Verifying Compliance of Business Processes with Temporal Answer Sets

Davide D'Aprile[1], Laura Giordano[1], Valentina Gliozzi[2], Alberto Martelli[2], Gian Luca Pozzato[2], and Daniele Theseider Dupré[1]

[1] Dipartimento di Informatica, Università del Piemonte Orientale
{davide.daprile,laura.giordano,dtd}@mfn.unipmn.it
[2] Dipartimento di Informatica, Università di Torino
{gliozzi,mrt,pozzato}@di.unito.it

Abstract. In this paper we provide a framework for the specification and the verification of business processes, which is based on a temporal extension of answer set programming (ASP) and we address the problem of verifying the compliance of business processes to norms. The logical formalism we use, is a combination of Answer Set Programming and Dynamic Linear Time Temporal Logic (DLTL), and allows for a declarative specification of a business process, as well as the specification of norms, by means of a set of temporal rules and a set of temporal constraints. A notion of commitment, borrowed from the social approach to agent communication, is used to capture obligations within norms. Besides allowing for a declarative specification of business processes, the proposed framework can be used for encoding business processes specified in conventional workflow languages. The verification of temporal properties of a business process, expressed by temporal formulas, can be done by encoding bounded model checking techniques in ASP. Verifying compliance of a business process to norms consists, in particular, in verifying that there are no executions of the business process which leave commitments unfulfilled.

1 Introduction

The problem of verifying the compliance of business processes has attracted a lot of interest in recent years. Many organizations (banks, hospitals, public administrations, etc.), whose activities are subject to regulations, are required to justify their behaviors with respect to the norms and to show that the business procedures they adopt conform to such norms. For instance, in the financial domain, the Sarbanes-Oxley Act (commonly named SOX), enacted in 2002 in the USA, describes mandates and requirements for financial reporting, and was proposed in order to restore investors' confidence in capital markets after major accounting scandals. MiFID (Markets in Financial Instruments Directive) is a EU law, effective from 2007, with similar goals, including transparency.

In this paper, we address the problem of automatic verification of business process compliance and, to this purpose, we propose a framework for the specification and the verification of business processes, which is based on a temporal

extension of answer set programming (ASP [12]). The choice of a logical formalism for the specification of business processes has the advantage of allowing a declarative specification of business processes and web services, which has been advocated in recent literature [29, 27, 24], as opposed to the more rigid transition based approach. A declarative specification of a process is, generally, more concise than transition based specification as it abstracts away form rigid control-flow details and does not require the order among the actions in the process to be rigidly defined. A further advantage of logical formalisms is that the computational mechanisms of the underlying logic can then be exploited in the verification of business process properties.

The framework for the specification and the verification of business processes proposed in this paper, is based on a temporal extension of Answer Set Programming (ASP [12]) defined in [17], which combines Answer Set Programming with a temporal logic, namely Dynamic Linear Time Temporal Logic (DLTL) [22]. DLTL extends propositional temporal logic of linear time (LTL) with regular programs of propositional dynamic logic, that are used for indexing temporal modalities. The language in [17] is a temporal action language, which allows for the specification of atomic actions by means of their effects and preconditions, as well as for the specification of complex actions and general temporal constraints to specify the wanted interactions among the tasks (and, under this respect, our approach to business process specification has similarities with that of *DecSerFlow* [29]). Besides allowing for a declarative specification of business processes, this language is well suited for encoding processes specified workflow languages.

As concerns compliance verification, to represent the obligations which can be enforced by the application of norms, we make use of the notion of *commitment*, which is borrowed from the area of multi-agent communication [27, 11, 20, 16, 5]. We show that the temporal language is well suited to model norms as directional rules which generate commitments, and that defeasible negation of ASP can be exploited to capture exceptions to the norms, by allowing norms to be defeasible. Given the specification of a business process in temporal ASP and the specification of norms as a set of (defeasible) causal laws generating obligations (commitments), the problem of compliance verification can be given a logical characterization. It consists in verifying that there are no executions of the business process which leave some commitment unfulfilled.

For the verification of the business process properties and, in particular, for compliance verification, we exploit Bounded Model Checking [6]. In particular, we exploit an approach developed in [17] for DLTL bounded model checking in ASP, which extends the approach for Bounded LTL Model Checking with Stable Models that has been developed in [21].

2 A temporal extension of ASP

We shortly recall the temporal action language introduced in [17], which is based on a temporal extension of Answer Set Programming (ASP).

2.1 Temporal action language

The action language is based on the Dynamic Linear Time Temporal Logic (DLTL) [22]. DLTL extends LTL by allowing the *until* operator $\mathcal{U}^\pi$ to be indexed by a program π, an expression of Propositional Dynamic Logic: π can be any regular expression built from atomic actions using sequence (;), non-deterministic choice (+) and finite iteration ($*$). The usual LTL modalities $\Box$ (always), $\Diamond$ (eventually), $\bigcirc$ (next), and $\mathcal{U}$ (until) can be defined from $\mathcal{U}^\pi$ as well as the new temporal modalities $[\pi]$ and $\langle\pi\rangle$.

