Temporal Deontic Action Logic for the Verification of Compliance to Norms in ASP *

Laura Giordano
DISIT
Università del
Piemonte Orientale
Italy
laura.giordano@mfn.unipmn.it

Alberto Martelli
Dipartimento
di Informatica
Università di Torino
Italy
mrt@di.unito.it

Daniele Theseider Dupré
DISIT
Università del
Piemonte Orientale
Italy
dtd@di.unipmn.it

ABSTRACT

The verification of compliance of business processes to norms requires the representation of different kinds of obligations, including achievement obligations, maintenance obligations, obligations with deadlines and contrary to duty obligations. In this paper we develop a deontic temporal extension of Answer Set Programming (ASP) suitable for verifying compliance of a business process to norms involving such different types of obligations. To this end, we extend Dynamic Linear Time Temporal Logic (DLTL) with deontic modalities to define a Deontic DLTL. We then combine it with ASP to define a deontic action language in which until formulas and next formulas are allowed to occur within deontic modalities. We show that in the language we can model the different kinds of obligations which are useful in the verification of compliance to normative requirements. The verification can be performed by bounded model checking techniques.

1. INTRODUCTION

The verification of compliance of business processes to norms requires the representation of different kinds of obligations, including achievement obligations, maintenance obligations, obligations with deadlines and contrary-to-duty obligations. In [22] a classification of obligations is proposed through a conceptual analysis of several kinds of deadlines. These types of obligations are recognized in [21] as the relevant onesdeontic for business process compliance.

In [9] an approach based on reasoning about actions was proposed for the verification of compliance of business processes with norms. The approach is based on the temporal extension of ASP in [20], combining ASP with the dynamic linear time temporal logic DLTL [27], an extension of LTL in which the temporal operators are enriched with regular program expressions. The business process, its 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 [37] are used for representing achievement obligations.

In this paper we aim at generalizing the approach in [9] to deal with the different kinds of obligations mentioned above, including maintenance and punctual obligations. Our proposal is based on the idea of extending Dynamic Linear Time Temporal Logic (DLTL) [27] with Standard Deontic Logic (SDL) [40] to define a Deontic DLTL (DDLTL) in which temporal formulas are allowed to occur within the deontic operator. The proposed logic is similar in spirit to the Dynamic Deontic and Temporal Logic proposed by Dignum and Kuiper [13] to reason about obligations and deadlines. It is also related with the logic combining temporal and deontic logics in [6], where the product of LTL and SDL is the starting point to study propagation properties of obligations. In [6] Broersen and Brunel show that the genuine product of LTL and SDL is not compatible with the propagation properties, and define an alternative notion of temporal deontic frames based on levels of deontic ideality. They observe that propagation properties of obligations "are only valid if we assume that the 'deontic realm' is not changed by an explicit update of the norms". In this paper, we do not make this assumption and we consider situations in which actions may generate or cancel an obligation. For instance, the obligation to pay for goods can be cancelled by the action of withdrawing from the contract. For this reason we do not include the propagation properties in the definition of the temporal deontic logic. The dynamic of propagation and cancellation of obligations, in our approach, is addressed in the action theory, where obligations can be defined to be persistent or not (according to the kind of obligation) and an action which cancels an obligation blocks its (default) persistence.

In the paper we combine DDLTL with Answer Set Programming [15] to define a Temporal Deontic ASP, which is an extension of Temporal ASP introduced in [20]. In Temporal Deontic ASP, deontic fluents $\mathbf{O}(\alpha)$ are allowed, where α is restricted to until formulas or next formulas. Also, in a formula $\mathbf{O}(\alpha \mathcal{U} \beta)$ and in $\mathbf{O}(X\alpha)$, α and β must be non-temporal literals or "simple" temporal literals. This restriction is introduced in order to allow for a simple treatment of deontic literals exploiting bounded model checking techniques in the verification of temporal properties of action domains, where the consistency of a set of deontic literals is guaranteed through the verification of a set of constraints. We require the deontic operator to be serial, as usual in SDL [40]. We show that in Temporal Deontic ASP we can model the different kinds of obligations mentioned above, including those involving deadlines and violations, by dealing with the dynamics of obligations and, specifically, with propagation properties of obligations expressed in the action theory as non-monotonic causal laws.

