operational framework of $\mathcal{S}$CIFF, which has never been presented before, and in its relationship with the declarative model, in terms of soundness and completeness and related results. The reader interested in more focussed discussions about the relation of the $\mathcal{S}$CIFF framework with other work of literature, specific case studies, and in the general issue of agent interaction can refer to previously published work, including [13, 15, 5, 10, 2, 7, 3].

1.2 Simple examples of agent interaction

Let us consider, for the sake of example, a typical two-agent setting. Agents a and b interact by message exchange. Messages may have a format in the style of KQML [49] or FIPA ACL [50], but we simplify the notation and indicate only sender, recipient, content and context by functor symbols.[1]

Agent a *request*s agent b to perform an action p. Let us use the *tell* functor to represent messages:

$$tell(a, b, request(p), d)$$

where the context is the *dialogue* d, occurring between a and b.

[1]For the sake of simplicity and generality, we will not propose a concrete content language. We will assume that, depending on the context, some content language is adopted. Some suitable interface can be used to map FIPA messages onto other notation. One such interface is that integrated in the SOCS-SI tool [7]. The reader interested in such a mapping can access a large number of sample protocols and communication traces from the SOCS-SI Protocol Repository [93], and download and execute them using the publicly available SOCS-SI tool [94].

When reasoning about agent interaction, especially in tackling issues like evaluation and prediction of agent interaction (points (*c*) and (*d*) above), we need to consider not only messages already exchanged by agents, but also messages that we expect agents to exchange in the future. We will then need some sort of hypothetical reasoning to figure what messages are expected to be sent, given the protocols and the current state of affairs regarding the interaction. Dually, we will also need to represent and reason about messages are expected *not* to be sent.

We will distinguish between three different categories of messages: *sent* messages, messages *expected* to be sent, and messages *expected not* to be sent. To this end, we will adopt the special symbols **H**, **E**, and **EN** to represent, respectively, happened events, positive expectations, and negative expectations. Using the same intuitive meaning of symbols as in the example above, we will write:

$$\mathbf{EN}(\mathit{tell}(a, b, \mathit{refuse}(p), d), T) \ \wedge \ T \ < \ \mathit{10}$$

to indicate that, in the context of dialogue d, a is *expected not* to express before time 10 his possible refusal to perform action p, i.e., he is expected not to send a message in the format $\mathit{tell}(a,b,\mathit{refuse}(p),d)$, at any time $T < \mathit{10}$.

Note that, if we want to express, like in this example, that some event is expected not to happen *at any time*, we need T to be *universally* quantified. Dually, if we want to express the fact that some event is expected to happen *at some time*, we need T to be *existentially* quantified. The variable quantification applies not only to variables representing time: we may need to refer to messages that are expected by *some* agent, or we may be expecting messages about some content but we do not know exactly what the message will be. For example, a could be expected to make a *bid* for a specific item i, by offering a *quote* Q ($\mathbf{E}(\mathit{tell}(a,B,\mathit{quote}(i,Q),T)$): but we do not know exactly what Q will be. Dually, when we consider expectations that certain events will not occur, we will have to express, for example, that agent a is expected not to send *any* message to agent b, or that we expect no message following some pattern to be sent between any two agents (e.g., $\mathbf{EN}(\mathit{tell}(a,b,M,T))$, or $\mathbf{EN}(\mathit{tell}(A,B,\mathit{insult}(I),T)))$.

The relations among past, happened events and expected, future events correspond to the idea of protocols. *Integrity constraints* (ICs) are used precisely to relate sent messages, belonging to the category of happened **H** events, with conjunctions of **E**/**EN** expectations. In doing so, disjunctive constructs are needed to express that after some event, several alternative future sequences of events would be equally consistent with the protocols. For example, after $\mathit{tell}(a, b, \mathit{request}(p), d)$ we may expect either $\mathit{tell}(b, a, \mathit{agree}(p), d)$ or $\mathit{tell}(b, a, \mathit{refuse}(p), d)$:

$$\begin{array}{l} \mathbf{H}(\mathit{tell}(a, b, \mathit{request}(p), d), T) \\ \rightarrow \ \mathbf{E}(\mathit{tell}(b, a, \mathit{agree}(p), d), \mathit{T1}) \ \wedge \ T \ < \ \mathit{T1} \\ \ \vee \ \mathbf{E}(\mathit{tell}(b, a, \mathit{refuse}(p), d), \mathit{T1}) \ \wedge \ T \ < \ \mathit{T1} \end{array}$$

In checking agents' compliance to given protocols, we need to verify whether expectations are met by events. We do this by unification/disunification of positive/negative expectations with events.

For example, given the expectation $\mathbf{E}(\mathit{tell}(b, a, \mathit{agree}(p), d), \mathit{T1}) \land \mathit{1} < \mathit{T1}$, we need to be able to understand whether there is a matching event which makes it "fulfilled", or whether it is "violated", i.e., missing a matching event.

Dually, given the negative expectation $\mathbf{EN}(\mathit{tell}(b, a, \mathit{refuse}(p), d), \mathit{T1}) \land \mathit{1} < \mathit{T1}$, we need to be able to tell whether it is fulfilled, which is the case until no matching event is generated in the society, or whether it becomes "violated" because of an event like: $\mathbf{H}(\mathit{tell}(b, a, \mathit{refuse}(p), d), \mathit{5})$.

Finally, let us consider, at a higher level, interaction as a means to achieve a goal. From the individual agent standpoint, in line with many agent theories such as Rao and Georgeff's BDI [78] or the KGP model of agency [67], we can imagine an internal process that considers goals or intentions, and through a deliberation phase generates communicative acts as part of plans to achieve them [32]. For example, we can imagine that by issuing *request*(p), agent a hopes to find itself into a state in which some other agent performs p. Some policies internal to a may have lead him to issue such a request. From the social standpoint, we can associate a goal to the interaction of agents, which is independent of the single agents, but reflects a status to be reached by the system as a whole. We argue that the production of expectations can be interpreted as the "social" goal of agent interaction in this sense. In the example above we can thus define a high-level goal, *satisfy_request*, which becomes true when an expectation about *agree*(p) is produced (and fulfilled), in this way:

$$\begin{array}{l}\mathit{satisfy\_request}(A, T) \leftarrow \\ \mathbf{E}(\mathit{tell}(B, A, \mathit{agree}(p), D), \mathit{T1}) \land \mathit{T1} < T\end{array}$$

A and T are here universally quantified variables, whereas B, D and $\mathit{T1}$ are existentially quantified: *satisfy_request* becomes true, given A and T, if (or for all A and T such that) there exists an agent B, a context D and a time $\mathit{T1}$ in which B *agree*s to perform p. The verification of fulfilment of such expectations will ensure that the social goal has been met.

1.3 Ten good reasons to use $\mathcal{S}$CIFF for reasoning about MAS interaction

These examples capture some of the most relevant features of the $\mathcal{S}$CIFF abductive framework. What does then $\mathcal{S}$CIFF offer, with respect to other existing abductive logic programming frameworks, and in the domain of agent interaction specification and verification? In a nutshell, with respect to other proposals, $\mathcal{S}$CIFF offers:

1. a *definition language* based on ICs. In the MAS domain, the $\mathcal{S}$CIFF language can be used to define agent interaction protocols. More in general, it can be used to describe the generation of hypotheses and expectations by events;

2. *variables* (e.g. to model time), and *constraints* on variables occurring in hypotheses and expectations: a feature which only a few protocol definition languages offer, but which is needed to express partially specified events and deadlines;

3. a very rich, flexible and transparent (i.e., implicit) *quantification* of variables, which allows to keep the $\mathcal{S}$CIFF language neat and simple, and makes it suitable to express interaction protocols in an intuitive way;

4. *social goals*, defined as predicates. Goals can express the social aim or outcome of some agent interaction, or they can be used to start an abductive derivation in the more classical ALP tradition;

5. the possibility to model *dynamically upcoming events*, which will be used by $\mathcal{S}$CIFF as an extension of a social environment knowledge base;

6. the ability to generate *positive and negative expectations*, beside making hypotheses, and the concepts of *fulfilment and violation* of expectations. These features are needed to implement dynamic checking of protocol compliance, which is a distinguishing feature of the $\mathcal{S}$CIFF framework, in the sense that at the time of writing, to the best of our knowledge, no other frameworks offer this feature with such a wide range of protocols;

7. a *declarative semantics* given in terms of ALP. In the MAS domain, it can be used to give an abductive interpretation to agent societies;

8. a *proof-procedure*, called $\mathcal{S}$CIFF, which extends Fung and Kowalski's IFF [56] in the directions above (variables, constraints, quantification, expectations, hypotheses confirmation/disconfirmation, dynamic occurrence of events). $\mathcal{S}$CIFF generates expectations from events, checks their confirmation/disconfirmation, and predicates about the achievement of social goals;

9. the properties of *termination*, *soundness*, and *completeness*, under reasonable assumptions, of $\mathcal{S}$CIFF. Together with the presentation of the $\mathcal{S}$CIFF proof-procedure, these properties represent the main contribution of this article and are among the most important achievements of the overall framework, since they set a correspondence between $\mathcal{S}$CIFF and its declarative, abductive semantics;

10. an efficient *implementation.* $\mathcal{S}$CIFF has been implemented on top of CHR [54, 4] and it is at the core of the SOCS-SI tool [7]. SOCS-SI is publicly available [94].

These features make $\mathcal{S}$CIFF a suitable framework for the MAS domain. Independently of this, and more generally, many of the individual features of $\mathcal{S}$CIFF, like its very expressive quantification of variables are, to the best of our knowledge, to be hardly found in most existing operational abductive frameworks.

In this article, we present the $\mathcal{S}$CIFF abductive framework. We present the declarative and operational semantics of $\mathcal{S}$CIFF in terms of transition rules, as an extension of Fung and Kowalski's IFF's operational semantics [56]. We then present three very important results: first, termination of $\mathcal{S}$CIFF under reasonable assumptions (i.e., in a way that does not prevent us from specifying any of the protocols that we have considered so far). Second, soundness of the proof-procedure with respect to its declarative semantics. Finally, we prove that $\mathcal{S}$CIFF is complete for a limited though very significant set of programs.

After the preliminaries, we introduce in Sect. 3 our framework's syntax. We then proceed to Sect. 4 and 5 which provide its declarative and operational semantics; results of termination, soundness and completeness are shown in Sect. 6, while Sect. 7 sketches the implementation of $\mathcal{S}$CIFF inside SOCS-SI. In Sect. 8 we present three applications of $\mathcal{S}$CIFF in different contexts, and finally we review related work in Sect. 9. Additional details about the syntax of the $\mathcal{S}$CIFF language and allowedness criteria for proving soundness can be found in [14].

2 Background and notation

In the remainder of the article, we assume a basic familiarity with the concepts, results and conventions of Logic Programming. A good introduction is that provided by Lloyd [73]. The words *integer*, *variable*, *term*, *atom* will be used in the following with their usual meaning in Logic Programming [73]. We adopt the Prolog convention that reserves capital letters $(X, Y, Z, ...)$ for variables, lower-case letters $(a, b, c, \dots)$ for constants and functors. We use capital-letter boldface fonts $(\mathbf{E}, \mathbf{H}, \mathbf{EN})$ for special functors defined in our language. The symbol $\neg$ stands for explicit negation, while *not* is for negation by failure.

As in ACLP [65], our language integrates Constraint Logic Programming [60] and Abductive Logic Programming [62].

Constraint Logic Programming (CLP) is a class of declarative languages. Each CLP language can be seen as an instance of a general CLP($\mathcal{X}$) scheme where each instance can be specified by instantiating the parameter $\mathcal{X}$ [61]. The parameter $\mathcal{X}$ represents a 4-tuple $(\Sigma_{\mathcal{X}}, \mathcal{D}_{\mathcal{X}}, \mathcal{L}_{\mathcal{X}}, \mathcal{T}_{\mathcal{X}})$ where $\Sigma_{\mathcal{X}}$ is a signature, $\mathcal{D}_{\mathcal{X}}$ is a $\Sigma_{\mathcal{X}}$-structure, $\mathcal{L}_{\mathcal{X}}$ is a class of $\Sigma_{\mathcal{X}}$-formulas and $\mathcal{T}_{\mathcal{X}}$ is a first-order $\Sigma_{\mathcal{X}}$-theory.

We will assume that the symbols $=$ and $\neq$ belong to the signature $\Sigma_{\mathcal{X}}$. As usual in CLP, the constraint $=$ replaces unification, and will be called equality constraint. The constraint $\neq$ is called disequality constraint; the semantics of $A \neq B$ is that of $not(A = B)$. With an abuse of notation, we will also apply equality (resp. disequality) on atoms; in such a case, we mean the conjunction (resp. disjunction) of equalities (disequalities) of corresponding arguments.

In our language, constraints can also be applied to universally quantified variables. The meaning, in such a case, is that of *quantifier restrictions* [25]. The semantics of quantifier restrictions is different for the case of universally quantified and existentially quantified variables, as follows (where $c \in \Sigma_{\mathcal{X}}$):

$$\begin{array}{lcl} \forall_{X:c(X)} p(X) & \Longleftrightarrow & \forall X \ (\ c(X) \rightarrow p(X)\ ) \\ \exists_{X:c(X)} p(X) & \Longleftrightarrow & \exists X \ (\ c(X) \wedge p(X)\ ) \end{array} \tag{1}$$

Notice that for existentially quantified variables, quantifier restrictions have the same meaning of constraints; for this reason, in the rest of the paper, we will use the terms *constraint* and *quantifier restriction* interchangeably.

Given a variable X, with $QR(X)$ we will denote the quantifier restrictions on X.

An *abductive logic program* [62] is a triple $\langle P, Ab, IC \rangle$ where:

- Ab a set of *abducible predicates.*
- P is a (normal) logic program, that is, 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 ($i = 0, \ldots, m+n$) is an atom. A_0 is built on a signature $\Sigma_{\mathcal{P}}$ of *defined predicates*, while each A_i ($i = 0, \ldots, m+n$) is built on a signature $\Sigma_{\mathcal{P}} \cup Ab \cup \Sigma_{\mathcal{X}}$. 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.
- IC is a set of integrity constraints, also built on $\Sigma_{\mathcal{P}} \cup Ab \cup \Sigma_{\mathcal{X}}$.

Without loss of generality, we assume $\Sigma_{\mathcal{P}}$, Ab and $\Sigma_{\mathcal{X}}$ to be pairwise disjoint.

Abducible predicates (or simply abducibles) are the predicates about which assumptions (hypotheses, abductions) can be made. For example, a ground abducible can be assumed to be true, or false. We will represent abducible predicate symbols in boldface, as in $\mathbf{a}(X)$.

Given an abductive logic program $T = \langle P, Ab, IC \rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (*abductive explanation*) in predicates in Ab which, together with P, "entails" G, i.e., $P \cup \Delta \models G$, and such that $P \cup \Delta$ "satisfies" IC, e.g. $P \cup \Delta \models IC$ (see [62] for other possible notions of integrity constraint "satisfaction"). Here, the notion of "entailment" $\models$ depends on the semantics associated with the logic program P (there are many different choices for such semantics, as it is well-documented in the Logic Programming literature [20]).

If F is a formula, with $vars(F)$ we mean the set of variables occurring in F.

3 Syntax

In this section, we define the syntax of the logic language used in the $\mathcal{S}$CIFF framework. The language is composed of entities for expressing:

- events, expectations about events, and hypotheses;
- relationships between events and hypotheses.

3.1 Representation of dynamically happening events

3.1.1 Events

Events are the abstraction used to represent the actual observations.

Definition 3.1 *An* event *is an atom:*

- *with predicate symbol* **H***;*
- *whose first argument is a ground term; and*
- *whose second (optional) argument is an integer.*

Intuitively, the first argument is meant to represent the description of the happened event, according to application-specific conventions, and the second argument is meant to represent the time at which the event has happened:

Example 3.2

$$\mathbf{H}(\mathit{tell}(\mathit{alice}, \mathit{bob}, \mathit{query\_ref}(\mathit{phone\_number}), \mathit{dialog\_id}), \mathit{10}) \tag{2}$$

could represent the fact that alice asked bob his phone_number with a query_ref message, in the context identified by dialog_id, at time 10.

A *negated event* is an event with the unary prefix operator *not* applied to it.[2] We will call *history* a set of events, and denote it with the symbol **HAP**.

3.1.2 Expectations

Expectations are the abstraction we use to represent the desired events. In a MAS setting, they would represent the ideal behaviour of the system, i.e., the actions that, once performed, would make the system compliant to its specifications.[3]

Expectations are of two types:

[2] *not* represents default negation (see declarative semantics of the $\mathcal{S}$CIFF framework, Sect. 4).

[3] Our choice of the terminology "expectation" is intended to stress that observations cannot be enforced, but only expected, to be as we would like them to be.

- *positive*: representing some event that is expected to happen;
- *negative*: representing some event that is expected *not* to happen.

Definition 3.3 *A* positive expectation *is an atom:*

- *with predicate symbol* **E***;*
- *whose first argument is a term; and*
- *whose second (optional) argument is a variable or an integer.*

Intuitively, the first argument is meant to represent an event description, and the second argument is meant to tell for what time the event is expected. If no time is specified, it means that the event is expected to happen any time.

Example 3.4 *The atom*

$$\mathbf{E}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{inform}(\mathit{phone\_number}, X), \mathit{dialog\_id}), T_i) \qquad (3)$$

could represent that bob is expected to inform alice at some time T_i that the value for the piece of information identified by phone_number is X, in the context identified by dialog_id.

A *negated positive expectation* is a positive expectation with the explicit negation operator $\neg$ applied to it.

As the example shows, expectations can contain variables, as it might be desirable to leave the expected behaviour not completely specified. Variables in positive expectations will be existentially quantified, supporting the intuition, as we have seen in Ex. 3.4.

Definition 3.5 *A* negative expectation *is an atom:*

- *with predicate symbol* **EN***;*
- *whose first argument is a term; and*
- *whose second (optional) argument is a variable or an integer.*

Intuitively, the first argument is meant to represent an event description, and the second argument is meant to tell for what time the event is expected not to happen. If time is not specified, it means that the event is expected not to happen at any time.

Example 3.6 *The atom*

$$\mathbf{EN}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r) \tag{4}$$

could represent that bob is expected not *to refuse to alice his phone_number, in the context identified by dialog_id, at* any *time.*

A *negated negative expectation* is a negative expectation with the explicit negation operator $\neg$ applied to it.

Note that $\neg\mathbf{E}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r)$ is different from $\mathbf{EN}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{refuse}(\mathit{phone\_number}), \mathit{dialog\_id}), T_r)$. The intuitive meaning of the former is: no *refuse* is expected by Bob (if he does, we simply did not expect him to), whereas the latter has a different, stronger meaning: it is expected that Bob does not utter *refuse* (by doing so, he would frustrate our expectations).

As the example shows, variables in negative expectations are naturally interpreted as universally quantified (Bob should *never* refuse). However, the same variable may occur in two distinct expectations, one of which positive, the other negative. In that case, the quantification will be existential (i.e., the convention adopted for positive expectations will prevail). This follows the intuitions, as we can see in the following example.

Example 3.7 *It is expected that (at least one) agent A performs task t_1, and that no other agent B interrupts A:*

$$\mathbf{E}(\mathit{perform}(A, t_1)), \mathbf{EN}(\mathit{interrupt}(B, A)).$$

Variable A is existentially quantified, while B is quantified universally.

The syntax of events and expectations is summarised in Tab. 3.1, and it will be used as such by the subsequent Tab. 3.2 and 3.3.

We also introduce, for ease of presentation, the syntactic element *ExistLiteral*, that lists the literals that are existentially quantified. Again, for simplifying the following presentation, we define *NbfLiteral*, that intuitively indicates negative literals with negation by failure. By *AbducibleAtom* we mean an atom built on an abducible predicate (i.e., a predicate in the set *Ab*).

3.2 Social specifications

A Social specification, i.e, a specification of the society in the $\mathcal{S}$CIFF framework, is composed of two elements:

- A *Knowledge Base*;
- A set of *Integrity Constraints*.