Informally, a formula $[\pi]\alpha$ is true in a world w of a linear temporal model if α holds in all the worlds of the model which are reachable from w through any execution of the program π. A formula $\langle\pi\rangle\alpha$ is true in a world w of a linear temporal model if there exists a world of the model reachable from w through an execution of the program π, in which α holds. A formula $\alpha\,\mathcal{U}^\pi\beta$ is true at a world w if "α until β" is true at w on a finite stretch of behavior which is in the linear time behavior of the program π.

A domain description is defined as a set of laws describing the effects of actions as well as their executability preconditions. Atomic propositions describing the state of the domain are called *fluents*. Actions may have direct effects, that are described by action laws, and indirect effects, that capture the causal dependencies among fluents and are described by causal laws. The execution of an action a in a state s leads to a new state s' in which the effect of the action holds. The fluents which hold in s and are not affected by the action a, still hold in s'.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluents*, including the inconsistency, $\bot$. A *simple fluent literal* l is a fluent name f or its negation $\neg f$. We will denote by Lit_S the set of all simple fluent literals. In the language, we also make use of *temporal literals*, that is, literals that are prefixed by the temporal modalities $[a]$, with $a \in \Sigma$, and $\bigcirc$. Their intended meaning is: $[a]l$ holds in a state s if l holds in the state obtained after the execution of action a in s; $\bigcirc l$ holds in a state s if l holds in the state next to s. Lit_T is the set of *temporal fluent literals*: if $l \in Lit_S$, then $[a]l, \bigcirc l \in Lit_T$, where a is an action name (an atomic proposition, possibly containing variables), and $[a]$ and $\bigcirc$ are the temporal operators introduced in the previous section. Let $Lit = Lit_S \cup Lit_T \cup \{\bot\}$. Given a (simple or temporal) fluent literal l, *not* l represents the default negation of l. A (simple or temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*.

The laws are formulated as rules of a temporally extended logic programming language. Rules have the form

$$\Box(t'_0 \; or \ldots or \; t'_h \leftarrow t_1, \ldots, t_m, not \; t_{m+1}, \ldots, not \; t_n) \tag{1}$$

or the form

$$t'_0 \; or \ldots or \; t'_h \leftarrow t_1, \ldots, t_m, not \; t_{m+1}, \ldots, not \; t_n \tag{2}$$

where the t'_i's and the t_i's are either simple fluent literals or temporal fluent literals. While laws of the form (1) can be applied in all states, laws of the form

(2) can only be applied in the initial state. Action laws, causal laws, precondition laws, persistency laws, initial state laws, etc., which are normally used in action theories, can all be defined as instances of (1) and (2).

A *domain description* D is defined as a tuple $(\Pi, \mathcal{C})$, where Π is a set of laws of the form 1 and 2 and $\mathcal{C}$ is a set of *temporal constraints*, i.e. general DLTL formulas.

2.2 Temporal Answer Sets

To define the the semantics of a domain description, we extend the notion of *answer set* [12] to capture the linear structure of temporal models. In the following, we consider the ground instantiation of the domain description Π, and we denote by Σ the set of all the ground instances of the action names in Π.

We define a *(partial) temporal interpretation* as a pair (σ, S), where $\sigma \in \Sigma^\omega$ is a sequence of actions and S is a consistent set of literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ, meaning that l holds in the state obtained by executing $a_1 \ldots a_k$. S is *consistent* iff it is not the case that both $[a_1; \ldots; a_k]l \in S$ and $[a_1; \ldots; a_k]\neg l \in S$, for some l, or $[a_1; \ldots; a_k]\bot \in S$. A temporal interpretation (σ, S) is said to be *total* if either $[a_1; \ldots; a_k]p \in S$ or $[a_1; \ldots; a_k]\neg p \in S$, for each $a_1 \ldots a_k$ prefix of σ and for each fluent name p.

We define the *satisfiability of a simple, temporal or extended literal t in a partial temporal interpretation (σ, S) in the state $a_1 \ldots a_k$*, (written $S, a_1 \ldots a_k \models t$) as follows:

$(\sigma, S), a_1 \ldots a_k \models \top, \qquad (\sigma, S), a_1 \ldots a_k \not\models \bot$

$(\sigma, S), a_1 \ldots a_k \models l$ *iff* $[a_1; \ldots; a_k]l \in S$, for a literal l

$(\sigma, S), a_1 \ldots a_k \models [a]l$ *iff* $[a_1; \ldots; a_k; a]l \in S$ or $a_1 \ldots a_k, a$ is not a prefix of σ

$(\sigma, S), a_1 \ldots a_k \models \bigcirc l$ *iff* $[a_1; \ldots; a_k; b]l \in S$, where $a_1 \ldots a_k b$ is a prefix of σ

$(\sigma, S), a_1 \ldots a_k \models \mathit{not}\ l$ *iff* $(\sigma, S), a_1 \ldots a_k \not\models l$

Observe that $[a]l$ is true in any state of a linear model in which a is not the next action to be executed. The satisfiability of rule bodies and rule heads in a temporal interpretation are defined as usual. A rule $\Box(H \leftarrow \mathit{Body})$ is satisfied in a temporal interpretation (σ, S) if, for all action sequences $a_1 \ldots a_k$ (including the empty one), $(\sigma, S), a_1 \ldots a_k \models \mathit{Body}$ implies $(\sigma, S), a_1 \ldots a_k \models H$.

