Expressing and Verifying Business Contracts with Abductive Logic Programming

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

February 10, 2007

Abstract

In this article, we propose to adopt the $\mathcal{S}$CIFF abductive logic language to specify business contracts, and show how its proof procedures are useful to verify contract execution and fulfilment. $\mathcal{S}$CIFF is a declarative language based on abductive logic programming, which accommodates forward rules, predicate definitions, and constraints over finite domain variables. Its declarative semantics is abductive, and can be related to that of deontic operators; its operational specification is the sound and complete $\mathcal{S}$CIFF proof procedure, defined as a set of transition rules, which has been implemented and integrated into a reasoning and verification tool. A variation of the $\mathcal{S}$CIFF proof-procedure (g-$\mathcal{S}$CIFF) can be used for static verification of contract properties.

We demonstrate the use of the $\mathcal{S}$CIFF language for business contract specification and verification, in a concrete scenario. In order to accommodate integration of $\mathcal{S}$CIFF with architectures for business contract, we also propose an encoding of $\mathcal{S}$CIFF contract rules in RuleML.

Authors' current affiliations

Marco Alberti, Marco Gavanelli, Evelina Lamma: ENDIF, University of Ferrara, Via Saragat 1, 44100 Ferrara, Italy.

Email: {`marco.alberti|marco.gavanelli|evelina.lamma`}`@unife.it`

Federico Chesani, Paola Mello, Marco Montali, Paolo Torroni: DEIS, University of Bologna, Viale del Risorgimento 2, 40123 Bologna, Italy.

Email: {`fchesani|pmello|mmontali|ptorroni`}`@deis.unibo.it`

1 Introduction

Business contracts are an important conceptual abstraction and a practical guiding and governance mechanism for cross-organizational collaboration. Contracts can be in fact considered as the main coordination mechanism for the extended enterprise [MGL$^+$04]. A business contract architecture [Mil95] is therefore an important part of the extended enterprise which aims to provide functionalities such as contract management and monitoring. Natural requirements for a contract management framework are a language with clear semantics for contract specification, and operational procedures for (i) verifying contract properties at design time, and (ii) verifying the conformance of parties to contracts at run time.

From a high-level, functional viewpoint, a contract management system is a component that is fed with the "what" of the problem, by domain expert users, and takes care of the "how," through a suitable execution model. Computational logics offer a broad range of languages and mechanisms that couple declarative ("what is") specification languages with sound operational ("how to") execution models that need not be disclosed to the user of the specification language. For this reason, we strongly believe that computational logic-based frameworks, adequately extended to support event-based monitoring of business activities associated with contracts, should play a key role in contract management systems.

Among the most influential computational logic frameworks for business contract representation and reasoning we find Courteous Logic Programming (CLP, [GLC99]) and Defeasible Logic (DL, [Gov05]), the former being in fact a variant of the latter [AMB00]. These are languages for nonmonotonic reasoning, mainly used in the context of business contracts to enable normative reasoning and to identify and resolve conflicts arising by events and contract rules, reason about violations, specify and enforce reparational obligations, and so on. In this article, in context of contract management systems, we are mailny concerned with the aspect of runtime monitoring and verification of contracts, rather than on the ontological and semantic aspects of contract specification. We focus primarily on the problem of *runtime evaluation* of contract policies, i.e., expressions consisting of behaviour constraints, event patterns and states [MGL$^+$04], to determine whether parties' obligations have been satisfied or whether there are violations to the contract.

We base our work on $\mathcal{S}$CIFF, the computational logic-based language and framework conceived within the context of the SOCS EU project [SOC05] to specify agent interaction protocols. $\mathcal{S}$CIFF consists of a logic language based on abductive logic programming, a sound and complete proof procedure [AGL$^+$05], and a software tool which implements it, based on an efficient inference engine and constraints solving technology [ACG$^+$06b]. First class entities in the $\mathcal{S}$CIFF language are *events*, which represent entities such as actions being taken, timeouts associated with deadlines, and external events such as messages being sent or services being requested, and *expectations*, which describe a desired behaviour in terms of events. Expectations are related with each other and with events by logical epxressions called Integrity Contraints (ICs). ICs express in fact behaviour contraints, and are the main building blocks in the specification of policies. Expectations are modeled in $\mathcal{S}$CIFF as abducible predicates, since they model event that may happen (but we do not know whether that will be the case). Thus the abductive nature of the framework. Expectations are related to deontic operators such as obligation, prohibition and permission [AGL+06].

In this paper, we propose $\mathcal{S}$CIFF as a language and operational framework to specify and reason upon business contracts. The deontic reading of $\mathcal{S}$CIFF specifications arising from such a relationship is one of the elements that make the $\mathcal{S}$CIFF language, we believe, a good candidate as a contract specification and reasoning language. Reasoning upon contract specifications (and events) can be done at two different stages of contract design and enactment: at run-time, for example in the way that we propose, and at design time, as it is the case with DL and CLP. We consider it important to enable these two kinds of verification within the same framework and, if possible, using the same specification language, in order to minimize translation errors and the unavoidable unaccuracy caused by different languages. To this end, an extension of $\mathcal{S}$CIFF, called g-$\mathcal{S}$CIFF, has been defined to verify protocol properties at design time [ACG+06c], and we show here how it can be used to enable design-time reasoning on contracts.

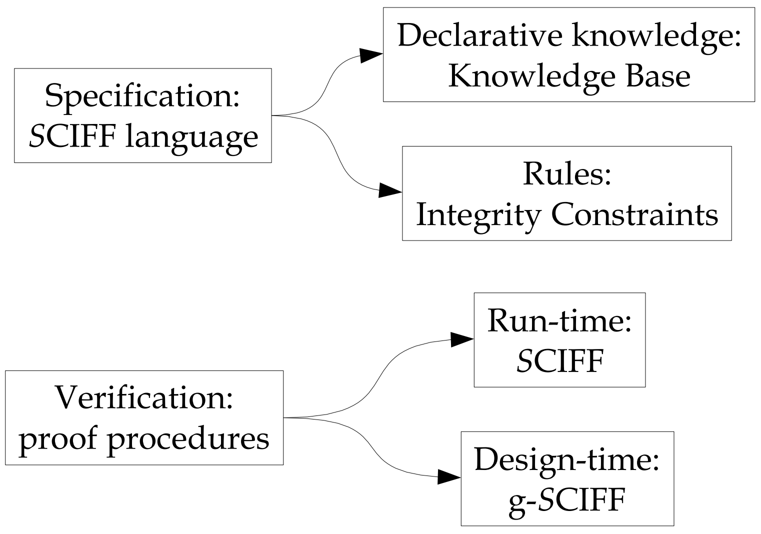

Figure 1: Contract specification and verification in the $\mathcal{S}$CIFF framework.

Fig. 1 summarizes the components of the $\mathcal{S}$CIFF framework that can be used in contract specification and verification.