Table 3.1 Syntax of events and expectations

$$\begin{array}{rcl}
\mathit{EventLiteral} & ::= & [\mathit{not}]\mathit{Event} \\
\mathit{Event} & ::= & \mathbf{H}(\mathit{GroundTerm}[, \mathit{Integer}]) \\
\\
\mathit{ExpLiteral} & ::= & \mathit{PosExpLiteral} \mid \mathit{NegExpLiteral} \\
\mathit{PosExpLiteral} & ::= & [\neg]\mathit{PosExp} \\
\mathit{NegExpLiteral} & ::= & [\neg]\mathit{NegExp} \\
\mathit{PosExp} & ::= & \mathbf{E}(\mathit{Term}[, \mathit{Variable} \mid \mathit{Integer}]) \\
\mathit{NegExp} & ::= & \mathbf{EN}(\mathit{Term}[, \mathit{Variable} \mid \mathit{Integer}]) \\
\\
\mathit{ExistLiteral} & ::= & \mathit{PosExpLiteral} \mid \mathit{AbducibleLiteral} \mid \mathit{Literal} \\
\mathit{NbfLiteral} & ::= & \mathit{not\ Atom} \mid \mathit{not\ AbducibleAtom} \\
\mathit{Literal} & ::= & [\mathit{not}]\mathit{Atom} \\
\mathit{AbducibleLiteral} & ::= & [\mathit{not}]\mathit{AbducibleAtom}
\end{array}$$

3.2.1 Knowledge Base

The Knowledge Base (KB) is a set of *Clause*s in which the body can contain (besides defined and abducible literals), expectation literals and constraints.

Intuitively, the KB is used to express declarative knowledge about the specific application domain.

Table 3.2 Syntax of the Knowledge Base

$$\begin{array}{rcl}
KB & ::= & [\mathit{Clause}]^{\star} \\
\mathit{Clause} & ::= & \mathit{Head} \leftarrow \mathit{Body} \\
\mathit{Head} & ::= & \mathit{Atom} \\
\mathit{Body} & ::= & \mathit{ExtLiteral}\ [\ \wedge\ \mathit{ExtLiteral}\ ]^{\star} \mid \mathit{true} \\
\mathit{ExtLiteral} & ::= & \mathit{Literal} \mid \mathit{AbducibleLiteral} \mid \mathit{ExpLiteral} \mid \mathit{Restriction}
\end{array}$$

The syntax of the Knowledge Base is given in Tab. 3.2, and it will be used as such also in Tab. 3.3.

Allowedness conditions The operational semantics (Sect. 5) will require some syntactic restrictions, which we will now introduce. In the sequel and throughout this article, we will assume that such restrictions hold in all cases we consider.

As usual in Logic Programming, we need to avoid floundering of variables in negative literals [73]:

Definition 3.8 *A clause Head* $\leftarrow$ *Body is* allowed *if and only if every variable that occurs in a NbfLiteral in Body, also occurs in the Head or in at least one ExistLiteral.*

Variable quantification and scope The quantification and scope of variables is implicit. In each clause, the variables are quantified as follows:

- universally with scope the $Clause$ if they occur in the $Head$ or in at least one $ExistLiteral$;
- otherwise (if they occur only in negative expectations and possibly restrictions) universally, with scope the *Body*.

This means that clauses will be quantified as in most other abductive logic programming languages, and in particular, in the language interpreted by the IFF proof-procedure, except for negative expectations. Variables that occur only in a negative expectation will be universally quantified with scope the *Body*. Let us see an example:

Example 3.9 *In order to have a task completed, it is expected that an agent performs it, and no agent is expected to interrupt the agent performing that task.*

$$\mathit{completed}(Task) \leftarrow \mathbf{E}(\mathit{perform}(A, Task)), \mathbf{EN}(\mathit{interrupt}(B, A)).$$

The quantification of the variables is most intuitive:

$$(\forall Task, \forall A)\ (\\ \quad \mathit{completed}(Task) \leftarrow \mathbf{E}(\mathit{perform}(A, Task)), (\forall B)\ (\ \mathbf{EN}(\mathit{interrupt}(B, A))\ ) \\ \quad)\ .$$

Definition 3.10 *A Clause is* restriction allowed *if the variables that are universally quantified with scope the body do not occur in quantifier Restrictions, and each variable that occurs in a restriction also occurs in at least one positive expectation PosExp, or in AbducibleLiteral in the body.* [4]

For example, the clause:

$$p \leftarrow \mathbf{EN}(X), X < 10$$

is not restriction allowed, because it contains a variable X that is universally quantified with scope the *Body*, and that is also in a quantifier restriction. Similarly, the clause:

$$p \leftarrow a(Y), Y < 10$$

[4]Def. 3.10 is needed for a correct handling of defined predicates literals in the integrity constraints. In fact, it turns out that unfolding a clause which is not restriction allowed could generate an integrity constraint which is not restriction allowed (see Def. 3.14). This will be clearer when we present the operational semantics in Sect. 5.

is not restriction allowed, because it contains an existentially quantified variable Y, with scope the *Body*, which does not appear in any *PosExp* literal (**E**) in the *Body.*

3.2.2 Goal

Thanks to the abductive interpretation, goal-directed societies are possible in the $\mathcal{S}$CIFF framework; non-goal directed societies are also supported, by considering the atom *true* as goal.

The syntax of the goal is the same as the *body* of a clause (Tab. 3.2). In order to avoid floundering, variables in the goal cannot occur only in *NbfLiteral*s. The quantification rules are the following:

- All variables that occur in an *ExistLiteral* are existentially quantified.
- All remaining variables are universally quantified.

Note that these rules are equivalent to those of the variables in the body of a clause (Sect. 3.2.1), considering that $\forall X.(H \leftarrow B)$ is equivalent to $H \leftarrow (\exists X.B)$ when X does not occur in H.

3.2.3 Integrity Constraints

Integrity Constraints (also ICs, for short, in the following) are implications that, operationally, are used as forward rules, as will be explained in Sect. 5. Declaratively, they relate the various entities in the $\mathcal{S}$CIFF framework, i.e., expectations, events, abducibles, and constraints/restrictions, together with the predicates in the knowledge base. The syntax of ICs is given in Tab. 3.3: the *Body* of ICs can contain

Table 3.3 Syntax of Integrity Constraints (ICs)

$\mathcal{IC}$	::=	$[IC]^\star$
IC	::=	*Body* $\rightarrow$ *Head*
Body	::=	(*EventLiteral* $\mid$ *ExpLiteral* $\mid$ *AbducibleLiteral*) [$\wedge$ *BodyLiteral* $]^\star$
BodyLiteral	::=	*EventLiteral* $\mid$ *ExtLiteral*
Head	::=	*HeadDisjunct* [$\vee$ *HeadDisjunct* $]^\star$ $\mid$ *false*
HeadDisjunct	::=	*ExtLiteral* [$\wedge$ *ExtLiteral*$]^\star$

conjunctions of all elements in the language (namely, **H**, **E**, and **EN** literals, definite and abducible literals and restrictions), and their *Head* contains a disjunction of conjunctions of all the literals in the language, except for **H** literals.

Let us now consider an interaction protocol taken from the MAS literature:

Table 3.4 Integrity Constraints and Knowledge Base for the *query_ref* specification.

$$
\begin{array}{rl}
 & \mathbf{H}(\mathit{tell}(A, B, \mathit{query\_ref}(\mathit{Info}), D), T) \ \wedge \\
 & \mathit{qr\_deadline}(TD) \\
\rightarrow & \mathbf{E}(\mathit{tell}(B, A, \mathit{inform}(\mathit{Info}, \mathit{Answer}), D), T1) \ \wedge \\
 & T1 \ < \ T \ + \ TD \\
\vee & \mathbf{E}(\mathit{tell}(B, A, \mathit{refuse}(\mathit{Info}), D), T1) \ \wedge \\
 & T1 \ < \ T \ + \ TD \\
 & \\
 & \mathbf{H}(\mathit{tell}(A, B, \mathit{inform}(\mathit{Info}, \mathit{Answer}), D), Ti) \\
\rightarrow & \mathbf{EN}(\mathit{tell}(A, B, \mathit{refuse}(\mathit{Info}), D), Tr)
\end{array}
$$

$\mathit{qr\_deadline}(10).$

Example 3.11 *Tab. 3.4 shows the ICs for the query_ref [50] specification.*

Intuitively, the first IC means that if agent A sends to agent B a query_ref message, then B is expected to reply with either an inform or a refuse message by TD time units later, where TD is defined in the Knowledge Base by the qr_deadline predicate.

The second IC means that, if an agent A sends an inform message, then it is expected not to send a refuse message about the same Info, to the same agent B and in the context of the same interaction D at any time.

Variable quantification and scope All variables in an integrity constraint should occur in an *EventLiteral*, *ExpLiteral*, or *AbducibleAtom*. The rules of scope and quantification for the variables in an integrity constraint $\mathit{Body} \rightarrow \mathit{Head}$ are as follows:

1. Each variable that occurs both in *Body* and in *Head* is quantified universally, with scope the integrity constraint.
2. Each variable that occurs only in *Head* cannot occur only in *NbfLiteral*s and
   - if it occurs in at least one *ExistLiteral* is existentially quantified and has as scope the disjunct where it occurs;
   - otherwise it is quantified universally.
3. Each variable that occurs only in *Body* is quantified with scope *Body* as follows:
   (a) existentially if it occurs in at least one *ExistLiteral* or *Event*;
   (b) universally, otherwise.

The given quantification rules let the user write integrity constraints without explicitly stating the quantification of the variables, and typically capture the intuitive meaning of the rules in protocols. Let us show it with an example.

Example 3.12 *Consider the following example:*

$$\mathbf{H}(p(X,Y)), not\ \mathbf{H}(q(Z,X)) \rightarrow \mathbf{E}(r(X,K)), \mathbf{EN}(f(Y,J))$$

Variables X and Y occur both in the body and in the head. Coherently with the literature in abduction, they will be universally quantified with scope the whole IC.

Variables K and J occur only in the Head. The quantification rules for those variables are the same as for the Goal (see Sect. 3.2.2), i.e., existential for K and universal for J.

Finally, $\neg\mathbf{H}(q(Z,X))$ means that, if no event happens matching $q(Z,X)$, then the IC's head should be true. For instance, if the set of happened events is

$$\mathbf{H}(p(2,1)), \mathbf{H}(q(3,2))$$

it is quite natural to understand the Body as false (the second event makes not $\mathbf{H}(q(Z,X))$ false). So, the existence of one atom ($\mathbf{H}(q(3,2))$ in the example) is enough for making not $\mathbf{H}(q(Z,X))$ false. This means that the IC should be read as "if $\mathbf{H}(p(X,Y))$ and for all values Z, $\mathbf{H}(q(Z,X))$ is false, the *Head* must hold". *Variable Z should be quantified as follows:*

$$[\forall Z \ldots, not\ \mathbf{H}(q(Z,X))] \rightarrow \ldots$$

thus, the quantification rules give the quantification

$$\forall X,Y \exists Z,K \forall J.\ \mathbf{H}(p(X,Y)), not\ \mathbf{H}(q(Z,X)) \rightarrow \mathbf{E}(r(X,K)), \mathbf{EN}(f(Y,J))$$

Allowedness conditions As in the case of the Knowledge Base syntax, the following syntactic restrictions are motivated by the operational semantics, and will be supposed to hold throughout the paper.

A variable cannot occur in an IC only in *NbfLiterals*. If it does occur in a literal with negation by failure, it necessarily has to appear in the same IC also in at least another literal within predicate symbol **H**, **E**, **EN**, or an abducible atom.

Since variables in positive expectations are existentially quantified, integrity constraints should not entail universally quantified positive expectations. For example,

$$not\ \mathbf{H}(p(A)) \rightarrow \mathbf{E}(q(A))$$

would entail in an empty history that $\forall A.\mathbf{E}(q(A))$. We avoid such situations with the following allowedness condition.

Definition 3.13 *An Integrity Constraint Body* $\rightarrow$ *Head is* quantifier allowed *if*

- *each variable that occurs in an ExistLiteral in Head either does not occur in Body, or it occurs in the Body in at least one Event or in a PosExpLiteral, or in an AbducibleAtom;*
- *each variable that occurs in a NbfLiteral in Body also occurs in at least one Event or PosExpLiteral or in an AbducibleAtom in Body*[5]*.*

Definition 3.14 *An integrity constraint is* restriction allowed *if*

- *all the variables that are universally quantified with scope Body do not occur in Restrictions;*
- *the other variables (that occur only in Head, or both in Head and in Body) can occur in Restrictions. Each Restriction occurring in the integrity constraint should:*
  - *either involve only variables that also occur in PosExpLiterals, Events or AbducibleAtom (in the same disjunct, or in the body),*
  - *or involve* one *variable that also occurs in at least one NegExpLiteral, and possibly other variables which only occur in Events.*

3.2.4 Abductive Specification

Given a Knowledge Base KB and a set $\mathcal{IC}$ of Integrity Constraints, we call the pair $\langle KB, \mathcal{IC} \rangle$ an *Abductive Specification.* We will often use the symbol $\mathcal{S}$ to denote an abductive specification.

Definition 3.15 *An abductive specification* $\mathcal{S} = \langle KB, \mathcal{IC} \rangle$ *is* quantifier allowed *if all the integrity constraints in* $\mathcal{IC}$ *are quantifier allowed.* $\mathcal{S}$ *is* restriction allowed *if all the clauses in* KB *and all the integrity constraints in* $\mathcal{IC}$ *are restriction allowed.* $\mathcal{S}$ *is* allowed *if it is quantifier allowed and restriction allowed, and* KB *is allowed.*

As a recap on allowedness conditions, we have the following table. For all the three syntactic elements (Clauses, Goal, and ICs), variables cannot occur only in *NbfLiteral*s. Besides, the following conditions must hold in order for Clauses/ICs to be restriction-/quantifier-allowed:

[5]This rule descends from the previous one, considering that $not(A), B \rightarrow C$ is equivalent to $C \rightarrow A \vee B$.

	Clause	Integrity Constraint
is restriction-allowed	if QRs only appear on vars in $\exists$ abducibles	if no QR appears in $\forall$ vars with scope *Body* and QRs do not involve more than one $\forall$ var
is quantifier-allowed	*always*	if *ExistLiterals* in *Head* do not contain vars that occur in the *Body* only in *not* **H**, $[\neg]$**EN** and all vars of *NbfLiterals* in *Body* also occur in other **H**, $[\neg]$**E**, or abducibles in *Body*

4 Declarative Semantics

In the following, we describe the (abductive) declarative semantics of the $\mathcal{S}$CIFF framework, which is inspired by other abductive frameworks, but introduces the concept of fulfilment, used to express a correspondence between the expected and the actual observations. The declarative semantics of a social specification is given for each specific history (see Sect. 3.1). We call a specification grounded on a history an *instance* of the society.

Definition 4.1 *Given an abductive specification* $\mathcal{S} = \langle KB, \mathcal{IC} \rangle$ *and a history* **HAP**, $\mathcal{S}_{\mathbf{HAP}}$ *represents the pair* $\langle \mathcal{S}, \mathbf{HAP} \rangle$, *called the* **HAP**-instance of $\mathcal{S}$ *(or simply an* instance of $\mathcal{S}$*).*

In this way, $\mathcal{S}_{\mathbf{HAP}^i}$, $\mathcal{S}_{\mathbf{HAP}^f}$ will denote different instances of the same abductive specification $\mathcal{S}$, based on two different histories: $\mathbf{HAP}^i$ and $\mathbf{HAP}^f$.

We adopt an abductive semantics for the society instance. The abductive computation produces a set Δ of hypotheses, which is partitioned in a set ΔA of general hypotheses and a set **EXP** of *expectations.* The set of abduced literals should entail the goal and satisfy the integrity constraints.

Definition 4.2 *Given an abductive specification* $\mathcal{S} = \langle KB, \mathcal{IC} \rangle$, *an instance* $\mathcal{S}_{\mathbf{HAP}}$ *of* $\mathcal{S}$, *and a goal* $\mathcal{G}$, Δ*is an abductive explanation of* $\mathcal{S}_{\mathbf{HAP}}$ *if:*

$$Comp(KB \cup \mathbf{HAP} \cup \Delta) \cup \text{CET} \ \cup T_{\mathcal{X}} \models \mathcal{IC} \tag{5}$$

$$Comp(KB \cup \Delta) \cup \text{CET} \ \cup T_{\mathcal{X}} \models \mathcal{G} \tag{6}$$

where Comp represents the completion *of a theory,* CET *is Clark's Equational Theory [31], and* $T_{\mathcal{X}}$ *is the theory of constraints [61].*

The symbol $\models$ can be interpreted in two or in three valued logics, depending on the type of application we are envisaging, and on the situation.

We also require consistency with respect to explicit negation [20] and between positive and negative expectations.

Definition 4.3 *A set* **EXP** *of expectations is* $\neg$-consistent *if and only if for each (ground) term p:*

$$\{\mathbf{E}(p), \neg\mathbf{E}(p)\} \not\subseteq \mathbf{EXP} \qquad \textit{and} \qquad \{\mathbf{EN}(p), \neg\mathbf{EN}(p)\} \not\subseteq \mathbf{EXP}. \tag{7}$$

Definition 4.4 *A set* **EXP** *of expectations is* **E**-consistent *if and only if for each (ground) term p:*

$$\{\mathbf{E}(p), \mathbf{EN}(p)\} \not\subseteq \mathbf{EXP} \tag{8}$$

The following definition establishes a link between happened events and expectations, by requiring positive expectations to be matched by events, and negative expectations not to be matched by events.

Definition 4.5 *Given a history* **HAP***, a set* **EXP** *of expectations is* **HAP**-fulfilled *if and only if*

$$\forall \mathbf{E}(p) \in \mathbf{EXP} \Rightarrow \mathbf{H}(p) \in \mathbf{HAP} \qquad \forall \mathbf{EN}(p) \in \mathbf{EXP} \Rightarrow \mathbf{H}(p) \notin \mathbf{HAP} \tag{9}$$

Otherwise, **EXP** *is* **HAP**-violated*.*

When all the given conditions (5-9) are met, we say that the goal is *achieved* and **HAP** is compliant to $\mathcal{S}_{\mathbf{HAP}}$ with respect to $\mathcal{G}$, and we write $\mathcal{S}_{\mathbf{HAP}} \models_{\Delta} \mathcal{G}$.

In the remainder of this article, when we simply say that a history **HAP** is compliant to an abductive specification $\mathcal{S}$, we will mean that **HAP** is compliant to $\mathcal{S}$ with respect to the goal *true*. We will often say that a history **HAP** *violates* a specification $\mathcal{S}$ to mean that **HAP** is not compliant to $\mathcal{S}$. When **HAP** is apparent from the context, we will often omit mentioning it.

Example 4.6 *Consider the query_ref abductive specification* $\mathcal{S} = \langle KB, \mathcal{IC} \rangle$*, where* KB *and* $\mathcal{IC}$ *are defined in Tab. 3.4. The history*

$$\begin{aligned} \{&\mathbf{H}(\mathit{tell}(\mathit{alice}, \mathit{bob}, \mathit{query\_ref}(\mathit{phone\_number}), \mathit{dialog\_id}), \mathit{10}), \\ &\mathbf{H}(\mathit{tell}(\mathit{bob}, \mathit{alice}, \mathit{inform}(\mathit{phone\_number}, \mathit{5551234}), \mathit{dialog\_id}), \mathit{12})\} \end{aligned} \tag{10}$$

is compliant to $\mathcal{S}$*.*

5 The $\mathcal{S}$CIFF proof procedure

The operational semantics of $\mathcal{S}$CIFF is given by an abductive proof procedure.

Since the language and declarative semantics of the $\mathcal{S}$CIFF framework are closely related with the IFF abductive framework [56], the $\mathcal{S}$CIFF proof procedure has also been inspired by the IFF proof procedure. However, some modifications were necessary, as mentioned in the introductory sections of this article. As a result, $\mathcal{S}$CIFF is a substantial extension of IFF, and the main differences between the frameworks are, in a nutshell:

- $\mathcal{S}$CIFF supports the dynamical happening of events, i.e., the insertion of new facts in the knowledge base during the computation;
- $\mathcal{S}$CIFF supports universally quantified variables in abducibles;
- $\mathcal{S}$CIFF supports quantifier restrictions;
- $\mathcal{S}$CIFF supports the concepts of fulfilment and violation (see Def. 4.5).

5.1 Data Structures

The $\mathcal{S}$CIFF proof procedure is based on a rewriting system transforming one node to another (or to others). In this way, starting from an initial node, it defines a proof tree.