A rule $H \leftarrow \mathit{Body}$ is satisfied in a partial temporal interpretation (σ, S) if, $(\sigma, S), \varepsilon \models \mathit{Body}$ implies $(\sigma, S), \varepsilon \models H$, where ε is the empty action sequence.

To define the answer sets of Π, we introduce the notion of *reduct* of Π, containing rules of the form: $[a_1; \ldots; a_h](H \leftarrow \mathit{Body})$. Such rules are evaluated in the state $a_1 \ldots a_h$.

Let Π be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$.

Definition 1. *A temporal interpretation (σ, S) is a* temporal answer set of Π *if S is minimal (in the sense of set inclusion) among the S' such that (σ, S') is a partial interpretation satisfying the rules in Π.*

To define answer sets of a program Π containing negation, given a temporal interpretation (σ, S) over $\sigma \in \Sigma^\omega$, we define the *reduct, $\Pi^{(\sigma,S)}$, of Π relative to* (σ, S) extending Gelfond and Lifschitz' transform [13] to compute a different reduct of Π for each prefix $a_1, \ldots, a_h$ of σ.

Definition 2. *The* reduct, $\Pi^{(\sigma,S)}_{a_1,\ldots,a_h}$, of Π relative to (σ, S) and to the prefix $a_1, \ldots, a_h$ of σ *, is the set of all the rules*

$$[a_1; \ldots; a_h](H \leftarrow l_1, \ldots, l_m)$$

such that $H \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ is in Π and $(\sigma, S), a_1, \ldots, a_h \not\models l_i$, for all $i = m+1, \ldots, n$.

The reduct $\Pi^{(\sigma,S)}$ of Π relative to (σ, S) *is the union of all reducts $\Pi^{(\sigma,S)}_{a_1,\ldots,a_h}$ for all prefixes $a_1, \ldots, a_h$ of σ.*

Definition 3. *A temporal interpretation (σ, S) is an* answer set of Π *if (σ, S) is an answer set of the reduct $\Pi^{(\sigma,S)}$.*

Although the answer sets of a domain description Π are partial interpretations, in some cases, e.g., when the initial state is complete and all fluents are inertial, it is possible to guarantee that the temporal answer sets of Π are total.

In case the initial state is not complete,we consider all the possible ways to complete the initial state by introducing in Π, for each fluent name f, the rules: $f \leftarrow not\ \neg f$ and $\neg f \leftarrow not\ f$. The case of total temporal answer sets is of special interest as a total temporal answer set (σ, S) can be regarded as temporal model (σ, V), where, for each finite prefix $a_1 \ldots a_k$ of σ, $V(a_1, \ldots, a_k) = \{p : [a_1; \ldots; a_k]p \in S\}$. In the following, we restrict our consideration to domain descriptions Π, such that all the answer sets of Π are total.

The notion of *extension* of a domain description $D = (\Pi, \mathcal{C})$ over Σ is defined in two steps: first, the answer sets of Π are computed; second, all the answer sets which do not satisfy the temporal constraints in $\mathcal{C}$ are filtered out. For the second step, we need to define when a temporal formula α is satisfied in a total temporal interpretation S. Observe that a total answer set S over σ can be regarded as a linear temporal (DLTL) model [22]. Given a total answer set S over σ we define the corresponding temporal model as $M_S = (\sigma, V_S)$, where $p \in V_S(a_1, \ldots, a_h)$ if and only if $[a_1; \ldots; a_h]p \in S$, for all atomic propositions p. We say that a total answer set S over σ satisfies a DLTL formula α if $M_S, \varepsilon \models \alpha$.

Definition 4. *An* extension of a domain description $D = (\Pi, \mathcal{C})$ over Σ*, is any (total) answer set S of Π satisfying the constraints in $\mathcal{C}$.*

3 Specifying the business process and the norms

Let us consider a business process of an investment firm, where the firm offers financial instruments to an investor. The description of the business processes given in Figure 1 in the language YAWL (Yet AnotherWorkflow Language) [28].