The paper is structured as follows. Sect. 2 is devoted to the specification of contract in the $\mathcal{S}$CIFF language: we first give the necessary background, by presenting the syntax (Sect. 2.1) and declarative semantics (Sect. 2.2) of the $\mathcal{S}$CIFF language, then we show how $\mathcal{S}$CIFF expectations are related to deontic operators (Sect. 2.3) and finally we demonstrate $\mathcal{S}$CIFF in a concrete scenario, by proposing a possible specification of contract clauses (Sect. 2.4). In Sect. 3, we investigate the verification issues: first run-time verification (Sect. 3.1), and then design-time verification (Sect. 3.2), recalling, in each case, the relevant proof procedure ($\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF, respectively).

The integration of $\mathcal{S}$CIFF with architectures for business contract is facilitated by a suitable encoding of $\mathcal{S}$CIFF contract rules in RuleML, summarized in Sect. 4. A discussion of related work follows.

2 Contract specification

A contract in the $\mathcal{S}$CIFF language is basically specified by means of two components: a knowledge base, which defines declaratively domain-specific knowledge (such as deadlines) and a set of integrity constraints, which describe contract clauses and can be seen as forward rules that generate expectations about the behaviour of the parties involved in the contract. A declarative semantics based on abductive logic programming determines whether the parties have complied to the contract. A useful feature of the $\mathcal{S}$CIFF language is its integration of Constraint Logic Programming [JM94], which makes deadlines easy to specify and efficient to verify.

2.1 Syntax of the $\mathcal{S}$CIFF language

The $\mathcal{S}$CIFF language is composed of entities for expressing:

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

2.1.1 Representation of the behaviour of parties

Events *Events* are the abstractions used to represent the actual behaviour.

Definition 2.1. *An* event *is an atom:*

- *with predicate symbol* **H***;*
- *whose first argument is a ground term; and*
- *whose second 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.

In this paper, we map all events to communicative events, identified by the functor *tell*. In particular, the description of happened events is of the format

$$tell(Sender, Receiver, Content[, Dialog]),$$

where the optional *Dialog* parameter is an identifier of the interaction being described and the other arguments have the obvious meaning.

Example 2.2.

$$\mathbf{H}(\ tell(\ telco,\ c,\ phone\_bill(390512093086, 145886, 205),\ 19). \tag{1}$$

says that telco sent to c a phone_bill (for the phone number 390512093086, whose identifier is 145886 and whose amount is 205) at time 19.

A *negated event* is a negative literal **not H**$(\ldots, \ldots)$. We will call *history* a set of happened events, and denote it with the symbol **HAP**.

Expectations *Expectations* are the abstractions used to represent the desired events from an external viewpoint. They represent the ideal behaviour of the system, i.e., the actions that, once performed, would make the system compliant to its specifications. Our choice of the terminology "expectation" is intended to stress that events cannot be enforced, but only expected, to be as we would like them to be.

Expectations are of two types:

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

Definition 2.3. *A* positive expectation *is an atom:*

- *with predicate symbol* **E***;*
- *whose first argument is a term; and*
- *whose second 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 (which should not be confused with the time at which the expectation is generated, which is not modeled by $\mathcal{S}$CIFF's declarative semantics). Expectations may contain variables, which leaves the expected event not completely specified. Variables in positive expectations are always existentially quantified: if the time argument is a variable, for example, this means that the event is expected to happen at *any* time. We do not associate a specific semantics to time; we rather treat an expectation's time argument as any other variable. This choice simplifies the $\mathcal{S}$CIFF language's declarative and operational semantics.

Example 2.4. *The atom*

$$\mathbf{E}(\ tell(\ telco,\ c,\ phone\_bill(390512093086, Id, Amount), T). \tag{2}$$

says that telco is expected to send to c a phone_bill (for the number 390512093086, whose identifier is Id and whose amount is Amount) at time T.

A *negated positive expectation* is a positive expectation with the explicit negation operator $\neg$ applied to it. As explained in Sect. 2.1.2, variables in negated positive expectations are quantified as those in positive expectations.

Definition 2.5. *A* negative expectation *is an atom:*

- *with predicate symbol* **EN***;*
- *whose first argument is a term; and*
- *whose second 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 in which time points the event is expected not to happen. As well as positive expectations, negative expectations may contain variables, which are typically universally quantified[1]: for example, if the time argument is a variable, then the event is expected not to happen at *all* times.

Example 2.6. *The atom*

$$\mathbf{E}(\ tell(\ telco,\ \ c,\ \ phone\_bill(390512093086, Id, Amount), T). \qquad (3)$$

means says that telco is expected not *to send to c a phone_bill (for the number 390512093086, with any Id and for any Amount) at any time T.*

A *negated negative expectation* is a negative expectation with the explicit negation operator $\neg$ applied to it. As explained in Sect. 2.1.2, variables in negated negative expectations are quantified as those in negative expectations.

Note that $\neg\mathbf{E}(tell(bob, alice, refuse(phone\_number), dialog\_id), T_r)$ is different from $\mathbf{EN}(tell(bob, alice, refuse(phone\_number), dialog\_id), T_r)$. 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).

The syntax of events and expectations is summarised in Tab. 2.1, and it will be used as such by the subsequent Tab. 2.2 and 2.3. The syntactical entities *ExistLiteral* and *NbfLiteral* will also be used in the subsequent Tab. 2.2 and 2.3.

2.1.2 Contract specifications

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

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

[1]For a complete treatment of quantification in the $\mathcal{S}$CIFF language, we refer the interested reader to [ACG$^+$06d].

Table 2.1 Syntax of events and expectations

EventLiteral	::=	[**not**]*Event*
Event	::=	**H**(*GroundTerm*, *Integer*)
ExpLiteral	::=	*PosExpLiteral* | *NegExpLiteral*
PosExpLiteral	::=	$[\neg]$*PosExp*
NegExpLiteral	::=	$[\neg]$*NegExp*
PosExp	::=	**E**(*Term*, *Variable* | *Integer*)
NegExp	::=	**EN**(*Term*, *Variable* | *Integer*)
ExistLiteral	::=	*PosExpLiteral* | *Literal*
NbfLiteral	::=	**not** *Atom*
Literal	::=	[**not**]*Atom*

Knowledge Base The Knowledge Base (KB_S) is a set of *Clause*s in which the body can contain (besides defined literals) expectation literals and restrictions. [2]

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

Table 2.2 Syntax of the Knowledge Base

KB_S	::=	[*Clause*]*
Clause	::=	*Head* $\leftarrow$ *Body*
Head	::=	*Atom*
Body	::=	*ExtLiteral* [$\wedge$ *ExtLiteral*]* [: *Restriction* [, *Restriction*]*] | *true*
ExtLiteral	::=	*Literal* | *ExpLiteral*

The syntax of the Knowledge Base is given in Tab. 2.2, and it will be used as such also in Tab. 2.3.

Goal In the $\mathcal{S}$CIFF framework, the role of the *goal* is the same as in the logic programming literature, i.e., a predicate that should be entailed. Therefore, the term "goal" does not necessarily have the typical connotation (of "common" or "social" goal) found in multi-agent systems literature, though it can be used for such a purpose.