A node can be either the special node *false*, or defined by the following tuple

$$T \equiv \langle R, CS, PSIC, \Delta A, \Delta P, \mathbf{HAP}, \Delta F, \Delta V \rangle. \tag{11}$$

We partition the set of expectations **EXP** into the confirmed (ΔF), disconfirmed (ΔV), and pending (ΔP) expectations. The other elements are:

- R is the resolvent: a conjunction, whose conjuncts can be literals or disjunctions of conjunctions of literals
- CS is the constraint store: it contains CLP constraints and quantifier restrictions
- $PSIC$ is a set of implications, called partially solved integrity constraints
- ΔA is the set of general abduced hypotheses (the set of abduced literals, except those representing expectations)
- **HAP** is the history of happened events, represented by a set of events, plus a *open/closed* attribute (see transition *closure* in the following)

If one of the elements of the tuple is *false*, then the whole tuple is the special node *false*, which cannot have successors. In the following, we indicate with Δ the set $\Delta A \cup \Delta P \cup \Delta F \cup \Delta V$.

5.1.1 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 $\mathcal{G}$, a set of social integrity constraints $\mathcal{IC}$, and an initial history $\mathbf{HAP}^i$, we build the first node in the following way:

$$T_0 \equiv \langle \{\mathcal{G}\}, \emptyset, \mathcal{IC}, \emptyset, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$$

i.e., the resolvent R is initially the query ($R_0 = \{G\}$) and the set of partially solved integrity constraints $PSIC$ is the set of integrity constraints ($PSIC_0 = \mathcal{IC}$).

The other nodes $T_j, j > 0$, are obtained by applying the transitions that we will define in the next section, until no further transition can be applied (we call this last condition *quiescence*).

Definition 5.1 *Given an instance $\mathcal{S}_{\mathbf{HAP}^i}$ of a social specification $\mathcal{S} = \langle KB, \mathcal{IC} \rangle$ and a set $\mathbf{HAP}^f \supseteq \mathbf{HAP}^i$ there exists a* successful derivation *for a goal G iff the proof tree with root node $\langle \{G\}, \emptyset, \mathcal{IC}, \emptyset, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$ has at least one leaf node*

$$\langle \emptyset, CS, PSIC, \Delta A, \Delta P, \mathbf{HAP}^f, \Delta F, \emptyset \rangle$$

where CS is consistent, and ΔP contains only negations of expectations $\neg\mathbf{E}$ and $\neg\mathbf{EN}$. In such a case, we write:

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

From a 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 are called *expectation answers*. To compute an expectation answer, a substitution σ' is computed such that

- σ' replaces all variables in N that are not universally quantified by a ground term
- σ' satisfies all the constraints in the store CS_N.

If the constraint solver is (theory) complete [61] (i.e., for each set of constraints c, the solver always returns *true* or *false*, and never *unknown*), then there will always exist a substitution σ' for each non-failure leaf node N. Otherwise, if the solver is incomplete, σ' may not exist. The non-existence of σ' is discovered during the answer extraction phase. In such a case, the node N will be marked as a failure node, and another success node can be selected (if there is one).

Definition 5.2 *Let $\sigma = \sigma'|_{vars(G)}$ be the restriction of σ' to the variables occurring in the initial goal $\mathcal{G}$. Let $\Delta_N = (\Delta F_N \cup \Delta P_N \cup \Delta A_N)\sigma'$. The pair (Δ_N, σ) is the* expectation answer *obtained from the node N.*

5.2 Variables

5.2.1 Quantification

Concerning variable quantification, $\mathcal{S}$CIFF differs from IFF in the following aspects:

- in IFF, all the variables that occur in the resolvent or in abduced literals are existentially quantified, while the others (that occur only in implications) are universally quantified; in $\mathcal{S}$CIFF, variables that occur in the resolvent or in abducibles can be universally quantified (as **EN** expectations can contain universally quantified variables);

- in IFF, variables in an implication are existentially quantified if they also occur in an abducible or in the resolvent, while in $\mathcal{S}$CIFF variables in implications can be existentially quantified even if they do not occur elsewhere (see Example 3.12).

For these reasons, in the $\mathcal{S}$CIFF proof procedure the quantification of variables is explicit.

5.2.2 Scope

The scope of the variables differs depending on where they occur:

- if they occur in the resolvent or in abducibles, their scope is the whole tuple representing the node (see Sect. 5.1);

- otherwise they occur in an implication; their scope, in such a case, is the implication in which they occur.

In the first case, we say that the variable is *flagged*. In the following, when we want to make explicit the fact that a variable X is flagged (when it is not clear from the context), it will be indicated with $\hat{X}$, while if we want to highlight that it is not flagged, it will be indicated with $\check{X}$.

Copy of a formula Since the $\mathcal{S}$CIFF syntax allows for abducibles with both existentially and universally quantified variables, the classical concept of renaming of a formula should be extended. Intuitively, universally quantified variables are renamed, in a sense, doubling the original formula, while existentially quantified variables are not. Let us call this operation *copy* of the formula.

When making a copy of a formula, we keep into account the scope of the variables it contains by means of their flagging status, as follows.

Definition 5.3 *Given a formula F, we call* copy *of F a formula F' where the universally quantified variables and the non flagged variables are renamed. We write*

$$F' = copy(F).$$

For example,

$$\exists_{\hat{Y}} \forall_{\hat{X}'>50} \forall_{\check{Z}'} \mathbf{E}(p(\hat{Y})) \wedge \mathbf{EN}(q(\hat{X}', \hat{Y})) \wedge [\mathbf{EN}(r(\hat{Y}, \check{Z}')) \rightarrow \exists_{\check{K}'} \mathbf{E}(p(\check{K}'))]$$

is a copy of the formula:

$$\exists_{\hat{Y}} \forall_{\hat{X}>50} \forall_{\check{Z}} \mathbf{E}(p(\hat{Y})) \wedge \mathbf{EN}(q(\hat{X}, \hat{Y})) \wedge [\mathbf{EN}(r(\hat{Y}, \check{Z})) \rightarrow \exists_{\check{K}} \mathbf{E}(p(\check{K}))]$$

Notice that, by Definition 5.3, if F contains only flagged existentially quantified variables, then $copy(F) \equiv F$ (so, for instance, the selected literal of SLD resolution would not be renamed, as in SLD resolution), while a universally quantified formula would be renamed (for instance, a clause would be renamed, as in SLD resolution).

Intuitively, by copying a formula we obtain a new fresh copy (unrelated to previous ones) of universally quantified variables and non flagged variables.

5.3 Transitions

The transitions are based on those of the IFF proof procedure, enlarged with those of CLP [60], and with specific transitions accommodating the concepts of fulfilment, dynamically growing history and consistency of the set of expectations with respect to the given definitions (Defs. 4.3, 4.4 and 4.2).

5.3.1 IFF-like transitions

The IFF proof-procedure The IFF is based on rewriting. It 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 its *inference rules*. By such rules, each node is always translated into a (disjunction of) conjunctions of atoms and implications; e.g., it can look like:

$$\begin{array}{ll} & (A_1 \wedge A_2 \wedge [A_3 \leftarrow B_1 \wedge B_2] \wedge [A_4 \leftarrow B_3 \wedge B_4]) \\ \vee & (A_i \wedge A_j \wedge A_k \wedge [A_z \leftarrow B_y] \wedge [false \leftarrow B_5]) \end{array}$$

The atoms have a similar meaning to those in the resolvent in LP, while the implications are (partially solved) integrity constraints.

Given a formula, its variables' quantification is defined by the following rules:

- *if* a variable is in the initial query, then it is *free*;
- *else if* it occurs in an atom, it is existentially quantified;

- *else* (it occurs only in implications) it is universally quantified.

A negated atom *not* A is rewritten as $false \leftarrow A$. Notice that this does not change the existential quantification of the atom because of the *allowedness condition.* A variable can occur in a negated atom only if it also occurs in a positive atom. A variable is universally quantified only if it occurs only in implications. Thus, if an implication $false \leftarrow A$ was generated by the transformation of a negated atom *not* A, the variables in A necessarily occur also in a positive atom, and must be considered existentially quantified.

The inference rules which IFF is based on are:

Unfolding: replaces resolution;

Propagation: propagates ICs;

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

Case analysis: if the body of an IC contains $X = t$, case analysis nondeterministically tries $X = t$ or $X \neq t$,

Factoring: tries to reuse a previously made hypothesis;

Rewrite rules for equality: use the inferences in the Clark Equality Theory;

Logical simplifications: try to simplify a formula through equivalences like $A \wedge false \leftrightarrow false$, $[A \leftarrow \mathit{true}] \leftrightarrow \mathit{true}$, etc.

In the following, we will show how these IFF transitions are adapted for the purposes of $\mathcal{S}$CIFF.

Unfolding Is adapted from the IFF proof-procedure.

Let L_i be the selected literal in the resolvent $R_k = L_1, \ldots, L_r$. Suppose that L_i is a predicate defined in the KB of the social specification. Unfolding generates a child node for each of the definitions of L_i; in each node, L_i is replaced with its definition.

More formally, if $H_1 \leftarrow B_1$, ..., $H_n \leftarrow B_n$ are the clauses in the KB such that $H_1, \ldots, H_n$ unify with L_i, unfolding generates n nodes. In the j-th node:

- first, a copy with fresh new variables of the clause is obtained $H'_j \leftarrow B'_j = \mathit{copy}(H_j \leftarrow B_j)$;
- then, all the variables in B'_j (that do not occur in the head) are flagged;

- the constraints of unification are added to the constraint store $CS_{k+1} = CS_k \cup \{H'_j = L_i\}$ (where $H'_j = L_i$ is a shorthand for the conjunction of equations between corresponding arguments of H'_j and L_i);
- B'_j is substituted to L_i in the new resolvent, i.e.,
  $R_{k+1} = L_1, \ldots, L_{i-1}, B'_j, L_{i+1}, \ldots, L_r$.

Moreover, as in the IFF proof procedure, unfolding is also applied to a defined atom in the body of an implication. In this case, only one child node is generated, which contains a new implication for each definition of the atom.

Formally, if

$$PSIC_k = \{Atom, BodyIC \rightarrow HeadIC\} \cup PSIC',$$

unfolding replaces *Atom* with all its definitions; i.e., if the clauses $H_1 \leftarrow B_1$, …, $H_n \leftarrow B_n$ belong to the KB and $H_1, \ldots, H_n$ unify with *Atom*,

$$\begin{aligned} PSIC_{k+1} = \ & \{B_1, BodyIC^1 \rightarrow HeadIC^1, \\ & \ldots, \\ & B_n, BodyIC^n \rightarrow HeadIC^n\} \cup PSIC' \end{aligned}$$

$$CS_{k+1} = CS_k \cup \{B_1 = Atom, \ldots, B_n = Atom\}$$

where, for all i, $BodyIC^i \rightarrow HeadIC^i$ is a copy of $BodyIC \rightarrow HeadIC$.

Abduction Since the $\mathcal{S}$CIFF proof procedure (differently from the IFF) keeps the set of abducibles separate from the resolvent, a transition has been introduced for abduction which, intuitively, moves an abducible from the resolvent to the set of abduced atoms.

More precisely:

- if $R_k = L_1, \ldots, L_r$, and the selected literal L_i is of type **E**, **EN**, ¬**E**, or ¬**EN**, then $R_{k+1} = L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_r$ and $\Delta P_{k+1} \equiv \Delta P_k \cup \{L_i\}$.
- otherwise, if the selected literal L_i is an abducible then $R_{k+1} = L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_r$ and $\Delta A_{k+1} \equiv \Delta A_k \cup \{L_i\}$.

Propagation Let $L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j$ be an implication belonging to the set $PSIC_k$, and let A be a literal that unifies with L_i in the body, such that A is

- either an event belonging to $\mathbf{HAP}_k$ (in which case A is an **H** event),
- or an abducible belonging to Δ_k,

then, *Propagation* produces a new node N_{k+1}:

- $PSIC_{k+1} = PSIC_k \cup \{A' = L'_i, L'_1, \dots, L'_{i-1}, L'_{i+1} \dots, L'_n \rightarrow H'_1 \vee \dots \vee H'_j\}$,

where $(L'_1, \dots, L'_n \rightarrow H'_1 \vee \dots \vee H'_j) = \mathit{copy}(L_1, \dots, L_n \rightarrow H_1 \vee \dots \vee H_j)$, and $A' = \mathit{copy}(A)$.

The equality in the body of the implication will be handled by transition *case analysis.*

Splitting Given a node with

- $R_k = L_1, \dots, L_{i-1}, (L_i \vee L_{i+1}), L_{i+2}, \dots, L_r$

splitting produces two nodes, N^1 and N^2 such that in node N^1

- $R^1_{k+1} = L_1, \dots, L_i, L_{i+2} \dots, L_r$

and in node N^2

- $R^2_{k+1} = L_1, \dots, L_{i-1}, , L_{i+1}, \dots, L_r$

In the $\mathcal{S}$CIFF proof procedure, disjunctions may appear also in the constraint store. Depending on the type of underlying Constraint Solver, clever reasoning can be possible. For instance, when using a CLP(FD) solver, constructive disjunction [99] or the cardinality operator [98] can be used to handle disjunctions of constraints.

If the adopted constraint solver does not provide such facilities, *splitting* can be applied also to disjunctions in the store. In such a case, given a node with

- $CS_k = C_1, \dots, C_{i-1}, (C_i \vee C_{i+1}), C_{i+2}, \dots, C_r$

splitting produces two nodes, N^1 and N^2 such that in node N^1

- $CS^1_{k+1} = C_1, \dots, C_{i-1}, C_i, C_{i+2}, \dots, C_r$

and in node N^2

- $CS^2_{k+1} = C_1, \dots, C_{i-1}, C_{i+1}, C_{i+2}, \dots, C_r$.

Case Analysis Given a node with an implication

$$PSIC_k = PSIC' \cup \{A = B, L_1, \dots, L_n \rightarrow H_1 \vee \dots \vee H_j\}$$

the node is replaced by two identical nodes, except for the following.
In Node 1 we hypothesise that the equality $A = B$ holds:

- $PSIC^1_{k+1} = PSIC' \cup \{L_1, \dots, L_n \rightarrow H_1 \vee \dots \vee H_j\}$
- $CS^1_{k+1} = CS_k \cup \{A = B\}$

In Node 2, we hypothesise the opposite:

- $PSIC^2_{k+1} = PSIC'$
- $CS^2_{k+1} = CS_k \cup \{A \neq B\}$

Since our proof procedure also needs to deal with constraints in the body, we also extend case analysis to the following situation: Given a node with an implication

$$PSIC_k = PSIC' \cup \{c, L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$$

where c is a constraint, if all the variables in $vars(c)$ are flagged, then case analysis can be applied. Notice that if a variable is flagged then it occurs in some abduced literal or in the resolvent.

If all the variables in $vars(c)$ are existentially quantified, then case analysis generates two nodes.
Node 1:

- $PSIC^1_{k+1} = PSIC' \cup \{L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$
- $CS^1_{k+1} = CS_k \cup \{c\}$

Node 2:

- $PSIC^2_{k+1} = PSIC'$
- $CS^2_{k+1} = CS_k \cup \{not\ c\}$

If all the variables in $vars(c)$ are universally quantified, then only one node is generated, in which the quantifier restriction $\forall_{X \in vars(c):c}$ is added to the constraint store, and

- $PSIC_{k+1} = PSIC' \cup \{L_1, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_j\}$

For example,

$$\forall_{\hat{Y}}\ PSIC_k = \{\hat{Y} > 0 \rightarrow \mathbf{EN}(q(\hat{Y}))\}$$

becomes

$$\forall_{\hat{Y}>0}\ PSIC_{k+1} = \{true \rightarrow \mathbf{EN}(q(\hat{Y}))\}$$

that will result in (see transitions *logical equivalence* and *abduction*):

$$\forall_{\hat{Y}>0}\ \Delta P_{k+i} = \{\mathbf{EN}(q(\hat{Y}))\}.$$

If $vars(c)$ contain both universally and existentially quantified variables, case analysis will not be applied. However, we make the hypothesis that the social specification is Constraint Allowed (Definition 3.15), so this case is forbidden by the syntax.

Factoring In the IFF proof procedure, transition factoring separates answers in which abducible atoms are merged from answers in which they are distinct. It is important for keeping the set of assumptions small (ideally, minimal). It generates two nodes: in one node two hypotheses unify, in the other one a constraint is imposed in order to avoid the unification of the hypotheses.

In the $\mathcal{S}$CIFF proof procedure, abducibles can contain universally quantified variables; it is not reasonable to unify atoms with universally quantified variables, because we would lose some of the information given by the abduced atoms.

Example 5.4 *Suppose that the set of expectations, in a node N_k, is:*

$$\forall \hat{X}, \hat{Y}\ \Delta P_k = \{\mathbf{EN}(p(1, \hat{X})), \mathbf{EN}(p(\hat{Y}, 2))\}.$$

By unifying the two hypotheses, we would obtain

$$\Delta P_{k+1} = \{\mathbf{EN}(p(1, 2))\}$$

which has a different meaning from the union of the two previous hypotheses: now, e.g., $p(1, 7)$ is no longer expected not to happen.

For this reason, we apply *factoring* only if the two atoms only contain existentially quantified variables. Notice that this coincides with the factoring transition of the IFF proof procedure.

Factoring can be applied in a node N_k, in which:

- $\Delta_k \supseteq \{A_1, A_2\}$

where A_1 and A_2 are (abducible) atoms in which all the variables are existentially quantified (and, of course, flagged). Factoring generates two children nodes, N^1 and N^2. In N^1:

- $CS^1_{k+1} = CS_k \cup \{A_1 = A_2\}$

and in N^2:

- $CS^1_{k+1} = CS_k \cup \{A_1 \neq A_2\}$

Equivalence Rewriting The equivalence rewriting operations are delegated to the constraint solver. Note that a constraint solver works on a constraint domain which has an associated interpretation. In addition, the constraint solver should handle the constraints among terms derived from unification. Therefore, beside the specific constraint propagation on the constraint domain, we assume that the constraint solver is equipped with further inference rules for coping with unification. In other words we will suppose that:

- the constraint theory contains rules for the equality constraint
- the constraint solver contains the same rules for equality that are in the IFF proof procedure, i.e., the function *infer*(CS) (see Sect. 5.3.5) performs the following substitutions in the constraint store:

   1. Replaces $f(t_1, \ldots, t_j) = f(s_1, \ldots, s_j)$ with $t_1 = s_1 \wedge \cdots \wedge t_j = s_j$.
   2. Replaces $f(t_1, \ldots, t_j) = g(s_1, \ldots, s_l)$ with $false$ whenever f and g are distinct or $j \neq l$.
   3. Replaces $t = t$ with $true$ for every term t.
   4. Replaces $X = t$ by $false$ whenever t is a term containing X.
   5. (a) Replaces $t = X$ with $X = t$ if X is a variable and t is not
      (b) Replaces $Y = X$ with $X = Y$ whenever X is a universally quantified variable and Y is not.
   6. (a) If $X = t \in CS_k$, applies the substitution X/t to the entire node.

Moreover, we also have to consider that our language is more expressive than that of the IFF proof-procedure, as we can abduce atoms with universally quantified variables. For this reason, we introduced *flagged* variables, and we need to deal with them in the theory of unification. We add the following rules:

5'. If $\check{X} = t \in CS_k$, $\check{X}$ is existentially quantified and not flagged, then replace $X = t$ with *false*.

5". If $X = \check{Y} \in CS_k$, $\check{Y}$ is an existentially quantified, not flagged variable, and X is existentially quantified or it has some (non trivially true) quantifier restrictions, then replace $X = \check{Y}$ with *false*.

Thus, an existentially quantified variable which is not flagged, unifies only with universally quantified variables that do not have quantifier restrictions.

Logical Equivalence The rule

$$\text{“}true \rightarrow A \text{ is equivalent to } A\text{”}$$

of the IFF proof procedure is translated as follows. If $PSIC_k = PSIC' \cup \{true \rightarrow A\}$, we generate a new node such that:

- $PSIC_{k+1} = PSIC'$
- $R_{k+1} = R_k, A'$

where A' is obtained from A by flagging all the variables that were not already flagged.