Also, let us consider a regulation containing the following norms:

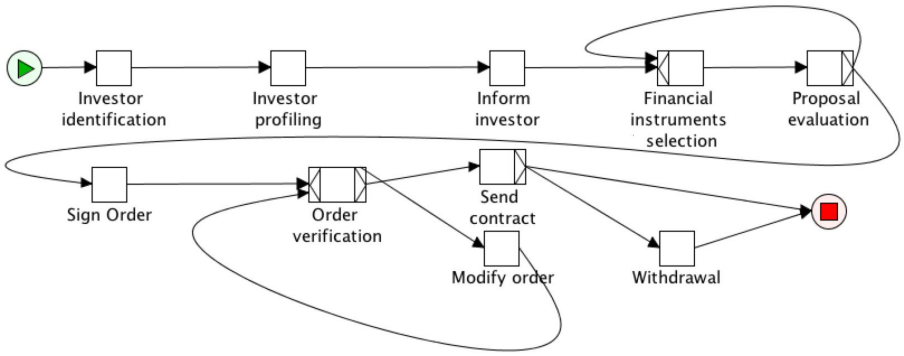

Fig. 1. Example business process in YAWL

(1) the firm shall provide to the investor adequate information on its services and policies before any contract is signed;
(2) if the investor signs an order, the firm is obliged to provide him a copy of the contract.

The execution of each task in the process has some preconditions and effects. Due to the presence of norms, the execution of a task in the process above should generate obligations to be fulfilled. For instance, according to the second norm, signing an order generates for the firm the obligation to provide a copy of the contract to the investor. Verifying the compliance of a business process to a regulation requires to check that, in all the executions of the business process, the obligations triggered by the norms are fulfilled.

In the following, we provide the specification of the business process and of the related norms in an action theory. The problem of verifying compliance of the business process to the norms is then defined as a reasoning problem in the action theory. Concerning the specification of a business process, we distinguish two parts in this specification: the specification of the workflow itself, for which we will exploit the capability of the action language to represent complex actions, and the specification of the atomic tasks occurring in it. The description of the atomic actions occurring in the business process provides the background knowledge, which is common both to the business process and to the norms. The effects and, possibly, the preconditions of the atomic tasks are defined by introducing propositions representing the properties of the world that are affected by the execution of the tasks and that are subject to the norms. They are the properties whose value is to be checked, for verifying the compliance of the process to the norms themselves. Such properties are sometimes used in the literature [14, 19, 30] as annotations that decorate the business process. Here, we exploit the action language to provide a more expressive formalism for formulating properties annotations. In the next subsections, we address the problems of specifying the atomic tasks, specifying the business process control and specifying the norms.

3.1 Semantic annotations: the specification of atomic tasks

The atomic tasks occurring in the business process are those represented by boxes in the YAWL specification. In the action theory, they are modeled as atomic actions with the same name as the atomic tasks.

To introduce the propositions needed to describe the effects and, possibly, the preconditions of atomic tasks, we introduce the following fluent names in $\mathcal{P}$:

$investor(C)$: C has been identified as a client (an investor);
$investor\_classified(C)$: the investor C has been classified;
$risk\_averse(C)$, $risk\_seeking(C)$ represent the profile of the client;
$informed(C)$: the client has been informed about the bank services and policies;
$selected(T, C)$: client C has selected financial instrument T;
$accepted(T, C)$: client C has accepted financial instrument T;
$order\_signed(T, C)$: client C has signed the order of financial instrument T;
$order\_confirmed(T, C)$: the order has been confirmed by the bank;
$sent\_contract(T, C)$: the contract of the order has been sent to the client;
$order\_deleted(T, C)$: the order has been canceled.

In the following, *profiling* stands for *investor profiling*, *inform* stands for *inform investor*, *fi_selection* stands for *financial instrument selection*, *p_eval* stands for *proposal evaluation* and, finally, *order_verif* stands for order verification.

The following action and causal laws describe the effect of the actions in the business process. Although the action language is propositional, in the following, we use variables in fluent names and in atomic action, so that each action/causal law stands for a finite number of ground action/causal laws:

$\Box([investor\_identification(C)]investor\_identified(C)) \leftarrow client(C))$
$\Box([profiling(C)]investor\_classified(C) \leftarrow client(C))$
$\Box([profiling(C)]risk\_averse(C)$ *or* $[profiling(C)]risk\_seeking(C)$
$\leftarrow client(C))$
$\Box([inform(C)]informed(C) \leftarrow client(C))$
$\Box([fi\_selection(C)]selected(t_1, C)$ *or* $\ldots$ *or* $[fi\_selection(C)]selected(t_n, C)$
$\leftarrow financial\_instr(t_1) \wedge \ldots \wedge financial\_instr(t_n) \wedge risk\_averse(C))$
$\Box([p\_eval(T, C)]accepted(T, C)$ *or* $[p\_eval(T, C)]\neg accepted(T, C))$
$\Box([sign\_order(T, C)]order\_signed(T, C) \leftarrow order(T), client(C))$
$\Box([order\_verif(T, C)]confirmed(T, C)$ *or* $[order\_verif(T, C)]\neg confirmed(T, C)$
$\leftarrow order(T), client(C))$
$\Box([send\_contract(T, C)]sent\_contract(T, C) \leftarrow order(T), client(C))$
$\Box([withdraw(T, C)]order\_deleted(T, C)) \leftarrow order(T), client(C))$
$\Box(\neg confirmed(T, C) \leftarrow order\_deleted(T, C))$
$\Box([end\_procedure]end)$

Laws defining executability conditions for atomic tasks can also be given. For instance,

$\Box([send\_contract(T, C)]\bot \leftarrow not\ confirmed(T, C))$

states that it is possible to send a contract to the investor only if the contract has been confirmed. Such temporal formulae can be seen as "good properties", that the modeler would like to verify on the business process and they can be verified to hold in all possible executions of the business process with the same technique that will be introduced for verifying norm compliance. Indeed, some of the norms will be formalized as precondition laws.