The syntax of the goal is the same as the *Body* of a clause (Tab. 2.2). The quantification rules are the following:

[2]In the $\mathcal{S}$CIFF language, restrictions can be considered as CLP constraints [JM94], that can also be applied to universally quantified variables with the semantics defined by Bürckert [Bür94].

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

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. 3. Declaratively, they relate the various entities in the $\mathcal{S}$CIFF framework, i.e., expectations, events, and constraints/restrictions, together with the predicates in the knowledge base.

Table 2.3 Syntax of Integrity Constraints (ICs)

$$\begin{array}{rcl}
\mathcal{IC}_S & ::= & [IC]^\star \\
IC & ::= & Body \rightarrow Head \\
Body & ::= & (EventLiteral \mid ExpLiteral)\ [\ \wedge BodyLiteral\ ]^\star \\
 & & \ [:\ Restriction\ [,\ \ Restriction\ ]^\star\ ] \\
BodyLiteral & ::= & EventLiteral \mid ExtLiteral \\
Head & ::= & HeadDisjunct\ [\ \ \vee HeadDisjunct\ ]^\star \mid false \\
HeadDisjunct & ::= & HeadLiteral\ [\ \ \wedge HeadLiteral]^\star\ [:\ Restriction\ [,\ \ Restriction\ ]^\star\ ] \\
HeadLiteral & ::= & Literal \mid ExpLiteral
\end{array}$$

The syntax of ICs is given in Tab. 2.3: the *Body* of ICs can contain conjunctions of all elements in the language (namely, **H**, **E**, and **EN** literals, defined literals and restrictions), and their *Head* contains a disjunction of conjunctions of any of the literals in the language, except for **H** literals.

Contract Specification Given a Knowledge Base KB_S and a set $\mathcal{IC}_S$ of Integrity Constraints, we call the pair $\langle KB_S, \mathcal{IC}_S \rangle$ a *Contract Specification.* Intuitively, a contract specification is a description of the acceptable, or desirable, histories, as defined by its declarative semantics, given formally in Sect. 2.2.

2.2 Declarative Semantics

In the following, we describe the (abductive) declarative semantics of the $\mathcal{S}$CIFF framework, which is inspired by other abductive frameworks such as the IFF by Fung and Kowalski [FK97], but introduces the concept of fulfilment, used to express a correspondence between the expected and the actual events. The declarative semantics of a contract specification is given for each specific history (see Sect. 2.1.1). We call a specification grounded on a history an *instance* of the contract.

Definition 2.7. Contract instance *Given a contract specification* $\mathcal{S} = \langle KB_S, \mathcal{IC}_S \rangle$ *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 contract specification $\mathcal{S}$, based on two different histories: $\mathbf{HAP}^i$ and $\mathbf{HAP}^f$.

We adopt an abductive semantics for the contract instance. Declaratively, a ground set **EXP** of hypotheses should entail the goal and satisfy the integrity constraints. In our case the set **EXP** of hypotheses is, in particular, a set of ground expectations, positive and negative, possibly negated by explicit negation. Notice that, by virtue of explicit negation, all of such expectations are positive abducible literals in ALP terminology.

Definition 2.8. Abductive explanation *Given a contract specification* $\mathcal{S} = \langle KB_S, \mathcal{IC}_S \rangle$, *an instance* $\mathcal{S}_{\mathbf{HAP}}$ *of* $\mathcal{S}$*, and a goal* $\mathcal{G}$*,* **EXP** *is an abductive explanation of* $\mathcal{S}_{\mathbf{HAP}}$ *for goal* $\mathcal{G}$ *if:*

$$Comp(KB_S \cup \mathbf{HAP} \cup \mathbf{EXP}) \cup \text{CET} \cup T_{\mathcal{X}} \models \mathcal{IC}_S \tag{4}$$

$$Comp(KB_S \cup \mathbf{EXP}) \cup \text{CET} \cup T_{\mathcal{X}} \models \mathcal{G} \tag{5}$$

where Comp represents the three-valued completion *of a theory [Kun87],* CET *is Clark [Cla78] Equational Theory, and* $T_{\mathcal{X}}$ *is the constraint theory [JM94].*

The symbol $\models$ is interpreted in three valued logics. In particular, if we interpret expectations as abducible predicates, we can rely upon a three-valued model-theoretic semantics as intended meaning, as done, for instance, in a different context, by Fung and Kowalski [FK97], Denecker and De Schreye [DS98].

We also require consistency with respect to explicit negation [AB94] and between positive and negative expectations.

Definition 2.9. ¬-consistency *A set* **EXP** *of expectations is* ¬-consistent *if and only if for each (ground) term p and integer t:*

$$\{\mathbf{E}(p,t), \neg\mathbf{E}(p,t)\} \not\subseteq \mathbf{EXP} \qquad \textit{and} \qquad \{\mathbf{EN}(p,t), \neg\mathbf{EN}(p,t)\} \not\subseteq \mathbf{EXP}. \tag{6}$$

Definition 2.10. E-consistency *A set* **EXP** *of expectations is* **E**-consistent *if and only if for each (ground) term p and integer t:*

$$\{\mathbf{E}(p,t), \mathbf{EN}(p,t)\} \not\subseteq \mathbf{EXP} \tag{7}$$

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 2.11. Fulfillment *Given a history* **HAP***, a set* **EXP** *of expectations is* **HAP**-fulfilled *if and only if*

$$\forall \mathbf{E}(p,t) \in \mathbf{EXP} \Rightarrow \exists \mathbf{H}(p,t) \in \mathbf{HAP} \;\; \textit{and} \;\; \forall \mathbf{EN}(p,t) \in \mathbf{EXP} \Rightarrow \nexists \mathbf{H}(p,t) \in \mathbf{HAP} \tag{8}$$

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

When all the given conditions (4-8) are met for at least one set of expectations **EXP**, we say that the goal is *achieved* and **HAP** is compliant to $\mathcal{S}$ with respect to $\mathcal{G}$ and **EXP**, and we write $\mathcal{S}_{\mathbf{HAP}} \models_{\mathbf{EXP}} \mathcal{G}$. In particular:

Operator	**Abducibile**
Forb A	$\mathbf{EN}(A)$
Obl A	$\mathbf{E}(A)$
Perm A	$\neg\mathbf{EN}(A)$
Perm $NONA$	$\neg\mathbf{E}(A)$

Table 1: Deontic notions as expectations

Definition 2.12. Goal achievement *Given an instance* $\mathcal{S}_{\mathbf{HAP}}$ *of a contract specification* $\mathcal{S} = \langle KB_S, \mathcal{IC}_S \rangle$ *and a goal* $\mathcal{G}$*, iff there exists an* **EXP** *that is an abductive explanation of* $\mathcal{S}_{\mathbf{HAP}}$ *for* $\mathcal{G}$*, and it is* $\neg$-consistent*,* **E**-consistent *and* **HAP**-fulfilled*, we say that* $\mathcal{G}$ *is* achieved w.r.t. **EXP** *(and we write* $\mathcal{S}_{\mathbf{HAP}} \models_{\mathbf{EXP}} \mathcal{G}$*). Given an instance* $\mathcal{S}_{\mathbf{HAP}}$ *and a goal* $\mathcal{G}$*, we say that* $\mathcal{G}$ *is* achieved *if* $\exists$**EXP** *such that* $\mathcal{G}$ *is achieved w.r.t.* **EXP**.