We also have the following rules, as in the IFF proof procedure:

$$\begin{array}{rcl} A \wedge false & \leftrightarrow & false \\ A \vee false & \leftrightarrow & A \\ A \wedge true & \leftrightarrow & A \\ A \vee true & \leftrightarrow & true \\ false \rightarrow A & \leftrightarrow & true \\ not\ A & \leftrightarrow & A \rightarrow false \\ not\ A, B \rightarrow C & \leftrightarrow & B \rightarrow A \vee C \end{array}$$

The last two rules are used only when applied to negation by failure of abducibles and definite predicates; they are not applied to *not* **H** literals (see transition *non-happening*, Sect. 5.3.2). For what concerns the negation of atoms **E** or **EN**, they are dealt with through *explicit negation*, i.e., for a negative literal ¬**E** we have an abducible (positive) literal *non***E** that cannot be true together with the corresponding literal **E** (see rules in Sect. 5.3.4), and symmetrically, the negative literal ¬**EN** is represented by an abducible *non***EN**.

5.3.2 Dynamically growing history

A set of transitions deals with a dynamically growing history **HAP**. The transitions are used to reason upon the happening (or non-happening) of events.

Closure In order to reason about non-happening of events, we adopt Closed World Assumption (CWA, [79]) on the set of currently happened events. Of course, this assumption is not acceptable if other events will happen in the future. For this reason, we non-deterministically assume that no other event will happen, i.e., we generate two child nodes. In the first we assume that no other events will happen, in the second that there will be other events. The *open/closed* attribute of the history (see Sect. 5.1) records if closed world is assumed on the happening of events.

Transition *Closure* is only applicable when no other transition is applicable. In other words, it is only applicable at the quiescence of the set of the other transitions.

Given a state where:

- closed($\mathbf{HAP}_k$) = *false*

in which no other transition is applicable, transition *Closure* produces two nodes. Node N^1 is the following:

- closed($\mathbf{HAP}_k$) = true

and node N^2 is identical to its father. In order to avoid infinite loops, transition *Closure* cannot be again applied to (descendants of) the node N^2 before a Happening transition has been applied.

Happening of Events The happening of events is handled by a transition *Happening*. This transition takes an event $\mathbf{H}(Event)$ from an external queue and puts it in the history **HAP**; the transition *Happening* is applicable only if an *Event* such that $\mathbf{H}(Event) \notin \mathbf{HAP}$ is in the external queue.

Given a state in which

- $\text{closed}(\mathbf{HAP}_k) = \mathit{false}$

the transition *Happening* produces a single successor node, where:

$$\mathbf{HAP}_{k+1} = \mathbf{HAP}_k \cup \{\mathbf{H}(Event)\}.$$

Otherwise, given a state in which

- $\text{closed}(\mathbf{HAP}_k) = \mathit{true}$

the transition *Happening* produces a single successor

$$\mathit{false}.$$

This means that a previously made CWA was wrong: we had assumed that no more events would have happened, but a new event has instead happened.

Non-happening The *Non-happening* transition can be considered an application of *constructive negation*. Constructive negation is a powerful inference that is particularly well suited in CLP [95].

Rule *non-happening* applies when the history is closed and a literal *not* **H** is in the body of a PSIC.

Given a node where:

- $PSIC_k = \{not\ \mathbf{H}(E_1), L_2, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_m\} \cup PSIC'$
- $\text{closed}(\mathbf{HAP}_k) = \mathit{true}$

non-happening produces a new node. Intuitively, we hypothesise that all the events matching with E_1 that are not in the history, do not happen at all.

The child node is produced as follows; we first give the intuition, then formalise the definition. We hypothesise that every event that would be able to match with E_1, and is not in the current history, will not happen. This can be seen as abducing an atom $non\mathbf{H}(E_1')$ where all the variables are substituted with universally quantified variables. We impose that the hypothesis holds in all cases except those already in the $\mathbf{HAP}_k$; we can state this by means of the quantifier restrictions, i.e., we impose that the hypothesis $non\mathbf{H}(E_1')$ does not unify with any of the happened events. This is equivalent to imposing a conjunction (for all the events in the history that match with E_1') of a disjunction (for all the variables appearing in E_1') of non unification restrictions (written $\neq$).

Let E_1' be a renaming of E_1 (i.e., all the variables in E_1 are substituted with fresh new variables). Let all the new variables in E_1' be universally quantified and flagged. For each variable $X_j \in vars(E_1)$, let $ren(X_j)$ be the corresponding, renamed variable in $vars(E_1')$. For all atoms $\mathbf{H}(E) \in \mathbf{HAP}_k$ that unify with $\mathbf{H}(E_1')$, we impose the quantifier restrictions on the variables in E_1' given by the following disjunction:

$$\bigwedge_{\substack{\mathbf{H}(E) \in \mathbf{HAP}_k \\ s.t. unifies(E, E_1')}} \left(\bigvee_{X_j \in vars(E_1)} ren(X_j) \neq t_j \right)$$

where t_j is the term in E corresponding to X_j in E_1.

The child node, $k+1$, is then defined by:

- $PSIC_{k+1} = \{E_1 = E_1', L_2, \ldots, L_n \rightarrow H_1 \vee \cdots \vee H_m\} \cup PSIC'$

Example 5.5

$$\neg\mathbf{H}(tell(\check{A}, \check{B}, propose(\check{I}))) \rightarrow \mathbf{EN}(tell(\check{B}, \check{A}, accept(\check{I}))) \tag{12}$$

Suppose that the history contains $\mathbf{H}(tell(yves, thomas, propose(nail)))$*. The condition in the body of the IC (12) is true, thus the IC triggers and the head is evaluated.*

$$\begin{tikzcd}
\mathbf{HAP}_k = \{\mathbf{H}(tell(yves, thomas, propose(nail)))\} \arrow[d, dash] \\
\forall_{\hat{A}' \neq yves \vee \hat{B}' \neq thomas \vee \hat{I}' \neq nail}\ \Delta P_{k+1} = \{\mathbf{EN}(tell(\hat{B}', \hat{A}', accept(\hat{I}')))\}
\end{tikzcd}$$

The last node states, correctly, that all events matching with $\mathbf{H}(tell(\hat{B}', \hat{A}', accept(\hat{I}')))$ *are forbidden, except* $\mathbf{H}(tell(thomas, yves, accept(nail)))$

5.3.3 Fulfilment and Violation

They are a group of transitions that nondeterministically try and match expectations with events. In general, these transitions generate two child nodes: in one we assume that one expectation and one event match, while in the other we assume they will not match.

Violation EN Given a node N with the following situation:

- $\Delta P_k = \Delta P' \cup \{\mathbf{EN}(E_1)\}$
- $\mathbf{HAP}_k = \mathbf{HAP'} \cup \{\mathbf{H}(E_2)\}$

violation **EN** produces two nodes N^1 and N^2, where N^1 is as follows:

- $\Delta V^1_{k+1} = \Delta V_k \cup \{\mathbf{EN}(E_1)\}$
- $CS^1_{k+1} = CS_k \cup \{E_1 = E_2\}$

and N^2 is as follows:

- $\Delta V^2_{k+1} = \Delta V_k$
- $CS^2_{k+1} = CS_k \cup \{E_1 \neq E_2\}$

Example 5.6 *Suppose that* $\mathbf{HAP}_k = \{\mathbf{H}(p(1,2))\}$ *and* $\exists\hat{X}\forall\hat{Y}\Delta P_k = \{\mathbf{EN}(p(\hat{X},\hat{Y}))\}$. Violation **EN** *will produce the two following nodes:*

$$\begin{tikzcd}
& \exists\hat{X}\forall\hat{Y}\Delta P_k = \{\mathbf{EN}(p(\hat{X},\hat{Y}))\}\ \mathbf{HAP}_k = \{\mathbf{H}(p(1,2))\} \arrow[dl, dash] \arrow[dr, dash] & \\
\begin{array}{c} CS^1_{k+1} = \{\hat{X} = 1 \wedge \hat{Y} = 2\} \\ \Delta V^1_{k+1} = \{\mathbf{EN}(p(1,2))\} \end{array} & & CS^2_{k+1} = \{\hat{X} \neq 1 \vee \hat{Y} \neq 2\} \arrow[d, dash] \\
& & CS_{k+2} = \{\hat{X} \neq 1\}
\end{tikzcd}$$

where the last simplification in the right branch is due to the rules of the constraint solver (see Sect. 5.3.5).

Fulfilment E Starting from a node N as follows:

- $\Delta P_k = \Delta P' \cup \{\mathbf{E}(Event_1)\}$
- $\mathbf{HAP}_k = \mathbf{HAP'} \cup \{\mathbf{H}(Event_2)\}$

Fulfilment **E** builds two nodes, N^1 and N^2, that are identical to their father except for the following.

In node N^1 we hypothesise that the expectation and the happened event unify:

- $\Delta P^1_{k+1} = \Delta P'$
- $\Delta F^1_{k+1} = \Delta F_k \cup \{\mathbf{E}(Event_1)\}$
- $CS^1_{k+1} = CS_k \cup \{Event_1 = Event_2\}$

In node N^2 we hypothesise that the two will not unify:

- $\Delta P^2_{k+1} = \Delta P_k$
- $\Delta F^2_{k+1} = \Delta F_k$
- $CS^2_{k+1} = CS_k \cup \{Event_1 \neq Event_2\}$

Violation E Violation of an **E** expectation can be proven only if there will not be an event matching the expectation. It is possible when we assume that no other event will happen; i.e., given a state where

- $\mathit{closed}(\mathbf{HAP}_k) = \mathit{true}$
- $\Delta P_k = \Delta P' \cup \{\mathbf{E}(\mathit{Event}_1)\}$

transition *Violation* **E** creates a successor node in which

- $\Delta V_{k+1} = \Delta V_k \cup \{\mathbf{E}(\mathit{Event}_1)\}$.
- $\Delta P_{k+1} = \Delta P'$

Note that, since the transition *closure* can be applied only if no other transition is applicable, the transition *Fulfilment* **E** has been already tried for each event in the history.

Another case in which violation of an **E** expectation can be proven is when its deadline has passed.

Given a node:

1. $\Delta P_k = \{\mathbf{E}(X, T)\} \cup \Delta P'$
2. $\mathbf{HAP}_k = \{\mathbf{H}(Y, T_c)\} \cup \mathbf{HAP'}$
3. $\forall \mathit{Event}_2 : \mathbf{H}(\mathit{Event}_2) \in \mathbf{HAP}$, Event_2 does not unify with Event_1
4. $\mathit{closed}(\mathbf{HAP}_k) = \mathit{false}$
5. $CS_k \models T < T_c$

transition Violation **E** is applicable and creates the following node:

- $\Delta P_{k+1} = \Delta P'$
- $\Delta V_{k+1} = \Delta V_k \cup \{\mathbf{E}(X, T)\}$.

Notice that one can avoid the expensive check of unification with all the elements in the history (condition 3) by choosing a preferred order of application of the transitions. By applying *Violation* **E** only if no other transition is applicable (except, possibly, for *closure*), the check (3) can be safely avoided.

Notice that this transition infers the current time from any happened event (condition 2); i.e., it infers that the current time cannot be less than the time of another happened event. In particular, there can be an event *current_time* that happens at every time tick. Of course, it is not necessary to perform the check for every event in the history; checking the last element in the history is enough.

A brief discussion is worth for the entailment $CS_k \models T < T_c$. The entailment of constraints from a constraint store is, in general, not easy to verify. In this particular case, however, we have that:

- the constraint $T < T_c$ is unary (T_c is always ground), thus a CLP(FD) solver is able to infer it very easily if the store contains only unary constraints (it is enough to check the maximum value in the domain of T);
- even if the store contains also non-unary constraints (thus a CLP(FD) solver performs, in general, incomplete propagation), the transition will not undermine soundness and completeness of the proof procedure. If the solver performs a powerful propagation, the violation will be detected very early. If the solver does not perform propagation at all, the violation will be detected later on, when the history gets closed.

Fulfilment EN Symmetrically to violation **E**, we can prove fulfilment of **EN** expectations. Given a state

- $\mathit{closed}(\mathbf{HAP}_k) = \mathit{true}$
- $\Delta P_k = \Delta P' \cup \{\mathbf{EN}(\mathit{Event}_1)\}$ does not unify with Event_1

transition *Fulfilment* **EN** creates a successor node in which

- $\Delta F_{k+1} = \Delta F_k \cup \{\mathbf{EN}(\mathit{Event}_1)\}$
- $\Delta P_{k+1} = \Delta P'$.

As for transition Violation **E**, the check of unification with all the atoms in the history can be avoided by establishing a preferred order of application of the transitions.

5.3.4 Consistency

E-Consistency In order to ensure **E**-consistency (see Def. 4.4) of the set of expectations, we impose the following integrity constraint:

$$\mathbf{E}(T) \wedge \mathbf{EN}(T) \rightarrow \mathit{false} \tag{13}$$

Example 5.7 *Suppose that $(\exists \hat{X})\mathbf{E}(p(\hat{X}))$ and $(\forall \hat{Y})\mathbf{EN}(p(\hat{Y}))$ have been abduced. By triggering the integrity constraint (13) we have that:*

$$\begin{array}{c}
\mathbf{E}(\check{T}) \wedge \mathbf{EN}(\check{T}) \rightarrow \mathit{false} \\
| \\
\mathbf{EN}(p(\hat{X})) \rightarrow \mathit{false} \\
| \\
\hat{X} = \hat{Y} \rightarrow \mathit{false} \\
| \\
\mathit{false}
\end{array}$$

Example 5.8 *Suppose that* $(\exists \hat{X})\mathbf{E}(p(\hat{X}))$ *and* $(\exists \hat{Y})\mathbf{EN}(p(\hat{Y}))$ *have been abduced. By triggering the integrity constraint (13) we have that:*

$$\begin{tikzcd}[column sep=small]
 & \mathbf{EN}(p(\hat{X})) \rightarrow \mathit{false} \arrow[d, dash] & \\
 & \mathbf{E}(\check{T}) \wedge \mathbf{EN}(\check{T}) \rightarrow \mathit{false} \arrow[d, dash] & \\
 & \hat{X} = \hat{Y} \rightarrow \mathit{false} \arrow[dl, dash] \arrow[dr, dash] & \\
\begin{array}{c} \hat{X} = \hat{Y} \\ \mathit{false} \end{array} & & \begin{array}{c} \hat{X} \neq \hat{Y} \\ \mathit{success} \end{array}
\end{tikzcd}$$

¬-Consistency In order to ensure ¬-consistency (see Def. 4.3) of the set of expectations, we impose the following integrity constraints:

$$\begin{array}{ccccc} \mathbf{E}(T) & \wedge & \neg\mathbf{E}(T) & \rightarrow & \mathit{false} \\ \mathbf{EN}(T) & \wedge & \neg\mathbf{EN}(T) & \rightarrow & \mathit{false} \end{array} \tag{14}$$

Example 5.9 *Suppose that* $\Delta P_k = \{\mathbf{E}(p(\hat{X})), \mathit{non}\mathbf{E}(p(1))\}$. *The integrity constraint (14) can trigger, and we have that:*

$$\begin{tikzcd}[column sep=small]
 & \mathbf{E}(\check{T}) \wedge \mathit{non}\mathbf{E}(\check{T}) \rightarrow \mathit{false} \arrow[d, dash] & \\
 & \mathbf{E}(p(1)) \rightarrow \mathit{false} \arrow[d, dash] & \\
 & \hat{X} = 1 \rightarrow \mathit{false} \arrow[dl, dash] \arrow[dr, dash] & \\
\begin{array}{c} \hat{X} = 1 \\ \mathit{false} \end{array} & & \begin{array}{c} \hat{X} \neq 1 \\ \mathit{success} \end{array}
\end{tikzcd}$$

5.3.5 CLP

The $\mathcal{S}$CIFF proof-procedure inherits the same transitions of CLP [60]. We suppose that the symbols $=$ and $\neq$ are in the constraint language and the theory behind them is, for equality, the one described in Sect. 5.3.1. Concerning $\neq$, we will again suppose that it is possible to syntactically distinguish the CLP-interpreted terms and atoms; the solver will perform some inference on the interpreted terms (typically, depending on the CLP sort, e.g., by deleting inconsistent values from domains in CLP(FD)), and will moreover contain the following rules, for uninterpreted terms:

1. Replaces $f(t_1, \ldots, t_j) \neq f(s_1, \ldots, s_j)$ with $t_1 \neq s_1 \vee \cdots \vee t_j \neq s_j$.
2. Replaces $f(t_1, \ldots, t_j) \neq g(s_1, \ldots, s_l)$ with *true* whenever f and g are distinct or $j \neq l$.

3. Replaces $t \neq t$ with $false$ for every term t.

4. Replaces $X \neq t$ by $true$ whenever t is a term containing X.

5. (a) Replaces $t \neq X$ with $X \neq t$ if X is a variable and t is not
   (b) Replaces $Y \neq X$ with $X \neq Y$ whenever X is a universally quantified variable and Y is not.

6. (a) Replace $A \neq B$ with $false$ if A is a universally quantified variable without quantifier restrictions (i.e., $QR(A) = \emptyset$)
   (b) If A is a universally quantified variable with quantifier restrictions $QR(A) = \{c_1(A), \ldots, c_d(A)\}$, and B is not universally quantified, replace $A \neq B$ with $\neg c_1(B) \vee \cdots \vee \neg c_d(B)$.[6]
   (c) If A and B are universally quantified, with quantifier restrictions $QR(A)$ and $QR(B)$ then
   - if $\neg QR(A) \cap \neg QR(B) = \emptyset$, replace $A \neq B$ with $true$.[7]
   - otherwise, replace $A \neq B$ with false.

Note that we do not introduce explicitly a rule for existentially quantified variables. In this case, we delegate to the specific solver (we do not make assumptions on its behavior). Some solvers can easily propagate constraints of this type. E.g., given $X \neq 1$ a Finite Domain solver can delete the value 1 from the domain of X. If the second term is not ground, the constraint is typically suspended (thus we do not have a transition). We will delay the $\neq$ constraint until it can be propagated by the given rules.

The constraint solver deals also with quantifier restrictions. If a quantifier restriction (due to unification) gets all the variables existentially quantified, then we replace it with the corresponding constraint. E.g., if in the tuple we have two variables $\hat{X}$ and $\hat{Y}$ quantified as follows:

$$\exists \hat{Y}, \forall_{\hat{X} \neq 1},$$

and variable $\hat{X}$ is unified with $\hat{Y}$, we obtain that $\exists \hat{Y}$, $\hat{Y} \neq 1$ (the quantifier restriction $\hat{X} \neq 1$ becomes a constraint on the variable $\hat{Y}$).

[6]Intuitively, A is universally quantified, thus it assumes every possible value except the ones forbidden by one of the c_i. Thus, the only way to satisfy this constraint is to impose that B assumes one of the values excluded for A.

[7]Intuitively, if the values taken by A have no intersection with the values taken by B, then $A \neq B$ is true.

Constrain Given a node with

- $R_k = L_1, \dots, L_r$

and the selected literal, L_i is a quantifier restriction, *constrain* produces a node with

- $R_{k+1} = L_1, \dots, L_{i-1}, L_{i+1}, \dots, L_r$
- $CS_{k+1} = CS_k \cup \{L_i\}$

Infer Given a node, the transition *Infer* modifies the constraint store by means of a function *infer*(CS). This function is typical of the adopted constraint sort. E.g., the function *infer* in a FD (Finite Domain) sort will typically compute (generalised) arc-consistency.

- $CS_{k+1} = \mathit{infer}(CS_k)$

Consistent Given a node, the transition *Consistent* will check the consistency of the constraint store (by means of a solver of the domain) and will generate a new node. The new node can either be a special node *fail* or a node identical to its father. Again, this transition is typical of the chosen constraint solver: in CLP(FD), for example, failures are discovered when a domain is empty.

If $consistent(CS_k)$ then

- $T_{k+1} = T_k$

If $\neg consistent(CS_k)$ then

- $T_{k+1} = fail$

6 Properties

We give the statements of soundness, completeness and termination of the $\mathcal{S}$CIFF proof procedure. We provide the sketch of the proofs, while the interested reader can find the full proofs on publicly available technical reports (for soundness and termination, see [58]; for completeness, see [57]).