3.2 The specification of the business process workflow

When we are faced with the problem of specifying a business process, even in a given language as the one introduced in section 2.2, many options are available.

In [9], we have shown that the control flow of a business process can be modeled in a rigid way by means of a program expression π, i.e. by defining a complex action using composition operators like sequence, non deterministic choice and finite iteration, as well as test actions $p?$ which can be suitably introduced in the language. Then, a temporal constraint $\langle\pi\rangle\top$ is introduced in the set of constraints $\mathcal{C}$ to select those extensions of the domain description, corresponding to the possible executions of the program π.

Although this is a very simple solution, in the general case, the workflow of a business process may be non-structured or, even, we may want to provide a declarative specification of the business process, as done, for instance, in the declarative flow language ConDec [25]. It must be observed that the logical nature of our action language makes it well suited for a declarative specification. Indeed, the presence of general DLTL constraints in action domains allows for a simple way to constrain activities in a business process. As DLTL is an extension of LTL, it is possible to provide an encoding of ConDec constraints our action language.

Besides allowing for a declarative specification of business processes, the language introduced in Section 2 is well suited for encoding processes specified in conventional workflow languages, through the specification of action effects, preconditions and constraints. Consider, for instance, the atomic task *Investor profiling*. We can model the fact that this task can be executed only if the atomic task *Investor identification* has been executed, by introducing the precondition law: $\Box([profiling(C)]\bot \leftarrow not\ investor\_identified(C))$. Moreover, the fact that *Investor profiling* has to be executed after *Investor identification* is executed can be modeled by the temporal constraint:

$$\Box[investor\_identification(C)]\Diamond\langle profiling(C)\rangle\top$$

Consider, in addition, the atomic task *Order verification*. After its execution *Send contract* is to be executed if the order has been confirmed, otherwise, the task *Modify order* has to be executed. This can be modeled introducing the precondition laws

$$\Box([send\_contract(T,C)]\bot \leftarrow \neg confirmed(T,C))$$
$$\Box([modify\_order(T,C)]\bot \leftarrow confirmed(T,C))$$

and temporal constraints

$$\Box[order\_verif(C)](confirmed(T,C) \rightarrow \Diamond\langle send\_contract(T,C)\rangle\top)$$
$$\Box[order\_verif(C)](\neg confirmed(T,C) \rightarrow \Diamond\langle modify\_order(T,C)\rangle\top)$$

This approach can be generalized for translating YAWL processes into a domain description by action effects and preconditions as well as constraints. The translation, in general, would require to introduce new fluent names as, for instance, for each atomic task a, a fluent $executable(a)$, which is made true when the execution of task a is enabled. Some technicalities (that we do not address here) are needed to model AND/OR splits and AND/OR joins.

The approach we adopt in this paper for reasoning about actions is well suited for reasoning about systems with infinite computations (see [17]). To deal with finite computations we introduce a dummy action, which can be repeated infinitely many times after the termination of the process (thus giving rise to an infinite computation). In practice, however, as an optimization of the ASP translation, we can avoid looking for arbitrary models with loops during model checking and restrict to ad hoc computations corresponding to finite traces.

3.3 Normative specification

According to the normative specification, the execution of each task in the business process can trigger some normative positions (obligations, permissions, prohibitions). For instance, the *identification* task in the business process above, which introduces a new investor C, also generates the obligation to inform the investor. This obligation must be fulfilled during the course of execution of the business process, if the process is compliant with the norm stating that the firm has the obligation to inform customers.

In the following we make use of causal laws to represent norms in the action theory, and we introduce a notion of commitment to model obligations. The use of commitments has long been recognized as a "key notion" to allow coordination and communication in multi-agent systems [27, 20, 11]. A notion of commitment for reasoning about agent protocols in a temporal action logic has been adopted in [16]. Following [16], we introduce two kinds of commitments (which are regarded as special fluent propositions): *base-level commitments* having the form $\mathrm{C}(i,j,A)$ and meaning that agent i is committed to agent j to bring about A (where A is an arbitrary propositional formula not containing commitment fluents); *conditional commitments* having the form $\mathrm{CC}(i,j,B,A)$ and meaning that agent i is committed to agent j to bring about A, if condition B is brought about.
A base level commitment $\mathrm{C}(i,j,A)$ can be naturally regarded as an obligation (namely, OA, "A is obligatory"), in which the debtor and the creditor are made explicit. The two kinds of base-level and conditional commitments we use here are essentially those introduced in [31]. Our present choice is different from the one in [20], where agents are committed to execute an action rather than to achieve a condition.