In the remainder of this article, when we simply say that a history **HAP** is compliant to a contract specification $\mathcal{S}$, we will mean that **HAP** is compliant to $\mathcal{S}$ with respect to the goal *true*. We will 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.

2.3 Expectations and deontic operators

In this section, we recall the mapping from deontic operators (obligation, permission, prohibition) to the expectations of the $\mathcal{S}$CIFF framework, proposed in [AGL+06].

Such a mapping can be used to attribute a deontic meaning to $\mathcal{S}$CIFF-based contract specifications.

The mapping is shown in Tab. 1.

The first line of the table proposes a correspondence between the notion of prohibition (which requires an action not to be performed) and ours of negative expectation (which requires an event not to belong to the history).

In fact, the correspondence is more apparent looking at Def. 2.11, which requires, for a set of expectation to be fulfilled, the absence, in the history of events, of any event matching a negative expectation. This definition resembles closely the reduction of the prohibition operator proposed by [Mey88], where "it is forbidden to perform (an action) α in (a state) σ iff one performs α in σ one gets into trouble" (in that paper, "trouble" means an "undesirable state of affairs"; which is a goode description of our state of violation).

Reasoning in a similar way, it is possible to notice a correspondence between the notion of obligation (which requires an action to be performed) and ours of positive expectation (which requires an event to belong to the history), as shown in the second line in Tab. 1.

Moreover, since a negative expectation $\mathbf{EN}(A)$ has to be read as *it is expected not A* (i.e., it is a shorthand for $\mathbf{E}(\textit{not } A)$), its (explicit) negation, $\neg\mathbf{EN}(A)$, corresponds to permission of A. Finally, due to the logical relations among obligation, prohibition and permission discussed in [Sar04], the fourth line of Table 1 shows how to map permission of a negative action.

[AGL$^+$06] provides a formal support of this mapping, based on a correspondence between the Kripke semantics of deontic operators and the declarative semantics of the $\mathcal{S}$CIFF frameworks.

2.4 Sample contract specification

In this section, we intend to demonstrate how to specify, inside the $\mathcal{S}$CIFF framework, contracts that may result too intricate for a representation based on other formalisms, such as finite state machines, coloured Petri nets, or AUML diagrams. 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 a contract, which makes use of expressive combinations of **E**, **EN**, and **H** predicates, CLP constraints and predicates defined in the KB_S. With $\mathcal{S}$CIFF we give a faithful representation of such a contract, 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.

2.4.1 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 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 a specifications of them in the $\mathcal{S}$CIFF framework. ICs are reported in Spec. 2.1, and the KB_S is reported in Spec. 2.2. We chose a set of clauses about bill and complaint handling:

1. After sending a phone bill to a customer, *telco* cannot send requests for payment before a pre-defined amount of time (call it $TWait$ has passed;

2. after $TWait$, either the customer has paid for the bill, or filed a complaint, or *telco* is allowed to send a request for payment;

3. after receiving a legitimate request for payment, either the customer pays for the bill, or *telco* is allowed to de-activate the carrier, after a further $TWait$;

4. if, upon receiving a request for payment, the customer pays by *TWait*, *telco* is not allowed to de-activate the carrier;

5. if a customer files, by *TWait*, an admissible complaint about a received bill, the customer is no longer expected to pay for it, and *telco* is not allowed to request a payment.

2.4.2 $\mathcal{S}$CIFF specification of the contract

Spec. 2.1 contains five ICs: roughly speaking, the first three describe in general what is the expected behaviour of *telco*, regarding bill handling, whereas the last two 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*, where *TWait* is the amount of time defined by the *default_wait* predicate in the KB_S.

- by [$IC2$], after *telco* sends a bill at time *T1*, one of the following 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* . We shall notice that all complaints that *C* possibly sends after the deadline (*T1*+ *TWait*) will not have an impact on the state of affairs in these procedures, since they will not match with any expectation;

- by [$IC3$], if *telco* sent a bill, and later a request for payment at a time in which it was not expected not 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. Notice that [$IC4$] fires independently of *telco* actually having the right to send a request for payments;

- by [$IC5$], after *C* complains about some part of the bill (*Partl_Amnt*), he is no longer expected to pay the bill *Bill_Amnt*.

In the KB_S part of the $\mathcal{S}$CIFF program, shown in Specs. 2.2, 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*. The predicate *member/2* used by *is_admissible_complaint/2* is predefined in most Prolog distributions; this example in particular uses the implementation that comes together with [SIC06].

Specification 2.1 $\mathcal{IC}_S$ in the contract between *telco* (T) and a customer (C).

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

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

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

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

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

Specification 2.2 KB_S in the contract between *telco* and a customer.

KB_S :

$\quad society\_goal.$
$\quad default\_wait(10).$

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

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

3 Contract verification

In the following, we describe two types of verification supported by the $\mathcal{S}$CIFF framework: in Sect. 3.1, a verification that the parties involved in a contract are interacting according to it, and in Sect. 3.2, a formal verification of whether a contract enjoys some properties.

3.1 Run-time verification

The run-time verification of contracts specified the $\mathcal{S}$CIFF language is performed by means of an abductive proof procedure, called itself $\mathcal{S}$CIFF [AGL+05]. We first recall the $\mathcal{S}$CIFF proof procedure, and then show its behaviour on samples interactions regulated by the contract described in Sect. 2.4.

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

Since the $\mathcal{S}$CIFF language and its declarative semantics are closely related with those of the IFF abductive framework [FK97], the $\mathcal{S}$CIFF proof procedure has also been inspired by the IFF proof procedure. $\mathcal{S}$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. 2.11).

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 tuple

$$T \equiv \langle R, CS, PSIC, \mathbf{PEND}, \mathbf{HAP}, \mathbf{FULF}, \mathbf{VIOL} \rangle. \tag{9}$$

We partition the set of expectations **EXP** into the fulfilled (**FULF**), violated (**VIOL**), and pending (**PEND**) 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
- **HAP** is the history of happened events, represented by a set of events, plus a *closed*(**HAP**) boolean attribute.

If one of the elements of the tuple is *false*, then the tuple is the special node *false*, without successors.

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 integrity constraints $\mathcal{IC}_S$, and an initial history $\mathbf{HAP}^i$, we build the first node in the following way:

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

with *closed(*$\mathbf{HAP}^i$*) = false.* The other nodes are obtained by applying the transitions defined in the next section, until no further transition can be applied.

Definition 3.1. *Given an instance $\mathcal{S}_{\mathbf{HAP}^i}$ of a contract specification $\mathcal{S} = \langle KB_S, \mathcal{IC}_S \rangle$ 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 $\langle \{G\}, \emptyset, \mathcal{IC}_S, \emptyset, \mathbf{HAP}^i, \emptyset, \emptyset \rangle$ has at least one leaf node*

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

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

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

From a non-failure leaf node N, answers (called *expectation answers*) can be extracted in a similar way to the IFF proof procedure. 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 [JM94] (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. 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 non-failure node can be selected (if there is one).