6.1 Termination of $\mathcal{S}$CIFF

Termination is proven, as for SLD resolution [19], for *acyclic* knowledge bases and *bounded* goals and implications. The notion of acyclicity of an abductive logic program is an extension of the corresponding notion given for SLD resolution. Intuitively, for SLD resolution a level mapping must be defined, such that the head of each clause has a higher level than the body. For the IFF, since it contains integrity constraints that are propagated forward, the level mapping should also map atoms in the body of an IC to higher levels than the atoms in the head; moreover, this should also hold considering possible unfoldings of literals in the body of an IC [103]. Similar considerations hold also for $\mathcal{S}$CIFF. We extended the level mapping for considering also CLP constraints. For definitions of boundedness and acyclicity for the society Knowledge Bases, the reader can refer to [103].

Theorem 6.1 (Termination of $\mathcal{S}$CIFF) *Let $\mathcal{G}$ be a query to a society $\mathcal{S} = \langle KB, \mathcal{IC} \rangle$, where KB, $\mathcal{IC}$ and $\mathcal{G}$ are acyclic w.r.t. some level mapping, and $\mathcal{G}$ and all implications in $\mathcal{IC}$ are bounded w.r.t. the level-mapping. Then, every $\mathcal{S}$CIFF derivation for $\mathcal{G}$ for each instance of $\mathcal{G}$ is finite, assuming that happening is not applied.*

Moreover, under the following conditions:

- *the number of happened events is finite*
- *happening is applied only after the other transitions have reached quiescence*
- *non-happening has higher priority than other transitions*

$\mathcal{S}$CIFF terminates also with dynamically incoming events.

Proof.[sketch]

The proof of termination is given with a generic constraint solver (that must nevertheless contain the rules for equality and disequality). We first state some reasonable assumptions on the constraint solver, then we prove that under such assumptions, substituting equivalence rewriting with constraint solving does not undermine the proof of termination of the IFF.

We then adapted to our extended language the proof of termination of the IFF proof procedure [103]. Such a proof affirms that when the history is given initially (i.e., no happening transition is ever applied), $\mathcal{S}$CIFF terminates.

Finally, we showed that the proof can be extended to the dynamic case. We assume that the event happening rate is low, i.e., between two happening transitions the $\mathcal{S}$CIFF reaches the quiescence. We divide the derivation in two parts: an open and a closed part (before and after the application of closure transition). In the open part, between the happening of two events, $\mathcal{S}$CIFF is applied, and it terminates. Assuming that the number of happened events is finite, the open derivation terminates. Closure is applicable only once. After application of closure, non-happening transitions are applicable. Since the number of *not* **H** literals is finite, non-happening will be applied finitely many times. After that, static $\mathcal{S}$CIFF is applied and it terminates. □

6.2 Soundness of $\mathcal{S}$CIFF

The $\mathcal{S}$CIFF proof-procedure uses a constraint solver, so its soundness depends on the solver. We proved soundness for a limited solver, containing only the rules for equality and disequality given in the operational semantics.

Theorem 6.2 (Soundness of $\mathcal{S}$CIFF) *Given a society instance $\mathcal{S}_{\mathbf{HAP}^f}$, if*

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

for some $\mathbf{HAP}^i \subseteq \mathbf{HAP}^f$, with expectation answer (Δ, σ), then

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

Proof.[sketch] The proof of soundness relies on the proof of soundness of the IFF [55], and on a set of lemmas [58]. Such lemmas draw a correspondence between IFF and $\mathcal{S}$CIFF, in order to apply the soundness theorem of the IFF.

1. A first lemma maps a correspondence between how disequalities are handled in IFF and in $\mathcal{S}$CIFF.
2. A second lemma proves that if the set of abducibles of the final node of a derivation does not contain universally quantified variables, then no universally quantified abducible has been generated during the derivation.
3. Third, we proved that for each successful derivation starting with an initial history $\mathbf{HAP}^i$ and ending in a node containing a final history $\mathbf{HAP}^f$, there exists a successful derivation starting with $\mathbf{HAP}^f$ and ending in the same node of the previous derivation. Such a lemma allows us to prove soundness of the static version of the $\mathcal{S}$CIFF, and derive immediately soundness of the dynamic version.
4. We then defined the IFF-like rewritten program: the $\mathcal{S}$CIFF knowledge base can be translated into IFF syntax, by mapping universally quantified abducibles to constant symbols. To the IFF-like rewritten program, the theorem of soundness of the IFF is applicable. We proved that each derivation in which abduced literals do not contain universally quantified variables has a correspondent in an IFF derivation.

   Thus, each $\mathcal{S}$CIFF derivation that does not abduce literals with universally quantified variables is sound with respect to the semantics of the IFF. We proved then that soundness also holds with respect to the $\mathcal{S}$CIFF semantics (i.e., that the resulting set of expectations is also **E**- and $\neg$-consistent and fulfilled).

5. Finally, we extended the result to the derivations in which $\mathcal{S}$CIFF abduces literals containing universally quantified variables. Considering a derivation D, we build a derivation D' for a program that contains the literals in the final node of D as definition of a predicate in the KB. Derivation D' does not abduce universally quantified literals (and, thus, it is sound, as shown in step 4), and terminates in a node that is the same obtained by D.

This proves that $\mathcal{S}$CIFF is sound. $\square$

6.3 Completeness of $\mathcal{S}$CIFF

Completeness states that if goal G is achieved under the expectation set $\mathbf{EXP}$, then a successful derivation can be obtained for G, possibly computing a set $\mathbf{EXP}'$ of the expectations whose grounding (according to the expectation answer) is a subset of $\mathbf{EXP}$.

Theorem 6.3 *Given a society instance $\mathcal{S}_{\mathbf{HAP}}$, a (ground) goal G, for any ground set Δ such that $\mathcal{S}_{\mathbf{HAP}} \models_{\Delta} \mathcal{G}$ then $\exists \Delta'$ such that $\mathcal{S}_{\emptyset} \vdash_{\Delta'}^{\mathbf{HAP}} G$ with an expectation answer (Δ', σ) such that $\Delta'\sigma \subseteq \Delta$.*

Proof.[sketch] Completeness is currently proven for a limited set of programs, namely those without universally quantified variables in the abducibles, and without $\neg\mathbf{H}$ literals.

Similarly to the case of soundness, the proof is based on the IFF-like rewritten program, and on the lemma stating that equality rewriting is equivalent to constraint solving.

We first proved that for every IFF derivation there exists a $\mathcal{S}$CIFF derivation reaching the same node, starting from an initial node containing the final history $\mathbf{HAP}$. Together with soundness results of the IFF, this proves that

$$\mathcal{S}_{\mathbf{HAP}} \models_{\Delta} G \Longrightarrow \mathcal{S}_{\mathbf{HAP}} \vdash_{\Delta'}^{\mathbf{HAP}} G.$$

Extending to the dynamic case is trivial: given a derivation D starting from $\mathcal{S}_{\mathbf{HAP}}$, we can build a derivation starting from $\mathcal{S}_{\emptyset}$ by adding *happening* transitions before the first node of D. $\square$

7 Implementation

In this section, we briefly sketch the current implementation of $\mathcal{S}$CIFF, which consists of a program for SICStus Prolog [90] and, in particular, its *CHR* [54] library. Due lack of space, we do not introduce *CHR* here; the reader can refer to [54] for a complete introduction.

The most usual technique for implementing (abductive) proof procedures has probably been meta-interpretation, which lets the programmer adjust the built-in search strategy of Prolog to application-specific requirements in a compact (if computationally expensive) way. However, the common understanding of abducible and constraints suggested by Kowalski *et al.* [71] paved the way for some authors to implement abduction in the *Constraint Handling Rules* language [1, 59, 28], with advantages in execution time with respect to meta-interpretation. For the $\mathcal{S}$CIFF implementation, we followed the *CHR* approach: a *CHR*-based implementation offers, as a byproduct, the possibility of the seamless integration of a high-level implementation of constraint solvers, which $\mathcal{S}$CIFF needs in order to manage quantifier restrictions.

In the $\mathcal{S}$CIFF implementation, *CHR* constraints have been used to implement most of the data structures. In particular, events (**HAP**), abducibles (the ΔA, ΔP, ΔF, ΔV sets, see Sect. 5.1) and integrity constraints (the $PSIC$ set) are represented as *CHR* constraints. In this way, many transitions can be implemented as *CHR* rules. Handling of constraints is delegated to the CLP constraint solvers.

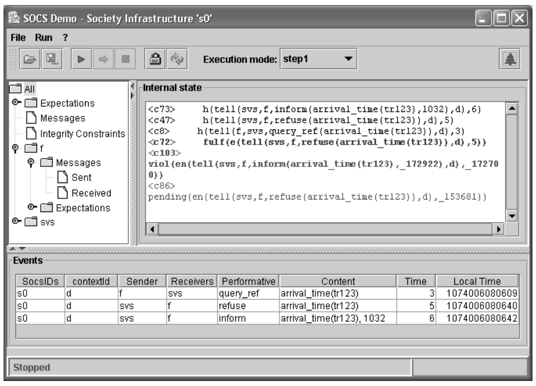
The $\mathcal{S}$CIFF operational semantics defines the proof tree, but delegates the search strategy in the tree to the implementation. The current $\mathcal{S}$CIFF implementation employs a depth-first search strategy of the proof tree. This choice enabled us to tailor the implementation upon the operational semantics of Prolog: in particular, the resolvent of the proof (see Sect. 5.1) is represented by the Prolog resolvent, and thus the Prolog stack is used directly for chronological backtracking.

The inputs to the $\mathcal{S}$CIFF implementation are those defining an instance of an abductive specification (see Sect. 4), i.e.:

- The Knowledge Base
- the set of ICs;
- the history **HAP**.

Success and failure of the implementation map directly the corresponding notions of $\mathcal{S}$CIFF. In particular, the implementation returns success when a state of goal achievement is found; instead, all the failure conditions, such as inconsistency (both with respect to **E**-consistency and ¬-consistency, see Defs. 4.4 and 4.3), inconsistent constraint store and violation generate a failure, possibly causing backtracking.

The current implementation of the $\mathcal{S}$CIFF proof-procedure is publicly available on the web (`lia.deis.unibo.it/research/sciff/`). It is at the core of the SOCS-SI tool [7], whose Graphical User Interface written in Java, shows the state of the proof (i.e., the components of the tuple in Eq. 11), the derivation tree (with violated nodes), the agents involved in the interaction (Fig. 1). SOCS-SI is also equipped with interfaces that make it compatible with a number of agent platforms, including JADE [23], ProSOCS [24], and tuProlog [40].



```
<c73>     h(tell(svs,f,inform(arrival_time(tr123),1032),d),6)
<c47>     h(tell(svs,f,refuse(arrival_time(tr123)),d),5)
<c8>     h(tell(f,svs,query_ref(arrival_time(tr123)),d),3)
<c72>     fulf(e(tell(svs,f,refuse(arrival_time(tr123)),d),5))
<c103>
viol(en(tell(svs,f,inform(arrival_time(tr123),_172922),d),_17270
0))
<c86>
pending(en(tell(svs,f,refuse(arrival_time(tr123)),d),_153681))
```

SocsIDs	contextId	Sender	Receivers	Performative	Content	Time	Local Time
s0	d	f	svs	query_ref	arrival_time(tr123)	3	1074006080609
s0	d	svs	f	refuse	arrival_time(tr123)	5	1074006080640
s0	d	svs	f	inform	arrival_time(tr123), 1032	6	1074006080642

Figure 1. *Screenshot of the GUI of the application*

8 Sample Applications

In this section, we demonstrate the features of the $\mathcal{S}$CIFF framework by examples taken from different domains. Our first example is the Abductive Event Calculus (AEC) [46]. We use AEC to show how $\mathcal{S}$CIFF operates in a classical application of abductive proof-procedures that has been used by many other authors before [88, 66, 45]. Then we show an application of $\mathcal{S}$CIFF in the MAS domain. To this end, we take a protocol defined by the Foundation for Intelligent Physical Agents (FIPA),[8] one of the main agent standardization bodies, and demonstrate the usage of $\mathcal{S}$CIFF for the specification of well-known protocols, and for a verification of compliance that can be done at agent execution time. We show cases of compliance, violation by uttering a communicative act forbidden by the protocol, and violation by not respecting a deadline. We conclude by showing a case taken from the domain of normative systems. In this example, a number of norms dictate the protocol to be followed in order for a customer to have a telephone line installed. The purpose of this example is to demonstrate the interaction between **E**, **EN**, and **H** predicates, CLP constraints and predicates defined in the KB. The protocol is taken from real life, and the authors have reasons to believe that it would be difficult to specify it using other protocol definition languages, such as FSM, Petri nets, or AUML interaction diagrams. The declarative nature of $\mathcal{S}$CIFF makes such an intricate protocol understandable, modular in its representation, and verifiable, as our examples of fulfilment and violation show.

[8]`http://www.fipa.org/`.

8.1 Planning with the Abductive Event Calculus

AEC [46] is a classical application of abductive proof-procedures, and it can be used for planning in agent systems [67, 74]. In order to understand it we must give some background. The Event Calculus (EC) [70, 87] is a framework to reason about properties (called *fluents*) that may hold in a system inside time intervals. The EC consists of four ingredients:

1. A set of known causal relations, stating which events initiate or terminate the validity of a fluent. For example, in the description of a robot in the block world we can imagine the fluents $ontable(X)$ (block X is on the table) and $holding(X)$ (the robot holds in its hand the block X). Rules could state that if the robot is holding the block, then the action of putting a block on the table initiates the fluent "block X is on the table":

   $$initiates(putdown(X), ontable(X), T) \leftarrow holdsat(holding(X), T).$$

   On the contrary, the fluent "block X is on the table" is terminated by the action of picking X up. Therefore the definition:

   $$terminates(pickup(X), ontable(X)).$$

2. The initial situation provided by the *initially* predicate. For example, the robot is initially holding block number 1:

   $$initially(holding(1)).$$

3. A narrative of happened events; for example

   $$\begin{array}{l} happens(putdown(1), 3). \\ happens(pickup(1), 5). \end{array}$$

4. The general theory of EC, defined as a set of domain-independent rules which state that a fluent holds at a given time if it was either initially true, or if it has become true after an event, and it has not ever since been *clipped*, i.e., its truth has not been terminated in the meanwhile.

   $$\begin{array}{rl} holdsat(F,T) \leftarrow & initially(F), not\ clipped(0,F,T). \\ holdsat(F,T) \leftarrow & happens(E,T_1), initiates(E,F), \\ & not\ clipped(T_1,F,T). \\ clipped(T_1,F,T_2) \leftarrow & happens(E,T), T_1 < T < T_2, terminates(E,F). \end{array} \tag{15}$$

Based on this theory, by deduction one can infer for instance that the fluent $ontable(1)$ is true at time 4 and it is false at times 2 and 10.

Building on this result, Eshghi [46] pointed that planning problems can be solved by interpreting the event calculus in abduction. The user states the initial situation (through the *initially* predicate) and a goal, typically requiring the validity of some fluents in the final situation. The narrative of events is no longer given, but is considered as a set of actions that should be performed in order to obtain the goal; i.e., *happens* atoms are abducible. In the example, if the goal was

$$holdsat(ontable(1), 10) \tag{16}$$

the Abductive Event Calculus (AEC) would reply that in order to obtain the goal, the *putdown* action should be performed before time 10.

Many other authors address planning through abduction. Some of them consider the precedence relationship between events to be abducible [88]. Others use a discrete representation of time and thus rely on efficient constraint solvers [66, 45].

The $\mathcal{S}$CIFF framework easily accommodates the AEC. Additionally, it keeps happened events separate from expected events, which we consider to be an improvement, in terms of representation: what is planned or supposed to happen, not necessarily coincides with what is actually happening. An agent could plan to perform an action, but the action might fail. In the blocks world example, a block could slip, thus making a *pickup* action unsuccessful. The robot expected to pickup the block, but the actual action did not match. This unexpected event generates the need for alternative possible course of events, such as a retrial, or a totally different plan.

Therefore, in this implementation of AEC via $\mathcal{S}$CIFF, plans are defined through **E** predicates rather than **H** events. If the agent wants to get to a goal state, it should perform plan for and execute actions, which makes such actions *expected*: by no means, the actions in the plan are already *happened* at planning time.

Positive **E** expectations state actions that should be taken in order for the plan to be effective. Negative **EN** expectations (which do not exist in previous abductive event calculus proposals) inform about those actions that should not be executed in order for the plan to be successful.

The $\mathcal{S}$CIFF implementation of the AEC theory is shown in Tab. 8.1.

The rules in the KB are direct translation of the first two in Eq. (15). We introduce a new abducible predicate, **unclipped**, to represent the $not(clipped)$ literals found in the classical event calculus. In order for the plan to be successful, the fluent should not be clipped in the given time interval. This is ensured by the application of the integrity constraint in Tab. 8.1, which imposes that every event that would terminate the validity of the fluent is expected not to happen (in the given time interval). This mechanism lets us exploit better the underlying constraint solver, which is tailored to reason about positive and negative expectations. Moreover, the

Table 8.1 Abductive Event Calculus theory in $\mathcal{S}$CIFF.

$\mathcal{IC}$:

$$\mathbf{unclipped}(T_1, F, T_2), \mathit{terminates}(A, F) \rightarrow \mathbf{EN}(A, T), T_1 < T < T_2.$$

KB :

$$\mathit{holdsat}(F, T) \leftarrow \mathit{initially}(F), \mathbf{unclipped}(0, F, T).$$
$$\mathit{holdsat}(F, T) \leftarrow \mathbf{E}(A, T_1), 0 < T_1 < T, \mathit{initiates}(A, F, T_1), \mathbf{unclipped}(T1, F, T).$$

planner will explicitly provide, in the form of negative (**EN**) expectations, which actions should be avoided in order not to endanger the execution of the plan.

In the blocks world example $\mathcal{S}$CIFF provides

$$\mathbf{E}(\mathit{putdown}(1), T_1), \mathbf{EN}(\mathit{pickup}(1), T_2), T_1 < T_2 < 10$$

which we read as:

$$\exists_{T_1} \forall_{T_2 : T_1 < T_2 < 10}\ \mathbf{E}(\mathit{putdown}(1), T_1), \mathbf{EN}(\mathit{pickup}(1), T_2),$$

i.e., in order to achieve the goal $\mathit{holdsat}(\mathit{ontable}(1), 10)$, expressed in Eq. (16), the robot should drop block 1, and avoid picking it up before time 10. The times of these planned actions are given in terms of domains (intervals), thanks to the underlying constraint solver. In order to obtain punctual times, one can anytime resort to grounding. This very useful feature is present in $\mathcal{S}$CIFF as well as in other frameworks of literature, such as $ACLP$ [65] and $\mathcal{A}$-System [66] (for a discussion, see Sect. 9).

8.2 Specifying and verifying agent interaction

In this section, we demonstrate the the usage of $\mathcal{S}$CIFF in MAS domains, via a simple agent interaction protocol.[9] We show first the $\mathcal{S}$CIFF-based implementation of the FIPA Request Interaction Protocol, then we propose three sample dialogue instances and we discuss the outcome of $\mathcal{S}$CIFF when they are confronted with the specified protocol.

8.2.1 The FIPA Request Interaction Protocol

The FIPA Request Interaction Protocol [51], depicted in Fig. 2 allows one agent to request another to perform some action. The normal protocol flow is composed of the following steps:

[9]A collection of sample protocols and protocol runs can be found in [2, 93].

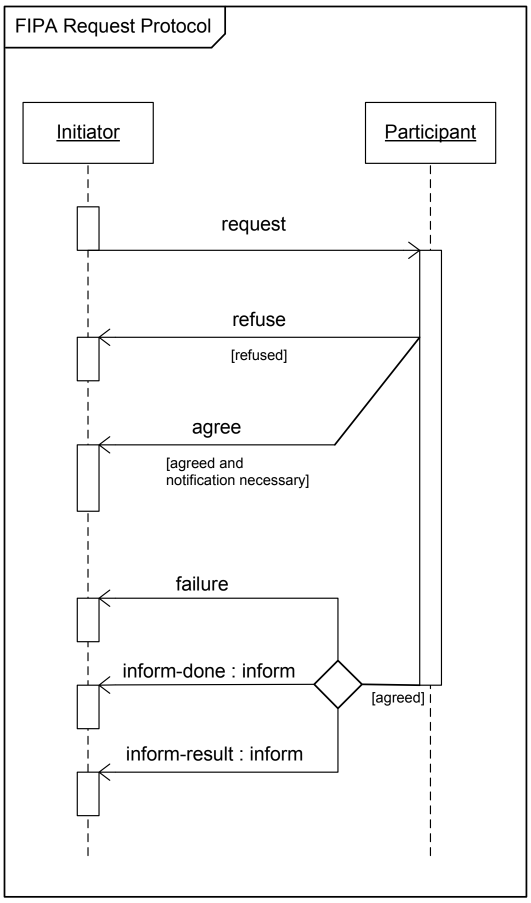


Figure 2. *FIPA Request Interaction Protocol.*

1. The *Initiator* agent issues a *request* to a *Participant* agent to perform an action P.

2. *Participant* can either

   - *refuse* to perform P, in which case the protocol ends; or
   - *accept* to perform P; in this case, after performing the action,

3. *Participant* will issue to *Initiator* one of the following:

   - *inform_done*(P), which simply tells *Initiator* that P has been performed;
   - *inform_result*(P,R), which also contains, in R, some information about the result of performing the action;
   - *failure*(P), which reports a failure.