The idea is that commitments (or obligations) are created as effects of the execution of some basic tasks in the business process and they are "discharged" when they have been fulfilled. A commitment $\mathrm{C}(i, j, A)$, created at a given state of a run of the process, is regarded to be *fulfilled* in the run if there is a later state of the run in which A holds. As soon as a commitment is fulfilled in a run, it is considered to be satisfied and no longer active: it can be discharged.

Given the notion of commitment introduced above, the norms which generate obligations to be fulfilled can be modeled as causal laws which trigger new commitments/obligations. Other norms which define preconditions on the executability of some actions or, in general, ordering constraints on the executions of atomic tasks can be encoded by general temporal formulas. For instance, we can encode the norms (1) and (2) above by the following precondition and causal laws:

$$\Box([sign\_order(T, C)]\bot \leftarrow not\ informed(C))$$
$$\Box(\mathrm{C}(firm, C, sent\_contract(T, C)) \leftarrow order\_signed(T, C))$$

The first one is a precondition for $sign\_order(T, C)$, stating that, if the client has not been informed, he cannot sign an order. The second one, a causal law, states that when an order is signed by C, the firm is committed to C to send her the information required.

Causal laws are needed for modeling the interplay of commitments and fluent changes. In particular, for each commitment $\mathrm{C}(i, j, \alpha)$, we introduce the following dynamic causal laws in the domain description:

(i) $\Box(\bigcirc\neg\mathrm{C}(i, j, \alpha) \leftarrow \mathrm{C}(i, j, \alpha), \bigcirc\alpha)$
(ii) $\Box(\bigcirc\mathrm{C}(i, j, \alpha) \leftarrow \mathrm{CC}(i, j, \beta, \alpha), \bigcirc\beta)$
(iii) $\Box(\bigcirc\neg\mathrm{CC}(i, j, \beta, \alpha) \leftarrow \mathrm{CC}(i, j, \beta, \alpha), \bigcirc\beta)$

A commitment to bring about α is considered fulfilled and is discharged as soon as α holds (i). A conditional commitment $\mathrm{CC}(i, j, \beta, \alpha)$ becomes a base-level commitment $\mathrm{C}(i, j, \alpha)$ when β has been brought about (ii) and, in that case, the conditional commitment is discharged (iii).

One of the central issues in the representation of norms comes from the defeasible nature of norms. Norms may have exceptions: recent norms may cancel older ones; more specific norms override more general norms and, in other cases, explicit priority information (not necessarily related to recency or specificity) is needed for eliminating conflicts. Consider the following example from [19]:

r_1: $\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), premium\_customer(M)$
r_2: $\neg\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), special\_order(S, M, O)$

Rule r_1 states that a seller has the obligation to apply a discount to premium customers. Rule r_2 states that customers are not entitled for a discount in case the order (O) is a special order. Observe that, if two rules are regarded as being strict, a state in which the fluents $premium\_customer(M)$, $special\_order(S, M, O)$ and $sells(S, M, O)$ hold results to be inconsistent.

To avoid conflicting situations as the one above, priorities among rules can be incorporated. Suppose the two rules above are regarded as defeasible and assume that rule r_2 has preference over rule r_1 (we write $r_2 > r_1$). The priority between the conflicting norms r_1 and r_2, with $r_2 > r_1$, can be modeled using default negation. For instance, we can transform the rules r_1 and r_2 as follows:

$\Box(\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), premium(M), not\ bl(r_1(S, M, O)))$
$\Box(\neg\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), special\_order(C), not\ bl(r_2(S, M, O)))$
$\Box(bl(r_1(S, M, O)) \leftarrow sells(S, M, O) \wedge special\_order(C), not\ bl(r_2(S, M, O)))$

where $bl(r_i(S, M, O))$ means that rule r_i is blocked. In this way, rule r_2, when applicable, blocks the application of r_1, but not vice-versa.

This treatment of priorities among conflicting rules, in essence, relies on the idea of using abnormality predicates for capturing exceptions. It is not intended to provide a general solution to the problem of modeling priorities among rules, as, in the general case, priorities may be also allowed between non conflicting rules. The problem of dealing with prioritized programs under the answer set semantics has been addressed, for instance, in [7] and in [10] in a more general setting. We believe that the approach proposed in [10] can be exploited in this setting to model defeasible norms as prioritized defeasible causal laws.

A further issue to be addressed when modeling norms is that of formalizing violations and reparation obligations, and we refer to [9] for a possible encoding of reparation chains in our language.

4 Compliance verification by model checking

In this section we provide a characterization of the problem of compliance, as a problem of reasoning about action in the action theory defined above. In Section 3.3, we have devised two different typologies of norms which we may want to verify compliance with: norms which can be encoded as a temporal formula (in the example, a precondition formula) and norms whose application generates obligations to be fulfilled, which can be modeled as causal laws generating commitments. Concerning the first kind of norms, the temporal formula encoding the norm has to be verified to be true in all the extensions of the domain description. Concerning the second kind of norms, verifying the compliance of the business process with such norms amounts to check that, in all the possible extensions of the domain description D, all the commitments generated will be eventually fulfilled, unless they have been cancelled: $\Box(\mathrm{C}(i, j, \alpha) \rightarrow \Diamond(\alpha \vee \neg\mathrm{C}(i, j, \alpha)))$. Action *withdraw*, for instance, might have the indirect effect of canceling the commitment to send the contract, if it has not yet been sent. Observe that canceling a commitment would not be possible if the commitment were encoded by the temporal formula $\Diamond\alpha$.