Definition 3.2. *Let $\sigma = \sigma'|_{vars(G)}$ be the restriction of σ' to the variables occurring in the initial goal $\mathcal{G}$. Let $\Delta_N = (\mathbf{FULF}_N \cup \mathbf{PEND}_N)\sigma'$. The pair (Δ_N, σ) is the* expectation answer *obtained from the node N.*

3.1.2 $\mathcal{S}$CIFF properties

In the following, we state the most significant formal properties of the $\mathcal{S}$CIFF proof procedure. For the proofs, the interested reader can refer to [ACG+06d].

Termination Termination is proven, as for SLD resolution [AB91], 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 [Xan03]. 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 contract specification, the reader can refer to [Xan03].

Theorem 3.3 (Termination of $\mathcal{S}$CIFF)**.** *Let $\mathcal{G}$ be a query to a contract $\mathcal{S} = \langle KB_S, \mathcal{IC}_S \rangle$, where KB_S, $\mathcal{IC}_S$ and $\mathcal{G}$ are acyclic w.r.t. some level mapping, and $\mathcal{G}$ and all implications in $\mathcal{IC}_S$ 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 when no other transitions can be applied, and*
- *non-happening has higher priority than other transitions,*

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

Soundness 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 of terms.

Theorem 3.4 (Soundness of $\mathcal{S}$CIFF). *Given a contract instance $\mathcal{S}_{\mathbf{HAP}^f}$, if*

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

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

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

Completeness 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 3.5. *Given a contract instance $\mathcal{S}_{\mathbf{HAP}}$, a (ground) goal G, for any ground set $\mathbf{EXP}$ such that $\mathcal{S}_{\mathbf{HAP}} \models_{\mathbf{EXP}} \mathcal{G}$ then $\exists \mathbf{EXP'}$ such that $\mathcal{S}_{\emptyset} \vdash_{\mathbf{EXP'}}^{\mathbf{HAP}} G$ with an expectation answer $(\mathbf{EXP'}, \sigma)$ such that $\mathbf{EXP'}\sigma \subseteq \mathbf{EXP}$.*

3.1.3 Runtime verification examples

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.

$$\begin{array}{l}
\mathbf{H}(\ tell(\ telco,\ c,\ phone\_bill(390512093086, 145886, 205),\ 19). \\
\mathbf{H}(\ tell(\ telco,\ c,\ request\_payment(390512093086, 145886, 205),\ 33). \\
\mathbf{H}(\ tell(\ c,\ telco,\ pay(390512093086, 145886, 205, 1674521),\ 37).
\end{array} \tag{10}$$

This sequence of events (10) 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 else *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 contract, and it gives *telco* explicit right to de-activate the carrier 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 No. *145886*.

As the second example shows (11), 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 contract, 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.

$$\begin{array}{l}\left.\begin{array}{l}\mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{phone\_bill}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 19). \\ \mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{request\_payment}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 33). \\ \mathbf{H}(\ \mathit{tell}(\ c,\ \mathit{telco},\ \mathit{pay}(\mathit{390512093086}, \mathit{145886}, \mathit{205}, \mathit{1674521}),\ 37).\end{array}\right] (10) \\ \mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{de\_activate}(\mathit{390512093086}, \mathit{reason}(\mathit{145886})),\ 38).\end{array} \tag{11}$$

Let us consider a third example, 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 contract, 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* looses 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 (12).

$$\begin{array}{l}\mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{phone\_bill}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 19). \\ \mathbf{H}(\ \mathit{tell}(\ c,\ \mathit{telco},\ \mathit{complain}(\mathit{390512093086}, \mathit{145886}, \mathit{150}),\ 33). \\ \mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{request\_payment}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 34). \\ \mathbf{H}(\ \mathit{tell}(\ c,\ \mathit{telco},\ \mathit{pay}(\mathit{390512093086}, \mathit{145886}, \mathit{205}, \mathit{1674521}),\ 37).\end{array} \tag{12}$$

In the last example, *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. Moreover, the complaint is judged admissible (in our example, shown with the *is_admissible_complaint* predicate). As a consequence, if *telco* sends C a request for payment (13), it causes a contract 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 [$IC5$].

$$\begin{array}{l}\mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{phone\_bill}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 19). \\ \mathbf{H}(\ \mathit{tell}(\ c,\ \mathit{telco},\ \mathit{complain}(\mathit{390512093086}, \mathit{145886}, \mathit{150}),\ 24). \\ \mathbf{H}(\ \mathit{tell}(\ \mathit{telco},\ c,\ \mathit{request\_payment}(\mathit{390512093086}, \mathit{145886}, \mathit{205}),\ 34).\end{array} \tag{13}$$

3.2 Design-time property verification

In order to verify contract properties, we have developed an extension of the $\mathcal{S}$CIFF proof-procedure, called g-$\mathcal{S}$CIFF [ACG+06c]. In the following, we briefly recall g-$\mathcal{S}$CIFF, and then show how it can be used to refute a formal property that the contract described in Sect. 2.4 does not enjoy.

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

Besides verifying whether a history is compliant to a contract, g-$\mathcal{S}$CIFF is able to generate a compliant history, given a contract. This is achieved by (*i*) considering **H** events as abducibles and (*ii*) adding a new transition to $\mathcal{S}$CIFF, which, when an expectation is added to the set of expectations, generates an event that fulfills it. g-$\mathcal{S}$CIFF has been proved sound [ACG+05], which means that the histories that it generates (in case of success) are guaranteed to be compliant to the interaction contracts while entailing the goal. Note that the histories generated by g-$\mathcal{S}$CIFF are in general not only a collection of ground events, like the **HAP** sets given as an input to $\mathcal{S}$CIFF. They can, in fact, contain variables, which means that they represent *classes* of event histories.

In order to use g-$\mathcal{S}$CIFF for verification, we express the property to be verified as a conjunction of literals. If we want to verify if a formula f is a property of a contract $\mathcal{P}$, we express the contract in our language and $\neg f$ as a g-$\mathcal{S}$CIFF goal. Then either:

- g-$\mathcal{S}$CIFF returns success, generating a history **HAP**. Thanks to the soundness of g-$\mathcal{S}$CIFF, **HAP** entails $\neg f$ while being compliant to $\mathcal{P}$: f is not a property of $\mathcal{P}$, **HAP** being a counterexample; or
- g-$\mathcal{S}$CIFF returns failure, suggesting that f is a property of $\mathcal{P}$.[3]

3.2.2 Design-time property verification example

In this section, we show the refutation, by means of g-$\mathcal{S}$CIFF, of a simple property of the contract described in Sect. 2.4.2. For simplicity, we will not show the details related to the management of restrictions and defined predicates.