Table 8.2 $\mathcal{IC}$ and KB for the FIPA Request interaction protocol.

$$
\begin{array}{ll}
\mathcal{IC}: & \\
& \mathbf{H}(tell(Initiator, Participant, request(P), D), T) \\
\rightarrow & \mathbf{E}(tell(Participant, Initiator, agree(P), D), T1) \ \wedge \ T \ < \ T1 \\
\vee & \mathbf{E}(tell(Participant, Initiator, refuse(P), D), T1) \ \wedge \ T \ < \ T1. \\
& \\
& \mathbf{H}(tell(Participant, Initiator, agree(P), D), T1) \\
\rightarrow & \mathbf{EN}(tell(Participant, Initiator, refuse(P), D), T2). \\
& \\
& \mathbf{H}(tell(Initiator, Participant, request(P), D), T) \ \wedge \\
& \mathbf{H}(tell(Participant, Initiator, agree(P), D), T1) \ \wedge \ T \ < \ T1 \\
\rightarrow & \mathbf{E}(tell(Participant, Initiator, failure(P), D), T2) \ \wedge \ T1 \ < \ T2 \wedge \\
& t\_failure(Td) \ \wedge \ T2 \ < \ T1 \ + \ Td \\
\vee & \mathbf{E}(tell(Participant, Initiator, inform\_done(P), D), T2) \ \wedge \ T1 \ < \ T2 \wedge \\
& t\_done(Td) \ \wedge \ T2 \ < \ T1 \ + \ Td \\
\vee & \mathbf{E}(tell(Participant, Initiator, inform\_result(P, R), D), T2) \ \wedge \ T1 \ < \ T2 \wedge \\
& t\_result(Td) \ \wedge \ T2 \ < \ T1 \ + \ Td. \\
& \\
& \mathbf{H}(tell(Participant, Initiator, failure(P), D), T) \\
\rightarrow & \mathbf{EN}(tell(Participant, Initiator, inform\_done(P), D), T1) \ \wedge \\
& \mathbf{EN}(tell(Participant, Initiator, inform\_result(P, R), D), T2). \\
& \\
& \mathbf{H}(tell(Participant, Initiator, inform\_done(P), D), T) \\
\rightarrow & \mathbf{EN}(tell(Participant, Initiator, failure(P), D), T1) \ \wedge \\
& \mathbf{EN}(tell(Participant, Initiator, inform\_result(P, R), D), T2). \\
& \\
KB: & \\
& t\_failure(10). \\
& t\_done(20). \\
& t\_result(50).
\end{array}
$$

The $\mathcal{S}$CIFF-based specification of the protocol is shown in Tab. 8.2.

The first IC imposes to a *Participant* who has received a *request* to perform an action, to reply with either *agree* or *refuse*. The second IC imposes mutual exclusion between *agree* and *refuse*: if *Participant* has *agree*d, it cannot *refuse* at any time. The third IC imposes *request* and *agree* to be followed by one among *inform_done*, *inform_result*, and *failure*. In order to make this protocol more realistic, we have introduced some time constraints that are not present in its original FIPA specifications. The messages *inform_done*, *inform_result*, and *failure* should be uttered within a certain deadline: the deadline is determined by the instant in which the *Participant* agreed, plus an interval defined in the knowledge base. The last two ICs impose mutual exclusion among the three possible utterances.

The knowledge base in this case contains only predicates defining timeout values.

8.2.2 A dialogue instance satisfying the protocol

First of all, we discuss a dialogue instance that fulfils the protocol specifications:

$\mathbf{H}(\ tell(\ a,\ b,\ request(\ check\_balance),\ r1),\ 3).$
$\mathbf{H}(\ tell(\ b,\ a,\ agree(\ check\_balance),\ r1),\ 6).$
$\mathbf{H}(\ tell(\ b,\ a,\ inform\_result(\ check\_balance,\ balance(300, usd)),\ r1),\ 7).$

In this dialogue instance, a first agent *a* requests to check a bank account to an agent *b* (let us imagine that *a* represents a customer, *b* a bank). Agent *b* agrees to provide the information, and later it communicates the balance.

The $\mathcal{S}$CIFF proof procedure elaborates several alternative sets of hypotheses about which events should be expected and which not. When the *Closure* (see Sect. 5.3.2) transition is applied, however, only one such set is confirmed, and the expectation about the *inform_result* is fulfilled (as shown in Fig. 3).[10]

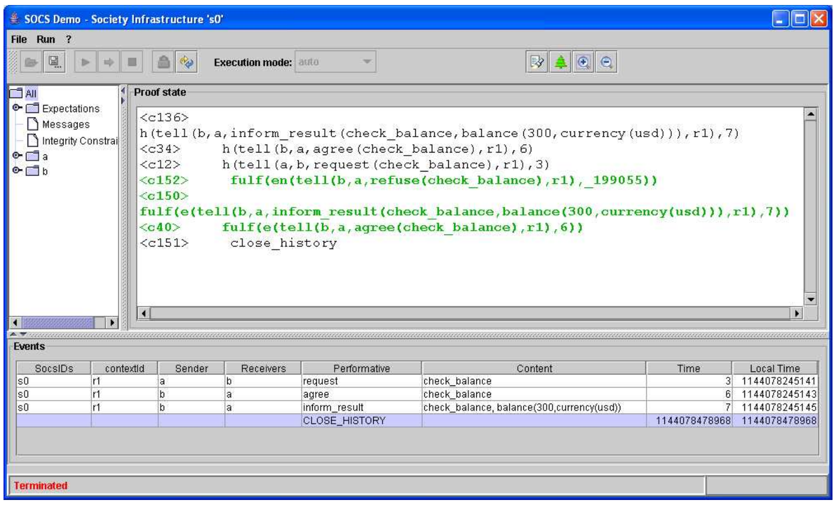
8.2.3 Uttering a message that is not permitted

In this second example, we discuss a dialogue instance that violates the protocol. In particular, agent *b*, after agreeing to provide the information requested, later "contradicts itself" by uttering a *refuse* message.

$\mathbf{H}(\ tell(\ a,\ b,\ request(\ check\_balance),\ r1),\ 3).$
$\mathbf{H}(\ tell(\ b,\ a,\ agree(\ check\_balance),\ r1),\ 6).$
$\mathbf{H}(\ tell(\ b,\ a,\ refuse(\ check\_balance),\ r1),\ 8).$

This dialogue instance clearly violates the second integrity constraint shown in Tab. 8.2. In Fig. 4 it is possible to see how this violation is detected by the $\mathcal{S}$CIFF

[10]In SOCS-SI, logical variables are preceded by underscore, like _65615. SOCS-SIalso shows the internal identifier of CHR constraints among angles, like ⟨c74⟩.



```
<c136>
h(tell(b,a,inform_result(check_balance,balance(300,currency(usd))),r1),7)
<c34>      h(tell(b,a,agree(check_balance),r1),6)
<c12>      h(tell(a,b,request(check_balance),r1),3)
<c152>      fulf(en(tell(b,a,refuse(check_balance),r1),_199055))
<c150>
fulf(e(tell(b,a,inform_result(check_balance,balance(300,currency(usd))),r1),7))
<c40>      fulf(e(tell(b,a,agree(check_balance),r1),6))
<c151>      close_history
```

SocsIDs	contextId	Sender	Receivers	Performative	Content	Time	Local Time
s0	r1	a	b	request	check_balance	3	1144078245141
s0	r1	b	a	agree	check_balance	6	1144078245143
s0	r1	b	a	inform_result	check_balance, balance(300,currency(usd))	7	1144078245145
				CLOSE_HISTORY		1144078478968	1144078478968

Figure 3. *A dialogue instance satisfying the protocol*

proof procedure. In particular, the violation is shown by functor *viol* (corresponding to an item in the ΔV set of the $\mathcal{S}$CIFF operational semantics).

8.2.4 A participant does not respect a deadline

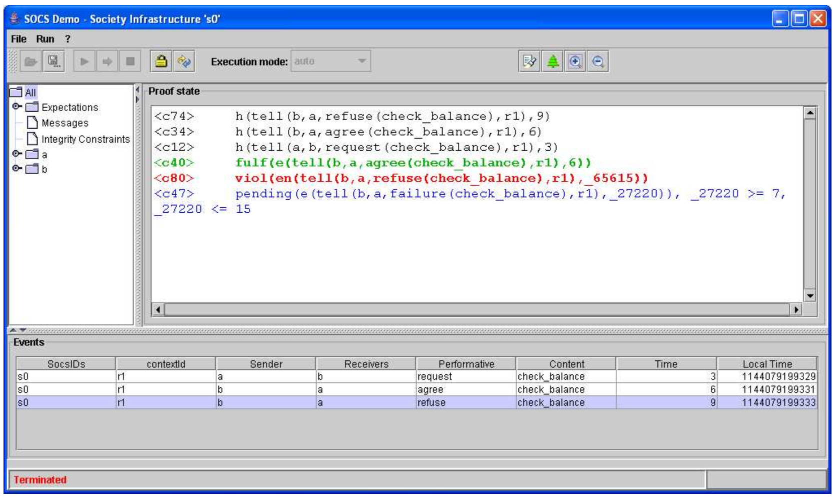
In this example, *b* fulfils the protocol specifications as for what concerns the messages and their content. However, the *inform_result* message is sent at time 58, whereas the *agree* message has been sent at time 6. Hence, the deadline specified in the third integrity constraint of Tab. 8.2 is not respected.

$$\begin{array}{l}
\mathbf{H}(\ tell(\ a,\ b,\ request(\ check\_balance),\ r1),\ 3).\\
\mathbf{H}(\ tell(\ b,\ a,\ agree(\ check\_balance),\ r1),\ 6).\\
\mathbf{H}(\ tell(\ b,\ a,\ inform\_result(\ check\_balance,\ balance(300, usd)),\ r1),\ 58).
\end{array}$$

Fig. 5 shows how this situation is detected ($gt\_current\_time(\ldots)$ implements the transition “Violation **E**”, see Sect. 5.3.3).

8.3 Reasoning with norms and events

Many specifications of norms, such as those dictating the correct behaviour of agents in institutionalized transactions, are not so different from the agent interaction protocols of which we have given one example above. Let the reader not be mislead: by no means do we intend to compete with frameworks for the specification of normative systems, as this is not the purpose of $\mathcal{S}$CIFF. However, we argue that the $\mathcal{S}$CIFF framework can be used to specify normative elements of MAS, and that a



```
<c74>     h(tell(b,a,refuse(check_balance),r1),9)
<c34>     h(tell(b,a,agree(check_balance),r1),6)
<c12>     h(tell(a,b,request(check_balance),r1),3)
<c40>     fulf(e(tell(b,a,agree(check_balance),r1),6))
<c80>     viol(en(tell(b,a,refuse(check_balance),r1),_65615))
<c47>     pending(e(tell(b,a,failure(check_balance),r1),_27220)), _27220 >= 7,
_27220 <= 15
```

SocsIDs	contextId	Sender	Receivers	Performative	Content	Time	Local Time
s0	r1	a	b	request	check_balance	3	1144079199329
s0	r1	b	a	agree	check_balance	6	1144079199331
s0	r1	b	a	refuse	check_balance	9	1144079199333

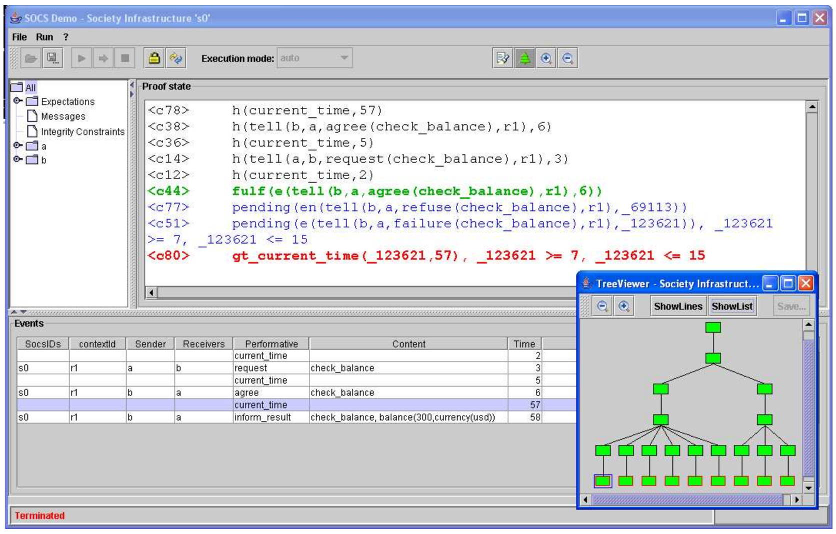
Figure 4. *Uttering a message that is not permitted*

mapping can be cast between the $\mathcal{S}$CIFF abductive framework and classical deontic logic. A proposal in this sense is shown in [11].

In this section, we intend to demonstrate how to specify, inside the $\mathcal{S}$CIFF framework, agent interaction protocols that may result too intricate for a representation based on other formalisms, such as FSM, coloured Petri nets, and the AUML diagrams seen above. The example we give is a simplified version of a real life situation, describing the activation of a telephone line (carrier) by a customer. We consider the clauses of the contract a user must sign as the building blocks of an interaction protocol, which makes use of expressive combinations of **E**, **EN**, and **H** predicates, CLP constraints and predicates defined in the KB. With $\mathcal{S}$CIFF we give a faithful representation of such a protocol, which makes it understandable, modular, and verifiable. Despite all effort put by the telephone company into making things as obscure as possible, at any time we (as customers) will be able to detect, via $\mathcal{S}$CIFF, whether the telephone company (*telco* in the following) has the right to interrupt the service or to request a payment from us, and whether we have the right to complain with *telco*, and not to pay part of the bill. Similarly, *telco* will receive indications about when to send requests for payment, or when (not) to activate or (not) to de-activate the carrier.

8.3.1 Description of the contract

The procedures that regulate the concession of a carrier to a customer are contained in a contract, that the parties (*telco* and the customer) agree upon. The contract is composed of several parts, stating what to do when the customer request a new



```
<c78>     h(current_time,57)
<c38>     h(tell(b,a,agree(check_balance),r1),6)
<c36>     h(current_time,5)
<c14>     h(tell(a,b,request(check_balance),r1),3)
<c12>     h(current_time,2)
<c44>     fulf(e(tell(b,a,agree(check_balance),r1),6))
<c77>     pending(en(tell(b,a,refuse(check_balance),r1),_69113))
<c51>     pending(e(tell(b,a,failure(check_balance),r1),_123621)), _123621
>= 7, _123621 <= 15
<c80>     gt_current_time(_123621,57), _123621 >= 7, _123621 <= 15
```

SocsIDs	contextId	Sender	Receivers	Performative	Content	Time
				current_time		2
s0	r1	a	b	request	check_balance	3
				current_time		5
s0	r1	b	a	agree	check_balance	6
				current_time		57
s0	r1	b	a	inform_result	check_balance, balance(300,currency(usd))	58

Figure 5. *A Participant does not respect a deadline*

carrier, the procedures for paying the bills, for handling complaints, what obligations/penalties apply in case of late payments, and how to delegate authority to the relevant bureaus, to make any necessary determination as to whether the parties have complied with all requirements as set forth in the contract. We enucleated a set of clauses in the contract, and gave their specifications in the $\mathcal{S}$CIFF framework. The ICs are reported in Tab. 8.3, and the KB is reported in Tab. 8.4. We chose a set of clauses about bill and complaint handling. Before we proceed onto explaining them one by one, we point out that the contract never obliges the customer to pay money to *telco*, although it gives *telco* the right to send a request for payment if the customer does not pay by a deadline. If after the request for payment the customer still does not pay, *telco* has the right to de-activate the carrier. On the other hand, the customer can decide to complain about (part of) the bill, provided that he has not received a request for payment about it. In case of complaint, the customer can no longer be expected to pay, and *telco* cannot request further payments about that bill.

8.3.2 Specifications in terms of ICs

Given the double reading (protocol vs. contract) of this example, we will sometimes use terms such as “it cannot”, “it is obliged”, or “it may”/“it is possible”/“it has the right to” where we should say, in terms of $\mathcal{S}$CIFF, “it is expected not to”, “it is expected to”, or “it is not expected not to”. We do this for the sake of readability. Let the reader not be confused by this terminological shift, as the semantics of the

Table 8.3 $\mathcal{IC}$ in the contract between *telco* (T) and a customer (C).

$$
\begin{array}{rl}
[IC1] & \mathbf{H}(\mathit{tell}(T, C, \mathit{phone\_bill}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T1}) \wedge \\
& \mathit{default\_wait}(\mathit{TWait}) \\
\rightarrow & \mathbf{EN}(\mathit{tell}(T, C, \mathit{request\_payment}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Any\_Amnt}), D), \mathit{T2}), \\
& \mathit{T2} > \mathit{T1}, \quad \mathit{T2} < \mathit{T1} + \mathit{TWait}.
\end{array}
$$

$$
\begin{array}{rl}
[IC2] & \mathbf{H}(\mathit{tell}(T, C, \mathit{phone\_bill}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T1}) \wedge \\
& \mathit{default\_wait}(\mathit{TWait}) \\
\rightarrow & \mathbf{E}(\mathit{tell}(C, T, \mathit{pay}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}, \mathit{Paymt\_Rcpt}), D), \mathit{T2}), \\
& \mathit{T2} < \mathit{T1} + \mathit{TWait} \\
\vee & \mathbf{E}(\mathit{tell}(C, T, \mathit{complain}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Partl\_Amnt}), D), \mathit{T3}), \\
& \mathit{T3} < \mathit{T1} + \mathit{TWait} \\
\vee & \neg\mathbf{EN}(\mathit{tell}(T, C, \mathit{request\_payment}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T4}), \\
& \mathit{T4} > \mathit{T1} + \mathit{TWait}.
\end{array}
$$

$$
\begin{array}{rl}
[IC3] & \mathbf{H}(\mathit{tell}(T, C, \mathit{phone\_bill}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T1}) \wedge \\
& \mathbf{H}(\mathit{tell}(T, C, \mathit{request\_payment}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T2}) \wedge \\
& \neg\mathbf{EN}(\mathit{tell}(T, C, \mathit{request\_payment}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T2}) \wedge \\
& \mathit{default\_wait}(\mathit{TWait}) \\
\rightarrow & \neg\mathbf{EN}(\mathit{tell}(T, C, \mathit{de\_activate}(\mathit{Phone\_No}, \mathit{reason}(\mathit{Bill\_Id})), D), T3), \\
& T3 > \mathit{T2} + \mathit{TWait} \\
\vee & \mathbf{E}(\mathit{tell}(C, T, \mathit{pay}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}, \mathit{Paymt\_Rcpt}), D), \mathit{T4}), \\
& T4 < \mathit{T2} + \mathit{TWait}.
\end{array}
$$

$$
\begin{array}{rl}
[IC4] & \mathbf{H}(\mathit{tell}(T, C, \mathit{request\_payment}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T1}) \wedge \\
& \mathbf{H}(\mathit{tell}(C, T, \mathit{pay}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}, \mathit{Paymt\_Rcpt}), D), \mathit{T2}) \wedge \\
& \mathit{default\_wait}(\mathit{TWait}) \wedge \mathit{T2} < \mathit{T1} + \mathit{TWait} \\
\rightarrow & \mathbf{EN}(\mathit{tell}(T, C, \mathit{de\_activate}(\mathit{Phone\_No}, \mathit{reason}(\mathit{Bill\_Id})), D), \mathit{T3}).
\end{array}
$$

$$
\begin{array}{rl}
[IC5] & \mathbf{H}(\mathit{tell}(T, C, \mathit{phone\_bill}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T1}) \wedge \\
& \mathbf{H}(\mathit{tell}(C, T, \mathit{complain}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Partl\_Amnt}), D), \mathit{T2}) \wedge \\
& \mathit{default\_wait}(\mathit{TWait}) \wedge \mathit{T2} < \mathit{T1} + \mathit{TWait} \wedge \\
& \mathit{is\_admissible\_complaint}(\mathit{Bill\_Id}, \mathit{Partl\_Amnt}) \\
\rightarrow & \neg\mathbf{E}(\mathit{tell}(C, T, \mathit{pay}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Partl\_Amnt}, \mathit{Paymt\_Rcpt}), D), \mathit{T3}), \\
& \mathit{T3} > \mathit{T1}, \\
& \mathbf{EN}(\mathit{tell}(T, C, \mathit{request\_payment}(\mathit{Phone\_No}, \mathit{Bill\_Id}, \mathit{Bill\_Amnt}), D), \mathit{T4}).
\end{array}
$$

Table 8.4 KB in the contract between *telco* and a customer.