Let $D_B = (\Pi_B, \mathcal{C}_B)$ be a domain description defined as the specification of a given business process B, including the specification of the atomic tasks involved in the process (semantic annotations). Let N be a set of norms, which have been encoded by a set of causal laws Π_N and a set of temporal formulas P_N. The

domain description resulting from the encoding of the business process and the norms can then be defined as $D = (\Pi_B \cup \Pi_N, \mathcal{C}_B)$. We can define the problem of verifying the compliance of a business process to a set of norms as follows:

Definition 5. *The business process B is compliant with the set of norms $N = (\Pi_N, P_N)$ if, for each extension (σ, S) of the domain description $D = (\Pi_B \cup \Pi_N, \mathcal{C}_B)$, the following conditions hold:*

- *for each temporal formula α in P_N, (σ, S) satisfies α;*
- *for each commitment $\mathrm{C}(i, j, \alpha)$ occurring in Π_N, (σ, S) satisfies the formula $\Box(\mathrm{C}(i, j, \alpha) \rightarrow \Diamond(\alpha \vee \neg\mathrm{C}(i, j, \alpha)))$.*

Dually, the problem of identifying a *violation* to the norms can be regarded as a satisfiability problem: the problem of finding an execution of the business process which violates some of the norms, that is, the problem of finding an extension (σ, S) of D such that either (σ, S) contains unfulfilled commitments, i.e., it satisfies $\Diamond(\mathrm{C}(i, j, \alpha) \wedge \neg\Diamond(\alpha \vee \neg\mathrm{C}(i, j, \alpha)))$, or it falsifies a formula in Π_N.

Consider the domain description D, including the specification D_B of the business problem example and the causal law $\Box(\mathrm{C}(firm, C, sent\_contract(T, C)) \leftarrow order\_signed(T, C))$. Each extension S of the domain description satisfies the temporal formulas

$$\Box(\mathrm{C}(firm, C, sent\_contract(T, C)) \rightarrow \Diamond sent\_contract(T, C))$$
$$\Box([sign\_order(T, C)]\bot \leftarrow informed(C))$$

Hence, the business process is compliant with the norms. In fact, in all the execution of the business process, the commitment to send the contract is eventually fulfilled by the execution of the action *send_contract*, which has to be eventually executed in the business process; and, for the second formula, the execution of *sign_order* is always after $inform$ which makes the client informed.

In [17] we exploit bounded model checking (BMC) techniques [6] for computing the extensions of a temporal domain description and for verifying its temporal properties. More precisely, we describe a translation of a temporal domain description into standard ASP, so that the temporal answer sets of the domain description can then be computed as the standard answer sets of its translation. Extensions of the domain description satisfying the temporal constraints or given temporal properties are computed by bounded model checking, following the approach proposed in [17] for the verfication of DLTL formulas, which extends the one developed in [21] for bounded LTL model checking with Stable Models.

As an alternative to encoding the business process control flow in the logical formalisms (as done in section 3.2), a direct encoding of the workflow computations in the ASP program is also feasible, and makes the verification more efficient. In this case, the action language is used only for the specification of the semantic annotations and of the norms. Based on these ideas, we have used bounded model checking in ASP verify business process compliance. The implementation we have developed is based on the DLV system [23].

5 Conclusions and related work

The paper presents an approach to the verification of the compliance of business processes with norms. The approach is based on a temporal extension of ASP. Both the business process, their semantic annotation and the norms are encoded using temporal ASP rules as well as temporal constraints. In particular, defeasible causal laws are used for modeling norms and commitments are introduced for representing obligations. The verification of compliance can be performed by using BMC techniques. In particular, we exploit an approach developed in [17] for DLTL bounded model checking in ASP, which extends the approach for bounded LTL model checking with Stable Models in [21]. We are currently testing our implementation on several workflow examples to verify the scalability of the approach, and to compare with other approaches to compliance verification, including the traditional Petri net approach.

Several proposals in the literature introduce annotations on business processes for dealing with compliance verification [14, 19, 30]. In particular, [19] proposes a logical approach to the problem of business process compliance based on the idea of annotating the business process. Process annotations and normative specifications are provided in the same logical language, namely, the Formal Contract Language (FCL), which combines defeasible logic [3] and deontic logic of violations [18]. Compliance is verified by traversing the graph describing the process and identifying the effects of tasks and the obligations triggered by task execution. Ad hoc algorithms for propagating obligations through the process graph are defined.