The property is the following: if a phone bill is sent, then the customer will pay for it. Using our formalism for events, the property can be written as follows:

$$\begin{aligned} &\mathbf{H}(\ \mathit{tell}(\ T,\ C,\ \mathit{phone\_bill}(N,I,A),D),T_b) \\ \rightarrow&\mathbf{H}(\ \mathit{tell}(\ C,\ T,\ \mathit{pay}(N,I,A,R),D),T_p) \end{aligned} \tag{14}$$

The negation of the property is:

$$\begin{aligned} &\mathbf{H}(\ \mathit{tell}(\ T,\ C,\ \mathit{phone\_bill}(N,I,A),D),T_b) \\ \wedge\neg&\mathbf{H}(\ \mathit{tell}(\ C,\ T,\ \mathit{pay}(N,I,A,R),D),T_p) \end{aligned} \tag{15}$$

[3]If we had a completeness result for g-$\mathcal{S}$CIFF, this would indeed be a proof and not only a suggestion.

Therefore, a history that entails Eq. (15) is a counterexample of the property that we want to verify. To try and find such a history, we write the following g-$\mathcal{S}$CIFF goal:

$$\begin{array}{c} \mathbf{E}(\ \mathit{tell}(\ T,\ C,\ \mathit{phone\_bill}(N, I, A), D), T_b) \\ \wedge \mathbf{EN}(\ \mathit{tell}(\ C,\ T,\ \mathit{pay}(N, I, A, R), D), T_p) \end{array} \tag{16}$$

and run g-$\mathcal{S}$CIFF. A history that achieves the goal will necessarily include events that are expected to happen, and not include events that are expected not to happen, in the goal.

g-$\mathcal{S}$CIFF imposes the first expectation of the goal,

$$\mathbf{E}(\ \mathit{tell}(\ T,\ C,\ \mathit{phone\_bill}(N, I, A), D), T_b),$$

which generates the following event:

$$\mathbf{H}(\ \mathit{tell}(\ T,\ C,\ \mathit{phone\_bill}(N, I, A), D), T_b)$$

which in turn, due to the first IC, generates the expectation

$$\mathbf{EN}(\mathit{tell}(T, C, \mathit{request\_payment}(N, I, A), D), T2)$$

and, due to the second, one of

$$\begin{array}{c} \mathbf{E}(\mathit{tell}(C, T, \mathit{pay}(N, I, A, PR), D), T2), \\ \mathbf{E}(\mathit{tell}(C, T, \mathit{complain}(N, I, PA), D), T3), \\ \neg\mathbf{EN}(\mathit{tell}(T, C, \mathit{request\_payment}(N, I, A), D), T4). \end{array}$$

The **E**-consistency requirement (Def. 2.10) rules out the first alternative, because of the negative (**EN**) expectation imposed by the goal (see Eq. (16)); so the second branch is explored, and the event

$$\mathbf{H}(\mathit{tell}(C, T, \mathit{complain}(N, I, PA), D), T3)$$

is generated.

Due to the fifth IC, the following expectations are generated:

$$\neg\mathbf{E}(\mathit{tell}(C, T, \mathit{pay}(N, I, PA, PR), D), T3)$$

and

$$\mathbf{EN}(\mathit{tell}(T, C, \mathit{request\_payment}(N, I, \mathit{Bill\_Amnt}), D), T4)$$

and finally g-$\mathcal{S}$CIFF terminates and returns success, with the history

$$\begin{array}{l} \mathbf{HAP} = \{\mathbf{H}(\mathit{tell}(T, C, \mathit{phone\_bill}(N, I, A), D), T_b), \\ \qquad\qquad \mathbf{H}(\mathit{tell}(C, T, \mathit{complain}(N, I, PA), D), T3)\} \end{array}$$

Thanks to the soundness of g-$\mathcal{S}$CIFF, **HAP** is a counterexample of the property that we wanted to prove, and it is also compliant to the contract. Thus, it shows that the contract does not enjoy the property. In particular, it shows that a customer can avoid being expected to pay by filing a complaint.

4 Rule Mark-Up

In [ACG$^+$06a], we propose an architecture and a formal framework that enables web services to reason on publicly available $\mathcal{S}$CIFF-based specifications: in particular, it is possible for a web service to verify whether it can interact with another and achieve a goal. We believe that an interested party could fruitfully perform such a step before agreeing with another party on a contract. Obviously, this requires a formalism that makes it practical to exchange $\mathcal{S}$CIFF-based specifications.

RuleML [AST05] is the perfect mark-up language for exchanging rules on the web, so our choice has been easy. RuleML 0.9 contains mark-ups for expressing important concepts of the $\mathcal{S}$CIFF proof-procedure. In particular, $\mathcal{S}$CIFF is a rule engine able to distinguish and use both backward and forward rules. Backward rules are used to plan, reason upon events, perform proactive reasoning. Forward rules are used for reactive reasoning, to quickly perform actions in response to occurred events. Both are seamlessly integrated in $\mathcal{S}$CIFF. RuleML 0.9 contains a *direction* attribute that can be attached to rules. Being based on abduction, $\mathcal{S}$CIFF can deal both with negation as failure and negation by default, that have an appropriate tagging in RuleML. In this work, we only used standard RuleML syntax; in future work we might be interested in distinguishing between defined and abducible predicates, or between expectations and events.

$\mathcal{S}$CIFF was implemented in SICStus Prolog: SICStus contains an implementation of the PiLLoW library [GH01], which makes it easy to perform http requests, as well as implementing services on the web. Finally, SICStus contains an XML parser, which allowed us to easily implement the RuleML parser. The RuleML parser is freely available on the $\mathcal{S}$CIFF web site [SCI05].

5 Related work

The reduction of deontic concepts such as obligations and prohibitions has been the subject of several past works: notably, by [And58] (according to which, informally, A is obligatory iff its absence produces a state of violation) and by [Mey88] (where, informally, an action A is prohibited iff its being performed produces a state of violation). These two reductions strongly resemble our definition of fulfillment (Def. 2.11), which requires positive (resp. negative) expectations to have (resp. not to have) a corresponding event.

Several papers discuss "sub-ideal" situations, i.e., how to manage situations in which some of the norms are not respected.

For instance, [vT99] shows 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. In particular, given the specification of a normative system (as a set of formulae which tell when a norm is violated) and a state of affairs, they define a minimal (with respect to inclusion) set of norms such that the violation of those norms is consistent with the specification and the state of affairs. The SOCS social framework, currently, only distinguishes between empty and non-empty sets of violations, and does not define minimal sets. However, it would be possible to do so by taking the minimal, with respect to inclusions, among the sets of expectations which are consistent with a social specification and a history, but possibly not fulfilled by the history. This will probably be our approach when we tackle the management of violations (by means of sanctions and recovery procedures) in future work.

[PS96] propose 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. They do so by introducing a new operator $O_B(A)$, meaning that A is obligatory given the sub-ideal context B. The semantics of this operator is of Kripke type, but it differs to the standard modal logic because of the accessibility relation: in that work, the accessible worlds are the best alternatives, given the truth of B. In the "main stream" of our research, we do not support contrary-to-duty obligations. However, we proposed a modified version of our framework [ADG+04] , which provides a simplified language and does support alternative obligations at different levels of priority; a further step could be to integrate priority levels in the main SOCS social framework.