$default\_wait(10).$

$is\_admissible\_complaint(Bill\_Id, Partl\_Amnt) \leftarrow$
$\quad list\_of\_bills(L1),$
$\quad member((Bill\_Id, Total\_Amnt), L1),$
$\quad Partl\_Amnt < Total\_Amnt.$

$member(X, [X|Tail]).$
$member(X, [Y|Tail]) \leftarrow$
$\quad member(X, Tail).$

$list\_of\_bills([(145886, 205), (114477, 407), (168945, 126)]).$

$\mathcal{S}$CIFF framework remains unchanged.

Tab. 8.3 contains five ICs: roughly speaking, the first three ones describe in general what is the expected behaviour of *telco*, regarding bill handling, whereas the last two ones are about the rights of the customer (*C*). The ICs state the following:

- by $[IC1]$, after sending a bill at time *T1*, *telco* may not send requests for payments before time $T1 + TWait$;
- by $[IC2]$, after *telco* sends a bill at time *T1*, one of the following three expectations hold: either *C* pays the bill in full by $T1 + TWait$, or *C* complains about (part of) the bill by $T1 + TWait$, or *telco* gains the right to send a request or payment at some time *T4* later than $T1 + TWait$.
- by $[IC3]$, if *telco* sent a bill, and later a request for payment at a time in which it had the right to do so, and if the request for payment concerns the bill in full, then ether *C* pays the bill, or *telco* gains the right to de-activate the carrier (although *telco* is not obliged to do so);
- by $[IC4]$, if *C* has paid the bill by the deadline, then *telco* cannot de-activate the carrier. The deadline is specified by predicate *default_wait*. Notice that $[IC4]$ fires independently of *telco* actually having the right to send a request for payments;
- by $[IC5]$, after *C* makes a complaint about some part of the bill (*Partl_Amnt*), he is no longer expected to pay *Bill_Amnt* (provided that the complaint is admissible).

In the KB part of the $\mathcal{S}$CIFF program, shown in Tab. 8.4, we specify deadlines, as in the previous example, and we define what an “admissible complaint” is. To this end, we define a predicate *is_admissible_complaint/2*, which relies upon a database of bills (“list of bills”). In this simplified example, the database is mimicked by a predicate named *list_of_bills/1*.

8.3.3 Sample interactions

Let us consider the following case: *telco* sends the bill, and C does not pay. As a consequence, after $TWait$ time units *telco* sends C a request for payment.

This sequence of events, i.e., the first three messages in (17), marked by $\star$, generates a set of fulfilled expectations. What happens is, after the first message at time 19 (the notification of the *phone_bill*), $[IC2]$ generates three alternative and equally plausible sets of expectations: either C is expected to pay before time 29, or C is expected to complain before time 29, or otherwise *telco* has the right ($\neg\mathbf{EN}$) to issue a request for payment after time 29. In all cases *telco* does not have the right to send a request for payment before time 29, because of $[IC1]$. At time 29 the first two alternatives become invalid due to the expired deadline. The message *request_payment* at time 33 is indeed acceptable, according to the protocol, and it gives *telco* explicit permission to de-activate the carrier at any time later than 29. In particular, by $[IC3]$, it generates a new choice point in the tree of expectation sets: in one case *telco* has the right to de-activate the carrier after time 39, in the other case C is expected to pay. Because of $[IC4]$, the last message, in which C notifies his payment to *telco*, has as a side effect that *telco* loses its right to de-activate the carrier at any time in connection to the bill N^o *145886*.

As the second example shows (17), a violation can be generated if *telco* de-activates the carrier. In that case, $\mathcal{S}$CIFF detects a violation because the fourth message violates the protocol, and in particular $[IC4]$, by which *telco* is expected not to de-activate the carrier if C pays within 10 time units after receipt of *telco*'s request for payment.

$$\left.\begin{array}{r}
\mathbf{H}(\ tell(\ telco,\ c,\ phone\_bill(390512093086, 145886, 205),\ 19).\\
\mathbf{H}(\ tell(\ telco,\ c,\ request\_payment(390512093086, 145886, 205),\ 33).\\
\mathbf{H}(\ tell(\ c,\ telco,\ pay(390512093086, 145886, 205, 1674521),\ 37).
\end{array}\right](\star)
\\
\mathbf{H}(\ tell(\ telco,\ c,\ de\_activate(390512093086, reason(145886)),\ 38).
\tag{17}$$

Let us consider a third example (18), starting by *telco* sending C a bill, as in all other examples. C complains, but he does it at time 33, which unfortunately is after the deadline of 10 time units after the bill. This complaint, although not specifically disallowed by the protocol, does not change the state of expectations in the system, since no IC fires. In particular, $[IC5]$ says that if C complains before the deadline, he is not expected any more to pay the amount he complained about, and *telco* loses the right to send requests for payment concerning either the amount C complained about or concerning the full amount of the bill. But $[IC5]$ (as well as the other ICs) does not say what happens in case of a late complaint. *telco* therefore sends him a request to payment, since it is its right, and the only options for C are either to pay, or to have the carrier de-activated. C pays and *telco* has no more right to de-activate the line, which incidentally makes that second option (have the carrier de-activated) inconsistent, besides fulfilling all the expectations of the first branch.

$$\begin{array}{l}
\mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{phone\_bill}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 19). \\
\mathbf{H}(\ \mathit{tell}(\ c,\ \mathit{telco},\ \mathit{complain}(\mathit{390512093086}, \mathit{145886}, \mathit{150}),\ 33). \\
\mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{request\_payment}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 34). \\
\mathbf{H}(\ \mathit{tell}(\ c,\ \mathit{telco},\ \mathit{pay}(\mathit{390512093086}, \mathit{145886}, \mathit{205}, \mathit{1674521}),\ 37).
\end{array} \tag{18}$$

In the last example (19), *telco* as usual sends C a bill. However, this time C sends his complaint before the deadline. C complains about an amount of €150 out of €205. As a consequence, if *telco* sends C a request for payment, it causes a protocol violation. Due to $[IC5]$, *telco* can no longer issue a request for payment. Unfortunately, *telco* does so at time 34, and consequently $\mathcal{S}$CIFF detects the violation of $[IC5]$. Note that, in order to demonstrate the role of predicate definitions in the KB, we have written $[IC5]$ so that it fires only if *is_admissible_complaint* (defined in the KB) holds.

$$\begin{array}{l}
\mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{phone\_bill}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 19). \\
\mathbf{H}(\ \mathit{tell}(\ c,\ \mathit{telco},\ \mathit{complain}(\mathit{390512093086}, \mathit{145886}, \mathit{150}),\ 24). \\
\mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{request\_payment}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 34).
\end{array} \tag{19}$$

9 Related Work and Discussion

In this section we relate the $\mathcal{S}$CIFF framework with other relevant work of literature. We will focus on other ALP frameworks and on other applications of computational logic to multi-agent systems. We do not intend to give an exhaustive account of the work done, but we will only touch the most closely related proposals and focus on the differences with respect with our own work.

9.1 ALP frameworks

By reading Kakas and colleagues' survey on ALP [63], one will be impressed by the amount of work done on this topic. The reasons why we are proposing yet another ALP framework are in the introductory section of this article. Now, we will try to relate our work with some of the most influential proposal of literature, although we are aware that many others will have to be left out.

Kakas and Mancarella [64] define a proof procedure (herein and below referred to as *KM*) for ALP, building on previous work by Eshghi and Kowalski [47]. *KM* assumes that the integrity constraints are in the form of *denials*, with at least one abducible literal in the conditions.[11] The semantics given by *KM* to the integrity constraints is that at least one of the literals in the integrity constraint must be false (otherwise, procedurally, *false* is derived). The procedure starts from a query and a set of initial assumptions Δ_i and results in a set of consistent hypotheses (abduced literals) Δ_o such that $\Delta_o \supseteq \Delta_i$ and Δ_o together with the program P entails the query. The proof procedure uses the notion of *abductive* and *consistency derivations*.[12]

Operationally, in *KM* abducibles must be ground when they are considered by the proof, and the procedure *flounders* if a selected abducible is not ground. Moreover, it treats *constraint predicates*, such as $<, \leq, \neq, \ldots$, as ordinary predicates, thus being unable to use specialised constraint solvers for such predicates. Therefore, extensions to *KM* have been proposed to cope with such limitations. Notably, *ACLP* [65] extends *KM* to deal with non-ground abduction and with constraints. ACLP programs can contain constraints on finite domains. ACLP interleaves consistency checking of abducible assumptions and constraint satisfaction.

Denecker and De Schreye [37, 39] introduce a proof procedure for normal abductive logic programs by extending SLDNF resolution to the case of abduction. The procedure is called *SLDNFA* and it is correct with respect to the completion semantics, and interestingly, it presents a crucial property: the treatment of *non-ground* abductive queries. [37] does not consider general integrity constraints, but only constraints of the kind $a, \mathit{not}\ a \Rightarrow false$. In later work [38], they propose adding integrity constraints by extending the program with rules $\mathit{false} \leftarrow \neg F$, for each integrity constraint F; the literal $\neg\mathit{false}$ is then added as an extra literal to the query. SLDNFA has been extended towards CLP constraints handling, giving rise to SLDNFA(C) [100].

The $\mathcal{A}$-System [66] is a merger of ACLP and SLDNFA(C), but it differs from them by its explicit treatment of non-determinism, which permits to perform heuristic search with different types of heuristics. Also $\mathcal{A}$-System, like $\mathcal{S}$CIFF, copes with

[11]The syntax of integrity constraints varies from framework to framework; while some frameworks require integrity constraints to be denials of literals, this is not true of other frameworks, such as $\mathcal{S}$CIFF, and IFF, as we will see.

[12]Intuitively, an abductive derivation is a standard SLD-derivation suitably extended in order to consider abducibles. As soon as an abducible atom δ is encountered which does not already occur in the current set of hypotheses, it is added to the current set of hypotheses, and it must be proved that any integrity constraint such that δ unifies with an abducible in it is satisfied. For this purpose, a consistency derivation for δ is started. Since the integrity constraints are denials only (i.e., queries), this corresponds to proving that every such query fails to hold. Therefore, δ is removed from all the denials with which it unifies, and it is proved that all the resulting queries fail. In this consistency derivation, when an abducible is encountered, an abductive derivation for its complement is started in order to prove the abducible's failure, so that the initial integrity constraint is satisfied.

non-ground abduction.

The *Active-KM* proof procedure by Terreni *et al.* [75] integrates in the original abductive computational scheme a limited but powerful type of implicative-form integrity constraints. It supports forward reasoning via integrity constraints (implications) which fire when their conditions (body) are satisfied. However, this procedure does not deal with non-ground abducibles. Finally, the *KM* proof-procedure has been used and extended in the context of MAS. In particular, Ciampolini *et al.*'s ALIAS framework [29] and the LAILA language [30] define mechanisms for the coordination of agent reasoning based on it.

Surely the most related abductive framework to $\mathcal{S}$CIFF is Fung and Kowalski's IFF proof-procedure [56], on which $\mathcal{S}$CIFF is based. The IFF proof procedure uses backward reasoning with the selective Clark completion [31] of the logic program[13] to compute abductive explanations for given queries. Forward reasoning is applied based on the conjunction of queries plus integrity constraints, which is done at the beginning of the abductive process. The integrity constraints can be any (closed) implications. The authors describe IFF as a sort of "hybrid of the proof procedure of Console *et al.* [35] and the SLDNFA procedure of Denecker and De Schreye (see [37])," mainly due to its use of the Clark completion semantics and because neither of them requires a safe selection rule for abducibles and negation.

IFF has been used to model the rational part of logic-based agents, since Kowalski and Sadri's seminal paper [68], and in further developments and refinements [69, 82, 72]. $\mathcal{S}$CIFF also applies ALP to the context of MAS, but differently from other work it does it at the social level, its initial purpose being to perform the compliance check of externally observable agent behaviour.

Recently, IFF has been refined to deal with negation as failure in integrity constraints [81], and extended with the definition of frameworks that treat abducibles and constraints uniformly [71, 45]. This last work also presents an implementation of IFF (the only one published, to the best of our knowledge), based on a meta-interpreter. Although these extensions improve IFF in several aspects, none of them handles universally quantified variables in abducible predicates, and of course do not deal with expectations. Finally, $\mathcal{S}$CIFF is implemented in CHR with attributed variables, which is a considerably efficient technology.

Given the CHR-based implementation of $\mathcal{S}$CIFF, we will also mention Abdennadher and Christiansen's work [1], which further developed into the HYPROLOG system [28]. HYPROLOG is not limited to abduction, but also encloses assumptive logic programming features. The abductive part of HYPROLOG, however, is much more restrictive in scope than $\mathcal{S}$CIFF: it has a limited use of negation, and integrity constraints cannot involve defined predicates (but only abducibles and built-ins). Thanks to these simplifications, the necessary machinery is much simpler than the

[13]The term "selective" refers to the fact that IFF does completion, but only of non-abducible predicates.

one used by $\mathcal{S}$CIFF. We implemented (and tested) a subset of the $\mathcal{S}$CIFF language based on ideas similar to HYPROLOG; this is documented in previous publications [59, 10].

Finally, related to our work on ALP are the abductive query evaluation method proposed by Satoh and Iwayama [85], and *Abdual* [16]: a system to perform abduction from extended logic programs grounded on the well-founded semantics. Abdual, which relies on tabled evaluation inspired to SLG resolution [27], handles only ground programs.

A little bit outside of ALP, but related to our work, Sergot [86] proposed a framework, *query-the-user*, in which some of the predicates are labelled as "askable"; the truth of askable atoms can be asked to the user. Our **E** predicates may be understood as information asking, while **H** atoms may be considered as new information provided during search. However, differently from Sergot's query-the-user, $\mathcal{S}$CIFF is not intended to be used interactively, but rather to provide a means to generate and to reason upon generated expectations, be them positive or negative. Moreover, $\mathcal{S}$CIFF presents expectations in the context of an abductive framework (integrating them with other abducibles). Hypotheses confirmation was studied also by Kakas and Evans [48], where hypotheses can be corroborated or refuted by matching them with observable atoms: an explanation fails to be corroborated if some of its logical consequences are not observed. The authors suggest that their framework could be extended to take into account dynamic events, possibly, queried to the user: *"this form of reasoning might benefit from the use of a query-the-user facility"*. In is work about AEC-based planning, Shanahan [89] also introduces a concept of expectation: a robot moves in an office, and has expectations about where it is standing, based on the values obtained by sensors. While our expectations should match with actual events, in Shanahan's work events and expectations are of the same nature, and both are abduced. We deal with expectations in a larger sense, as $\mathcal{S}$CIFF permits to express positive and negative expectations. We also have a different focus: while we assume that the history is known, Shanahan proposes to abduce the events.

In a sense, our work can be considered as a merger and extension of these works: it has confirmation of hypotheses, as in corroboration, and it provides an operational semantics for dynamically incoming events, as in query-the-user.

Also related to reasoning with dynamic incoming events are two additional works, which we briefly mention before we conclude this roundup. Speculative Computation [84] is a propositional framework for a multi-agent setting with unreliable communication. When an agent asks a query, it also abduces a default answer; if the real answer arrives within a deadline, the hypothesis is (dis-)confirmed; otherwise the computation continues with the default. In our work, expectations can be confirmed by events, with a wider scope: they are not only questions, and they can have variables, possibly constrained. The dynamics of incoming events can be seen as an instance of an Evolving Logic Program [17]. In EvoLP, the knowledge base can change both because of external events or because of internal results. $\mathcal{S}$CIFF does not generate new events, but only expectations about external events. Our focus is more on the expressivity of the expectations than on the evolution of the knowledge base.

9.2 Computational Logic and societies of agents

To the best of our knowledge, the SOCS approach to agent societies, upon which $\mathcal{S}$CIFF found its main motivations, is the first attempt to use ALP to reason about agent interaction at a social level. Many other logics have been proposed to represent richer social and institutional entities, such as normative systems and electronic institutions. Here also the literature is broad, and slightly aside of the focus of this article. However, our work shares some concepts with normative systems, being **E** related with the $\mathcal{O}$ (obligation) operator of deontic logic [83], and **EN** with the $\mathcal{F}$ (forbidden) operator.[14] We enucleate similarities and differences in [11], and comment on the main differences between our approach and others based on social semantics in a number of published papers [9, 12, 6]. Below we will only give a very synthetic and by no means exhaustive account of work based on computational logic, applied to agent interaction and social agent systems in the broader sense.

The social approach to the semantic characterisation of agent interaction is adopted by many researchers to allow for flexible, architecture-independent and verifiable protocol specification. Prominent schools, including Castelfranchi's [26], Singh *et al.*'s [91, 104], and Colombetti *et al.*'s [52, 33, 34] indicate commitments as first class entities in social agents, to represent the state of affairs in the course of social agent interaction. The resulting framework is more flexible than traditional approaches to protocol specification, as it does not necessarily define action sequences, nor it prescribes initial/final states or necessary transitions.

In [104], a variant of the Event Calculus is applied to commitment-based protocol specification. The semantics of messages (i.e., their effect on commitments) is described by a set of *operations* whose semantics, in turn, is described by *predicates* on *events* and *fluents*; in addition, commitments can evolve, independently of communicative acts, in relation to *events* and *fluents* as prescribed by a set of *postulates*. Similarly, [52] defines an operational specification of an ACL in an object-oriented framework by means of the *commitment* class. A commitment represents an obligation for its *debtor* towards its *creditor*. A commitment is described by a finite state automaton, whose states (which can take the values of *empty*, *pre-commitment*, *canceled*, *conditional*, *active*, *fulfilled* and *violated*) can change by application of methods

[14]The reduction of deontic concepts such as obligations and prohibitions has been the subject of several past works: notably, by [18] (according to which, informally, A is obligatory iff its absence produces a state of violation) and by [76] (where, informally, an action A is prohibited iff its being performed produces a state of violation).

of the commitment class, or of rules triggered by external conditions. The semantics of communicative acts is specified in terms of methods to be applied to a commitment when a communicative act is issued. We discuss in [9] the use of the SOCS framework for the social semantic specification of agent interaction protocols.

Artikis *et al.* [22] 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 systems where the behaviour of the members and their interactions cannot be predicted in advance. Three key components of computational systems are specified, namely the social constraints, social roles and social states. The specifications of these concepts is based on and motivated by the formal study of legal and social systems (a goal of the ALFEBIITE project), and therefore operators of Deontic Logic are used for expressing legal social behaviour of agents [102, 96]. ALFEBIITE has investigated the application of formal models of norm-governed activity to the definition, management and regulation of interactions between info-habitants in the information society. Their logical framework comprises a set of building blocks (including doxastic, deontic and praxeologic notions) as well as composite notions (including deontic right, power, trust, role and signalling acts).

Differently from [22] (and from other work on normative systems), we do not explicitly represent concepts such as institutional power of the society members and validity of action. Instead, permitted are all social events that do not determine a violation, i.e., all events that are not explicitly forbidden are allowed. Permission instead, if explicitly needed, is mapped the negation of a negative expectation ($\neg\mathbf{EN}$).