In [30] a formal execution semantics for annotated business processes is introduced. The proposed semantics combines a Petri-net like (token passing) semantics for BPMN process execution, coming from the workflow community, with a declarative specification of actions preconditions and effects in clausal form, coming from the AI literature of actions and state changes. Several verification tasks are defined to check whether the business process control flow interacts correctly with the behaviour of the individual activities. However, [30] does not address the problem of verifying compliance of the business process with norms.

An approach to compliance based on a commitment semantics in the context of multi-agent systems is proposed in [8]. The authors formalize notions of conformance, coverage, and interoperability, proving that they are orthogonal to each other. Another approach to the verification of agents compliance with protocols, based on a temporal action theory, has been proposed in [16]. These papers do not address the problem of compliance with norms.

[4] presents an approach to compliance checking for BPMN process models using BPMN-Q, a visual language based on BPMN. Compliance rules are given a declarative representation as BPMN-Q queries. Then, BPMN-Q queries are translated into temporal formulas for verification.

In [24] the Abductive Logic Programming framework SHIFF [2] is exploited in the declarative specification of business processes as well as in the (static and runtime) verification of their properties. In particular, in [1] expectations are

used for modelling obligations and prohibitions and norms are formalized by abductive integrity constraints.

In [26] Concurrent Transaction Logic (CTR) is used to model and reason about general service choreographies. Service choreographies and contract requirements are represented in CTR. The paper addresses the problem of deciding if there is an execution of the service choreography that complies both with the service policies and the client contract requirements.

Temporal rule patterns for regulatory policies are introduced in [15], where regulatory requirements are formalized as sets of compliance rules in a real-time temporal object logic. The approach is used essentially for event monitoring.

Acknowledgments

We want to thank the anonymous referees for their helpful comments. This work has been partially supported by Regione Piemonte, Project "ICT4LAW: *ICT Converging on Law: Next Generation Services for Citizens, Enterprises, Public Administration and Policymakers*"

References

1. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, and G. Sartor. Mapping of Deontic Operators to Abductive Expectations. *NORMAS*, pages 126–136, 2005.
2. Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: The sciff framework. *ACM Trans. Comput. Log.*, 9(4), 2008.
3. G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Representation results for defeasible logic. *ACM Trans. on Computational Logic*, 2:255–287, 2001.
4. Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using bpmn-q and temporal logic, lncs 5240. In *BPM*, pages 326–341. Springer, 2008.
5. Matteo Baldoni, Cristina Baroglio, and Elisa Marengo. Behavior-oriented commitment-based protocols. In *Proceedings ECAI 2010*, pages 137–142, 2010.
6. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
7. Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.
8. A.K. Chopra and M.P. Sing. Producing compliant interactions: Conformance, coverage and interoperability. *DALT IV, LNCS(LNAI) 4327*, pages 1–15, 2006.
9. D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. L. Pozzato, and D. Theseider Dupré. Verifying business process compliance by reasoning about actions. In *CLIMA XI*, pages 99–116, 2010.
10. James P. Delgrande, Torsten Schaub, and Hans Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, 2003.
11. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *AAMAS03*, pages 520–527.
12. M. Gelfond. Answer Sets. *Handbook of Knowledge Representation, chapter 7, Elsevier*, 2007.

13. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proc. of the 5th Int. Conf. and Symposium*, pages 1070–1080, 1988.
14. A. Ghose and G. Koliadis. Auditing business process compliance. *ICSOC, LNCS 4749*, pages 169–180, 2007.
15. C. Giblin, S. Müller, and B. Pfitzmann. From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. *IBM Reasearch Report*, 2007.
16. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic (Special issue on Logic Based Agent Verification)*, 5:214–234, 2007.
17. L. Giordano, A. Martelli, and D. Theseider Dupré. Reasoning about Actions with Temporal Answer Sets. *Proc. CILC 2010, 25th Italian Conference on Computational Logic*, 2010.
18. G. Governatori and A. Rotolo. Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. *Australasian Journal of Logic*, 4:193–215, 2006.
19. G. Governatori and S. Sadiq. The journey to business process compliance. *Handbook of Research on BPM, IGI Global*, pages 426–454, 2009.
20. F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems, Springer LNAI 2650*, 2003.
21. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.
22. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.
23. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.
24. Marco Montali, Paolo Torroni, Federico Chesani, Paola Mello, Marco Alberti, and Evelina Lamma. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundam. Inform.*, 102(3-4):325–361, 2010.
25. Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops, LNCS 4103*, pages 169–180. Springer, 2006.
26. Dumitru Roman and Michael Kifer. Semantic web service choreography: Contracting and enactment. In *International Semantic Web Conference, LNCS 5318*, pages 550–566, 2008.
27. M. P. Singh. A social semantics for Agent Communication Languages. *Issues in Agent Communication, LNCS(LNAI) 1916*, pages 31–45, 2000.
28. W. van der Aalst and A. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
29. Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*, 2006.
30. Ingo Weber, Jörg Hoffmann, and Jan Mendling. Beyond soundness: On the verification of semantic business process models. *Distributed and Parallel Databases (DAPD)*, 2010.
31. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. *AAMAS'02*, pages 527–534, 2002.