Deontic operators have been used not only at the social level, but also at the agent level. Notably, in IMPACT ([AOR+99, ESP99]), agent programs may be used to specify what an agent is obliged to do, what an agent may do, and what an agent cannot do on the basis of deontic operators of Permission, Obligation and Prohibition (whose semantics does not rely on a Deontic Logic semantics). In this respect, the IMPACT and SOCS social models have similarities even if their purpose and expressivity are different. The main difference is that the goal of agent programs in IMPACT is to express and determine by its application the behavior of a single agent, whereas the SOCS social model goal is to express rules of interaction and norms, that instead cannot really determine and constrain the behavior of the single agents participating to a society, since agents are autonomous.

Governatori [Gov05] uses defeasible logics with deontic operators of Obligation and Permission to define contracts. He proposes the introduction in RuleML of new tags for identifying obligations and permission. In $\mathcal{S}$CIFF, we usually do not use explicit permission, because everything is allowed by default; we typically state explicitly when an action is expected not to happen **EN**. There are connections between **EN** and $\neg P$ of deontic logics (studied in [AGL+06]), so we might use the same tags proposed by Governatori (e.g., we might use `<neg><Permission>` to represent **EN**).

Governatori [Gov05] also introduces an operator $\otimes$ to address recovery from violation. For example, $A \Rightarrow OB \otimes OC$ means that A implies that B is obligatory; however in case OB is violated, C becomes obligatory. In $\mathcal{S}$CIFF, recovery expectations can be inserted as an alternative in each of the rules: $A \Rightarrow OB \otimes OC$ could be written in $\mathcal{S}$CIFF as $\mathbf{H}(A) \rightarrow \mathbf{E}(B) \vee \mathbf{E}(C)$. Interestingly, Governatori [Gov05] proposes also an inference rule that derives recovery rules from the other rules of the contract (from $A \rightarrow OB$ and $\neg B \rightarrow OC$ derives $A \rightarrow OB \otimes OC$); this is an interesting line of research that we plan to apply in future work also to $\mathcal{S}$CIFF.

In cite [GM05, GM06] Governatori and Milosevic discuss the need for contract verification and contract monitoring to check how parties fulfil their policies. Both these issues are addressed by the adoption of a formal specification language for contracts. The system they propose, and their Business Contract Language (BCL) in particular, is based on the formalism for the representation of contrary-to-duty obligations (CTDs), i.e., obligations in force after some other obligations have been violated. The formal representation for contracts they adopt is based upon a propositional logic language, with the deontic operators of obligation, permission and contrary-to-duty. Each condition or policy of a contract is represented by a rule where the *antecedent* is a literal or a modal literal (built with the deontic operators of permission and obligation, possibly negated) and the *conclusion* of the rule is a CTD expression. Contract analysis is then done by reducing a contract to a normal form, where all the contract conditions that can be generated/derived from the given specification have been made explicit. The procedure to generate normal forms is expressed in terms of inference rules, which merge two rules in a new clause through the violations of conditions (e.g., when the former rule mentions an obligation $O\ A$ in its conclusion and the latter rule has the negation $\neg A$ in its antecedent, then their conclusions are composed in order to build a CDT formula for A). Normal forms are then a sort of *partial evaluation* of specification rules, in the logic of violation, aiming at producing rules with CDT formulas in their conclusions which summarize all the possible violations and recovery actions implicitly specified by the original (logic) representation of a contract. On generated normal forms they can therefore detect conflicts arising from, e.g., obligation of A and $\neg A$, or occurrence of A and $\neg A$ in conclusions without any CTD for A neither $\neg A$.

Although we do not have a language supporting CDTs, our language is first-order, and supports the deontic operators of permission and obligation (and their negation as discussed in [AGL$^+$06]). In our approach, $\mathcal{S}$CIFF is exploited at run-time for contract monitoring (e.g., conflicts and contradictions are detected at run-time by the notions of E-consistency and $\neg$-consistency), and more general contract properties (beside the absence of conflicts) can be also statically verified by g-$\mathcal{S}$CIFF. g-$\mathcal{S}$CIFF, in particular, generates all the possible *compliant* histories which satisfy a given goal, and a contract specified in the $\mathcal{S}$CIFF language. Each generated history can be considered as a set of obligations in the approach of Governatori and Milosevic [GM06], since g-$\mathcal{S}$CIFF turns obligations into events.

An interesting extension would be to equip the $\mathcal{S}$CIFF language with CTDs expressions, to occur in the head of ICs. This is subject for future work.

[Bv03] discuss how a normative system can be seen as a normative agent, equipped with mental attitudes, about which other agents can reason, choosing either to fulfill their obligations, or to face the possible sanctions. Conceptually, the social infrastructure in the SOCS model could be viewed as an agent, whose knowledge base is the society specification, whose mental attitude is a set of expectations, and whose reasoning process is the $\mathcal{S}$CIFF proof procedure.

[BDDM04] investigate the deontic logic of deadlines by introducing an operator $O(\rho \leq \delta)$, which means, intuitively, that the action ρ ought to be brought about before (or at the same time) another event δ happens. They model time by means of the CTL temporal logic. We can express a similar concept by means of an integrity constraints $\mathbf{H}(\delta, T_\delta) \rightarrow \mathbf{E}(\rho, T_\rho) \wedge T_\rho \leq T_\delta$, which says that, if δ has happened, than ρ is expected to have happened before (or at the same time).

The $\mathcal{S}$CIFF framework can capture, in a computational setting, the concept of (conditional) obligation with deadline presented by [DMDW02], with an explicit mapping of time. Dignum *et al.* write: `Oa(r<d|p)` to state that if the precondition `p` becomes valid, the obligation becomes active. The obligation expresses the fact that `a` is expected to bring about the truth of `r` before a certain condition `d` holds.

For instance, if we have:

$$\begin{aligned}
&p = \mathbf{H}(\mathit{tell}(S, a, \mathit{request}(G), D, T))\\
&r = \mathbf{H}(\mathit{tell}(a, S, \mathit{answer}(G), D, T')), T' > T\\
&d = T' > T + 2
\end{aligned}$$

we can map `Oa(r<d|p)` into a IC:

$$\begin{aligned}
&\mathbf{H}(\mathit{tell}(S, a, \mathit{request}(G), D), T) \rightarrow\\
&\mathbf{E}(\mathit{tell}(a, S, \mathit{answer}(G), D), T'), T' > T, T' \leq T + 2.
\end{aligned}$$

There have been many works using the Event Calculus (EC) for the purpose of reasoning over the effects of events, that are very close to this paper. In particular, our work is very related to the work in [FSSB05]. In [FSSB05] the authors have been principally concerned with the representation of contracts and their normative state in particular, in terms of obligation, power and permission. The effects of contract events on the normative state of a contract are specified using an XML formalisation of the Event Calculus. This representation may be used to track the state of the agreement, according to a narrative of contract events similar to our concept of History.

Like in [FSSB05], $\mathcal{S}$CIFF can be seen as a generic language for expressing backward and forward rules and reasoning about (conformance) properties of a specific where the representation of contracts is just one application.

Differently from [FSSB05] In this work we show that being able to describe contracts as logical theories is extremely useful not only for tracking, but also for for proving general or specific properties of the contracts by using the same formalism . A similar approach is used in [ASP03] by using a formalization in terms of transition systems and model checking techniques.