[80] provides a first-order framework of deontic reasoning that can model and compute social regulations and norms, and among the organizational models, [41, 43, 42] exploit deontic logic to specify the society norms and rules. Several papers discuss "sub-ideal" situations, i.e., how to manage situations in which some of the norms are not respected. For instance, [97] show the relation between diagnostic reasoning and deontic logic, importing the "principle of parsimony" from diagnostic reasoning into their deontic system, in the form of a requirement to minimize the number of violations. [77] proposes a solution to the problem and paradoxes stemming from earlier logical representations of *contrary-to-duty* obligations, i.e., obligations that become active when other obligations are violated. The Interactive Maryland Platform for Agents Collaborating Together (IMPACT) [21, 44] also uses deontic operators: not to describe social stances, but to program intelligent agents. A lot more work is surely relevant to $\mathcal{S}$CIFF and we indeed aim to incorporate some more advanced aspects of normative agent systems reasoning in the future: but for the economy of this article, we will redirect the interested reader to [11].

10 Conclusions

The operational framework presented in this article, based on the $\mathcal{S}$CIFF, represents an important improvement with respect to the state-of-the-art. Not only can it be used to specify a broad range of interaction protocols and verify agent interaction at runtime: $\mathcal{S}$CIFF abstracts away from the domain of multi-agent systems, and can be applied to specification and verification of interaction independently of the architecture of the participants in the interaction. $\mathcal{S}$CIFF presents a conjunction of features which is novel in the ALP literature, namely a particularly rich and intuitive treatment of variable quantification, the new concept of positive and negative expectations, an automatic way to verify compliance to specifications on-the-fly, an efficient implementation based on CHR, its embedding in an agent verification software tool able to interact with other standard components, and three important theoretical results: soundness, termination, and completeness for an important class of programs. This article focusses on the theoretical side of $\mathcal{S}$CIFF: it presents for the first time its operational semantics and proof of formal properties. In other publications we have demonstrated more in detail the use of $\mathcal{S}$CIFF in the multi-agent domain, and situated our approach inside the MAS literature.

Directions for future research include the definition of an extended $\mathcal{S}$CIFF which can be used to perform automatic verification of formal protocol properties. To this end, may rely on g-$\mathcal{S}$CIFF, as we demonstrate in [8]. We aim to produce a unified framework where protocol properties can be studies and verified statically, to produce specifications that can be used with no further translation steps in operating agent systems that we can monitor and verify at run time. We have already done some work in this sense, motivated by the need to reduce the number of steps involved in the complicated process of engineering and executing distributed and open computer systems applications. Other application domains for $\mathcal{S}$CIFF that we are considering are the medical domain (both for diagnosis, one of the classical ALP applications, and for medical guidelines specification), normative systems, open network computer systems, and web services choreography.

Acknowledgements

We would like to thank Luís M. Pereira and Marek J. Sergot, as well as the participants to the SOCS project, for many interesting and fruitful discussions.

This work is partially funded by the Information Society Technologies programme of the European Commission under the IST-2001-32530 SOCS Project (`http://lia.deis.unibo.it/research/socs/`), and by the MIUR COFIN 2003 projects *Sviluppo e verifica di sistemi multiagente basati sulla logica* (`http://www.di.unito.it/massive/`), and *La Gestione e la negoziazione automatica dei diritti sulle opere dell'ingegno digitali: aspetti giuridici e informatici.*

References

[1] Slim Abdennadher and Henning Christiansen. An experimental CLP platform for integrity constraints and abduction. In H.L. Larsen, J. Kacprzyk, S. Zadrozny, T. Andreasen, and H. Christiansen, editors, *FQAS, Flexible Query Answering Systems*, LNCS, pages 141–152, Warsaw, Poland, October 25–28 2000. Springer-Verlag.

[2] Marco Alberti, A. Bracciali, Federico Chesani, Anna Ciampolini, U. Endriss, Marco Gavanelli, A. Guerri, Antonis Kakas, Evelina Lamma, W. Lu, Paolo Mancarella, Paola Mello, Michela Milano, Fabrizio Riguzzi, Fariba Sadri, Kostas Stathis, G. Terreni, Francesca Toni, Paolo Torroni, and A. Yip. Experiments with animated societies of computees. Technical report, SOCS Consortium, 2005. Deliverable D14. Available electronically from the SOCS project web site: `http://lia.deis.unibo.it/research/socs/guests/publications/`.

[3] Marco Alberti, Federico Chesani, Marco Gavanelli, Alessio Guerri, Evelina Lamma, Paola Mello, and Paolo Torroni. Expressing interaction in combinatorial auction through social integrity constraints. *Intelligenza Artificiale*, II(1):22–29, 2005.

[4] Marco Alberti, Federico Chesani, Marco Gavanelli, and Evelina Lamma. The CHR-based Implementation of a System for Generation and Confirmation of Hypotheses. Number 2005-01 in Ulmer Informatik-Berichte, pages 111–122, 2005.

[5] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. A logic based approach to interaction design in open multi-agent systems. In *Proceedings of the 13th IEEE international Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004), 2nd international workshop "Theory And Practice of Open Computational Systems (TAPOCS)"*, pages 387–392, Modena, Italy, June 14–16 2004. IEEE Press.

[6] 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 Corrado Priami and Paola Quaglia, editors, *Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers*, volume 3267 of *Lecture Notes in Artificial Intelligence*, pages 324–339. Springer-Verlag, 2005.

[7] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Compliance verification of agent interaction: a logic-based tool. *Applied Artificial Intelligence*, 20(2-4):133–157, February-April 2006.

[8] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Security protocols verification in Abductive Logic Programming: a case study. In Oguz Dikenelli, Marie-Pierre Gleizes, and Andrea Ricci, editors, *Proceedings of ESAW'05, Kuşadasi, Aydin, Turkey, October 26-28, 2005*, volume 3963 of *Lecture Notes in Artificial Intelligence*, pages 106–124. Springer-Verlag, 2006.

[9] Marco Alberti, Anna Ciampolini, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. A social ACL semantics by deontic constraints. In V. Mařík, J. Müller, and M. Pěchouček, editors, *Multi-Agent Systems and Applications III. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691 of *Lecture Notes in Artificial Intelligence*, pages 204–213, Prague, Czech Republic, June 16–18 2003. Springer-Verlag.

[10] Marco Alberti, D. Daolio, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo 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.

[11] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, Giovanni Sartor, and Paolo Torroni. Mapping deontic operators to abductive expectations. *Computational and Mathematical Organization Theory*, 2006. To appear.

[12] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo 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.

[13] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Specification and verification of agent interactions using social integrity constraints. *Electronic Notes in Theoretical Computer Science*, 85(2), 2003.

[14] 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/informatica/tr/`.

[15] 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.

[16] 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.

[17] J. J. Alferes, A. Brogi, J. A. Leite, and L. M. Pereira. Evolving logic programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 50–61. Springer-Verlag, September 2002.

[18] A. Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, 67:100–103, 1958.

[19] Krzysztof R. Apt and Marc Bezem. Acyclic programs. *New Generation Computing*, 9(3/4):335–364, 1991.

[20] Krzysztof R. Apt and Roland N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994.

[21] K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: a Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14(2):64–72, March/April 1999.

[22] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002), Part III*, pages 1053–1061, Bologna, Italy, July 15–19 2002. ACM Press.

[23] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. Jade - a java agent development framework. In Rafael H. Bordini, Mehdi

Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multi-agent Systems, Artificial Societies, and Simulated Organizations*, pages 125–147. Springer-Verlag, 2005.

[24] Andrea Bracciali, Ulle Endriss, Neophytos Demetriou, Antonis C. Kakas, Wenjin Lu, and Kostas Stathis. Crafting the mind of prosocs agents. *Applied Artificial Intelligence*, 20(2-4):105–131, February-April 2006.

[25] H.J. Bürckert. A resolution principle for constrained logics. *Artificial Intelligence*, 66:235–271, 1994.

[26] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the First International Conference on Multiagent Systems, San Francisco, California, USA*, pages 41–48. AAAI Press, 1995.

[27] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, January 1996.

[28] Henning Christiansen and Verónica Dahl. HYPROLOG: A new logic programming language with assumptions and abduction. In Maurizio Gabbrielli and Gopal Gupta, editors, *Logic Programming, 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005, Proceedings*, volume 3668 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2005.

[29] Anna Ciampolini, Evelina Lamma, Paola Mello, Francesca Toni, and Paolo Torroni. Co-operation and competition in *ALIAS*: a logic framework for agents that negotiate. *Computational Logic in Multi-Agent Systems. Annals of Mathematics and Artificial Intelligence*, 37(1-2):65–91, 2003.

[30] Anna Ciampolini, Evelina Lamma, Paola Mello, and Paolo Torroni. LAILA: A language for coordinating abductive reasoning among logic agents. *Computer Languages*, 27(4):137–161, February 2002.

[31] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[32] P. R. Cohen and C. R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3), 1979.

[33] M. Colombetti, N. Fornara, and M. Verdicchio. The role of institutions in multiagent systems. In *Proceedings of the Workshop on Knowledge based and reasoning agents, VIII Convegno AI*IA 2002*, Siena, Italy, 2002.

[34] Marco Colombetti, Nicoletta Fornara, and Mario Verdicchio. A social approach to communication in multiagent systems. 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 191–220. Springer-Verlag, May 2004. First International Workshop, DALT 2003. Melbourne, Australia, July 2003. Revised Selected and Invited Papers.

[35] L. Console, D. Theseider Dupré, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[36] P. Dell'Acqua, Fariba Sadri, and Francesca Toni. Combining introspection and communication with rationality and reactivity in agents. In Jürgen Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence, European Workshop, JELIA'98, Dagstuhl, Germany, October 12–15, 1998, Proceedings*, volume 1489 of *Lecture Notes in Computer Science*, pages 17–32. Springer-Verlag, October 1998.

[37] M. Denecker and D. De Schreye. SLDNFA: An abductive procedure for normal abductive programs. In Krzysztof R. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming, Washington, USA*, pages 686–702, Cambridge, MA, November 9–13 1992. MIT Press.

[38] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In *Logic Programming, Proceedings of the 1993 International Symposium, Vancouver, British Columbia, Canada*, pages 147–163, Cambridge, MA, 1993. MIT Press.

[39] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.

[40] Enrico Denti, Andrea Omicini, and Alessandro Ricci. Multi-paradigm Java-Prolog integration in tuProlog. *Science of Computer Programming*, 57(2):217–250, August 2005.

[41] V. Dignum, J. J. Meyer, F. Dignum, and H. Weigand. Formal specification of interaction in agent societies. In *Proceedings of the Second Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS), Maryland*, October 2002.

[42] V. Dignum, J. J. Meyer, and H. Weigand. Towards an organizational model for agent societies using contracts. In C. Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Au-*

tonomous Agents and Multiagent Systems (AAMAS-2002), Part II, pages 694–695, Bologna, Italy, July 15–19 2002. ACM Press.

[43] V. Dignum, J. J. Meyer, H. Weigand, and F. Dignum. An organizational-oriented model for agent societies. In *Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications. AAMAS'02, Bologna*, 2002.

[44] T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, March 1999.

[45] Ulle Endriss, Paolo Mancarella, Fariba Sadri, Giacomo Terreni, and Francesca Toni. The CIFF proof procedure for abductive logic programming with constraints. 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 Artificial Intelligence*, pages 31–43. Springer-Verlag, 2004.

[46] K. Eshghi. Abductive planning with the event calculus. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington*, Cambridge, MA, 1988. MIT Press.

[47] K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proceedings of the 6th International Conference on Logic Programming*, pages 234–255, Cambridge, MA, 1989. MIT Press.

[48] C.A. Evans and A.C. Kakas. Hypotheticodeductive reasoning. In *Proc. International Conference on Fifth Generation Computer Systems*, pages 546–554, Tokyo, 1992.

[49] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, MA, 1997.

[50] FIPA Communicative Act Library Specification, August 2001. Published on August 10th, 2001, available for download from the FIPA website, `http://www.fipa.org`.

[51] FIPA Request Interaction Protocol Specification. Standard SC00026H, Foundation for Intelligent Physical Agents, December 2002. Published on December 3, 2002, available for download from the FIPA website.

[52] N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. 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 535–542, Bologna, Italy, July 15–19 2002. ACM Press.

[53] N. Fornara and M. Colombetti. Defining interaction protocols using a commitment-based agent communication language. In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, pages 520–527, Melbourne, Victoria, July 14–18 2003. ACM Press.

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

[55] T. H. Fung. *Abduction by Deduction.* PhD thesis, Imperial College London, 1996.

[56] 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.

[57] Marco Gavanelli, Evelina Lamma, and Paola Mello. Proof of completeness of the SCIFF proof-procedure. Technical Report CS-2005-02, Dipartimento di Ingegneria, Università di Ferrara, 2005. Available at `http://www.ing.unife.it/informatica/tr/CS-2005-02.pdf`.

[58] Marco Gavanelli, Evelina Lamma, and Paola Mello. Proof of properties of the SCIFF proof-procedure. Technical Report CS-2005-01, Dipartimento di Ingegneria, Università di Ferrara, 2005. Available at `http://www.ing.unife.it/informatica/tr/CS-2005-01.pdf`.

[59] Marco Gavanelli, Evelina Lamma, Paola Mello, Michela Milano, and Paolo Torroni. Interpreting abduction in CLP. In Francesco Buccafurri, editor, *APPIA-GULP-PRODE Joint Conference on Declarative Programming*, pages 25–35, Reggio Calabria, Italy, September 3–5 2003. Università Mediterranea di Reggio Calabria.

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

[61] 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.

[62] A. C. Kakas, R. A. Kowalski, and Francesca Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[63] A. C. Kakas, R. A. Kowalski, and Francesca 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.

[64] A. C. Kakas and Paolo Mancarella. On the relation between Truth Maintenance and Abduction. In T. Fukumura, editor, *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI-90, Nagoya, Japan*, pages 438–443. Ohmsha Ltd., 1990.

[65] A. C. Kakas, A. Michael, and C. Mourlas. ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, 44(1-3):129–177, July 2000.

[66] A. C. Kakas, B. van Nuffelen, and M. Denecker. A-System: Problem solving through abduction. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA (IJCAI-01)*, pages 591–596, Seattle, Washington, USA, August 2001. Morgan Kaufmann Publishers.

[67] A.C. Kakas, Paolo Mancarella, Fariba Sadri, Kostas Stathis, and Francesca Toni. The KGP model of agency. In R. Lopez de Mantaras and L. Saitta, editors, *Proceedings of the Sixteenth European Conference on Artificial Intelligence, Valencia, Spain (ECAI 2004)*. IOS Press, August 2004.

[68] R. A. Kowalski and Fariba Sadri. From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25(3/4):391–419, 1999.

[69] R. A. Kowalski, Fariba Sadri, and Francesca Toni. An agent architecture that combines backward and forward reasoning. In B. Gramlich and F. Pfenning, editors, *Proceedings of the CADE-15 Workshop on Strategies in Automated Deduction*, pages 49–56, November 1998.

[70] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[71] R.A. Kowalski, Francesca Toni, and G. Wetzel. Executing suspended logic programs. *Fundamenta Informaticae*, 34:203–224, 1998.

[72] Robert A. Kowalski. The logical way to be artificially intelligent. In Paolo Torroni and Francesca Toni, editors, *Computational Logic in Multi-Agent Systems, 6th International Workshop, CLIMA VI, London, UK, June 27-29, 2005, Revised Selected and Invited Papers*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2006.

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

[74] Paolo Mancarella, Fariba Sadri, Giacomo Terreni, and Francesca Toni. Planning partially for situated agents. In João Leite and Paolo Torroni, editors, *Computational Logic in Multi-Agent Systems, 5th International Workshop, CLIMA V, Lisbon, Portugal, September 29-30, 2004, Revised Selected and Invited Papers*, volume 3487 of *Lecture Notes in Artificial Intelligence*, pages 230–248. Springer-Verlag, 2005.

[75] Paolo Mancarella and Giacomo Terreni. An abductive proof procedure handling active rules. 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 105–117. Springer Verlag, September 23–26 2003.

[76] J. J. Ch. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame J. of Formal Logic*, 29(1):109–136, 1988.

[77] Henry Prakken and Marek Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.

[78] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers, April 1991.

[79] R. Reiter. On closed-word data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978.

[80] Young U. Ryu and Ronald M. Lee. Defeasible deontic reasoning: A logic programming model. In J.-J.Ch. Meyer and R.J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*, pages 225–241. John Wiley & Sons Ltd, 1993.

[81] Fariba Sadri and Francesca Toni. Abduction with negation as failure for active and reactive rules. In Evelina Lamma and Paola Mello, editors, *AI*IA'99: Advances in Artificial Intelligence, Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence, Bologna*, number 1792 in Lecture Notes in Artificial Intelligence, pages 49–60. Springer-Verlag, 2000.

[82] Fariba Sadri, Francesca Toni, and Paolo Torroni. An abductive logic programming architecture for negotiating agents. In Sergio Greco and Nicola Leone,

editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *Lecture Notes in Computer Science*, pages 419–431. Springer-Verlag, September 2002.

[83] Giovanni Sartor. *Legal Reasoning*, volume 5 of *Treatise*. Kluwer, Dordrecht, 2004.

[84] Ken Satoh, K. Inoue, K. Iwanuma, and C. Sakama. Speculative computation by abduction under incomplete communication environments. In *Proceedings of the Fourth International Conference on Multi-Agent Systems, Boston, Massachusetts, USA*, pages 263–270. IEEE Press, 2000.

[85] Ken Satoh and N. Iwayama. A Query Evaluation Method for Abductive Logic Programming. In K. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming, Washington, USA*, pages 671–685, Cambridge, MA, 1992. MIT Press.

[86] M. J. Sergot. A query-the-user facility of logic programming. In P. Degano and E. Sandwell, editors, *Integrated Interactive Computer Systems*, pages 27–41. North Holland, 1983.

[87] Murray Shanahan. The event calculus explained. In Michael Wooldridge and Manuela M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, volume 1600 of *Lecture Notes in Computer Science*, pages 409–430. Springer Verlag, 1999.

[88] Murray Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44(1-3):207–240, 2000.

[89] Murray P. Shanahan. Reinventing Shakey. In J. Minker, editor, *Logic-based Artificial Intelligence*, volume 597 of *Kluwer Int. Series In Engineering And Computer Science*, pages 233–253, 2000.

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

[91] M. Singh. Agent communication language: rethinking the principles. *IEEE Computer*, pages 40–47, December 1998.

[92] 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, 2002-2005. Home Page: `http://lia.deis.unibo.it/research/socs/`.

[93] The SOCS protocol repository, 2005. Available at `http://edu59.deis.unibo.it:8079/SOCSProtocolsRepository/jsp/index.jsp`.

[94] SOCS-$\mathcal{S}$I home page, 2006. `http://lia.deis.unibo.it/research/socs_si/`.

[95] P.J. Stuckey. Negation and constraint logic programming. *Information and Computation*, 118(1):12–33, 1995.

[96] L. van der Torre. Contextual deontic logic: Normative agents, violations and independence. *Annals of Mathematics and Artificial Intelligence*, 37(1):33–63, 2003.

[97] Leendert W. N. van der Torre and Yao-Hua Tan. Diagnosis and decision making in normative reasoning. *Artificial Intelligence and Law*, 7(1):51–67, 1999.

[98] P. van Hentenryck and Y. Deville. The Cardinality Operator: A new Logical Connective for Constraint Logic Programming. In K. Furukawa, editor, *Logic Programming, Proceedings of the Eigth International Conference, Paris, France*, volume 2, pages 745–759, 1991.

[99] P. van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(fd). Technical Report CS-93-02, Department of Computer Sciences, Brown University, January 1993.

[100] B. van Nuffelen and M. Denecker. Problem solving in ID-logic with aggregates. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, NMR'00, Breckenridge, CO*, pages 1–9, 2000.

[101] M. Wooldridge. *Introduction to Multi-Agent Systems.* John Wiley & Sons, Ltd., 2002.

[102] G.H. Wright. Deontic logic. *Mind*, 60:1–15, 1951.

[103] I. Xanthakos. *Semantic Integration of Information by Abduction.* PhD thesis, Imperial College London, 2003. Available at `http://www.doc.ic.ac.uk/~ix98/PhD.zip`.

[104] P. Yolum and M.P. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. 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 527–534, Bologna, Italy, July 15–19 2002. ACM Press.