6 Conclusions

In this paper, we proposed the use of the $\mathcal{S}$CIFF framework, originally developed for agent interaction protocols, to specify and verify business contracts. We supported intuitively our proposal by showing a deontic reading of $\mathcal{S}$CIFF specifications. We gave the specification of sample business contract clauses in the $\mathcal{S}$CIFF language.

We also demonstrated how verification is performed in the $\mathcal{S}$CIFF framework: in particular, run-time verification by means of the $\mathcal{S}$CIFF proof procedure, and design-time property verification with the g-$\mathcal{S}$CIFF proof procedure. We also showed how $\mathcal{S}$CIFF rules can be encoded in RuleML, in order to be possibly exchanged to enable potential contract parties to reason on contracts in advance.

Future work will be devoted to experiment with the $\mathcal{S}$CIFF framework on real-world contracts, to test both the expressiveness of the $\mathcal{S}$CIFF language and the effectiveness of the proof procedures used for verification. We are also working on a formal completeness result (possibly for restricted cases) for g-$\mathcal{S}$CIFF.

Acknowledgments

This work has been partially supported by the MIUR PRIN 2005 projects *Specification and verification of agent interaction protocols* and *Vincoli e preferenze come formalismo unificante per l'analisi di sistemi informatici e la soluzione di problemi reali*, and by the MIUR FIRB project *Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet.*

References

[AB91] Krzysztof R. Apt and Marc Bezem. Acyclic programs. *New Generation Computing*, 9(3/4):335–364, 1991.

[AB94] Krzysztof R. Apt and Roland N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994.

[ACG+05] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. On the automatic verification of interaction protocols using *g*-$\mathcal{S}$CIFF. Technical Report DEIS-LIA-04-004, University of Bologna (Italy), 2005. LIA Series no. 72.

[ACG+06a] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, Marco Montali, and Paolo Torroni. Policy-based reasoning for smart web service interaction. In *Proceedings of the 1st International Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS*

2006), volume 196 of *CEUR Workshop Proceedings*, pages 87–102, Seattle, WA, USA, August 2006.

[ACG$^+$06b] 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.

[ACG$^+$06c] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Security protocols verification in abductive logic programming: a case study. In Ogus Dikenelli, Marie-Pierre Gleizes, and Alessandro Ricci, editors, *ESAW 2005 Post-proceedings*, number 3963 in LNAI, pages 106–124, Kusadasi, Aydin, Turkey, 2006. Springer-Verlag.

[ACG$^+$06d] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: the $\mathcal{S}$CIFF proof-procedure. Technical Report DEIS-LIA-06-001, University of Bologna (Italy), March 2006. LIA Series no. 75.

[ADG$^+$04] 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.

[AGL$^+$05] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. The $\mathcal{S}$CIFF abductive proof-procedure. In *Proceedings of the 9th National Congress on Artificial Intelligence, AI*IA 2005*, volume 3673 of *Lecture Notes in Artificial Intelligence*, pages 135–147. Springer-Verlag, 2005.

[AGL$^+$06] Marco Alberti, Marco Gavanelli, Evelina Lamma, Paola Mello, Giovanni Sartor, and Paolo Torroni. Mapping deontic operators to abductive expectations. *Computational and Mathematical Organization Theory*, 12(2–3):205 – 225, October 2006.

[AMB00] Grigoris Antoniou, Michael J. Maher, and David Billington. Defeasible logic versus logic programming without negation as failure. *J. Log. Program.*, 42(1):47–57, 2000.

[And58] A. Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, 67:100–103, 1958.

[AOR$^+$99] 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.

[ASP03] Alexander Artikis, Marek J. Sergot, and Jeremy Pitt. An executable specification of an argumentation protocol. In *ICAIL*, pages 1–11, 2003.

[AST05] Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors. *Rules and Rule Markup Languages for the Semantic Web, First International Conference, RuleML 2005, Galway, Ireland, November 10-12, 2005, Proceedings*, volume 3791 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[BDDM04] Jan Broersen, Frank Dignum, Virginia Dignum, and John-Jules Ch. Meyer. Designing a deontic logic of deadlines. In Alessio Lomuscio and Donald Nute, editors, *DEON*, volume 3065 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2004.

[Bür94] H.J. Bürckert. A resolution principle for constrained logics. *Artificial Intelligence*, 66:235–271, 1994.

[Bv03] Guido Boella and Leendert W. N. van der Torre. Attributing mental attitudes to normative systems. 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 942–943, Melbourne, Victoria, July 14–18 2003. ACM Press.

[Cla78] K. L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[DMDW02] 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.

[DS98] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.

[ESP99] T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, March 1999.

[FK97] 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.

[FSSB05] Andrew D. H. Farrell, Marek J. Sergot, Mathias Sallé, and Claudio Bartolini. Using the event calculus for tracking the normative state of contracts. *Int. J. Cooperative Inf. Syst.*, 14(2-3):99–129, 2005.

[GH01] Daniel Cabeza Gras and Manuel V. Hermenegildo. Distributed WWW programming using (Ciao-)Prolog and the PiLLoW library. *Theory and Practice of Logic Progr.*, 1(3):251–282, 2001.

[GLC99] Benjamin N. Grosof, Yannis Labrou, and Hoi Y. Chan. A declarative approach to business rules in contracts: courteous logic programs in xml. In *ACM Conference on Electronic Commerce*, pages 68–77, 1999.

[GM05] Guido Governatori and Zoran Milosevic. Dealing with contract violations: formalism and domain specific language. In *EDOC*, pages 46–57. IEEE Computer Society, 2005.

[GM06] Guido Governatori and Zoran Milosevic. A formal analysis of a business contract language. *International Journal of Cooperative Information Systems*, 15(4):659–685, 2006.

[Gov05] Guido Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216, 2005.

[JM94] J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.

[Kun87] K. Kunen. Negation in logic programming. In *Journal of Logic Programming*, volume 4, pages 289–308, 1987.

[Mey88] 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.

[MGL$^+$04] Z. Milosevic, S. Gibson, P. F. Linington, J. Cole, and S. Kulkarni. On design and implementation of a contract monitoring facility. In *Proceedings of the First International Workshop on Electronic Commerce (WEC'04)*, pages 62–70, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[Mil95] Zoran Milosevice. *Enterprise aspects of open distributed systems.* PhD thesis, Computer Science Department, The University of Queensland, October 1995.

[PS96] Henry Prakken and Marek Sergot. Contrary-to-duty obligations. *Studia Logica*, 57(1):91–115, 1996.

[Sar04] Giovanni Sartor. *Legal Reasoning*, volume 5 of *Treatise*. Kluwer, Dordrecht, 2004.

[SCI05] The *SCIFF* abductive proof procedure, 2005. `http://lia.deis.unibo.it/research/sciff/`.

[SIC06] SICStus prolog user manual, release 3.12.7, October 2006. `http://www.sics.se/isl/sicstus/`.

[SOC05] 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/`.

[vT99] 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.

[Xan03] 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`.