*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*".

Given a specification of a business process and of a set of business rules as an action domain in Temporal Deontic ASP, a temporal answer set of the action domain corresponds to a run of the business process which includes all the obligations triggered during the execution. Furthermore, it corresponds to a path in a temporal deontic model of the logic DDLTL. Compliance verification can be performed using Bounded Model Checking techniques [4], exploiting the approach developed in [20] for the verification of temporal properties of action theories by bounded model checking in ASP, which extends the approach for bounded LTL model checking with Stable Models in [26]. In fact, compliance requirements related to obligations can be expressed as temporal properties.

Therefore, ASP provides a framework suitable for representing the dynamics of obligations (and other fluents) in a process, as well as for verifying compliance of the process to the obligations that may be triggered in its execution.

2. DEONTIC DYNAMIC LINEAR TIME TEMPORAL LOGIC

In this section we combine Dynamic Linear Time Temporal Logic (DLTL) [27], an extension of LTL in which temporal operators are enriched with program expressions as in dynamic logic, with a deontic logic to define a Deontic Dynamic Linear Time Temporal Logic (DDLTL). The idea of combining dynamic logic, deontic logic and temporal logic has been proposed in [13] for the specification of deadlines. [13] first defines a dynamic deontic logic and, then, combines it with a temporal logic. Here we start from DLTL [27], which already combines temporal and dynamic logics, and we add a deontic operator **O**, whose semantics is that of Standard Deontic Logic [40].

The Dynamic Linear Time Temporal Logic DLTL [27] is an extension of LTL in which temporal operators are enriched with program expressions. In particular, in DLTL the next state modality can be indexed by actions, and the until operator $\mathcal{U}^\pi$ can be indexed by a program π which, as in PDL, can be any regular expression built from atomic actions using sequence (;), nondeterministic choice ($+$) and finite iteration ($*$).

Let Σ be a finite non-empty alphabet representing actions. DLTL allows until formulas of the form $\alpha \mathcal{U}^\pi \beta$, where the program $\pi \in Prg(\Sigma)$ is a regular expression built from a set Σ of atomic actions. More precisely,

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$$

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions containing $\top$ and $\bot$ (standing for *true* and *false*). We extend the language of the logic of DLTL over σ by including in the language the deontic operator **O** as follows:

$$\text{DDLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta \mid \mathbf{O}(\alpha)$$

where $p \in \mathcal{P}$ and α, β range over DDLTL(Σ).

As for LTL, DLTL models are infinite linear sequences of worlds (propositional interpretations), each one reachable from the initial world by a finite sequence τ of actions in the alphabet Σ. Here, to define the Kripke semantics of DDLTL, we generalize the semantics of DLTL as given in [27].

A state of a model is a pair (n, w), where n represents a time point and w a world. Informally, a model consists in a set of linear sequences, $(i, w), (i+1, w), \ldots$, representing the evolution of a world in time. At time n, from a state (n, w), other states (n, w') (with the same time point n) can be reached through the deontic accessibility relation R_d. Each accessible state has a linear sequence of worlds departing from it.

DEFINITION 1. *Let W be a non-empty set of worlds. A* temporal deontic model *over Σ is a tuple $M = (\mathcal{S}, R_t, R_d, V)$, where*

- *$\mathcal{S} \subseteq \mathbb{N} \times \mathbb{W}$ is a set of states, such that $W_i = \{w : (i, w) \in S\}$ is the set of worlds at time point i and $W_0 \subseteq W_1 \subseteq \ldots$;*
- *$R_t : \mathcal{S} \to \Sigma \times \mathcal{S}$ is a total* temporal accessibility function, *such that $R_t(n, w) = (a, (n+1, w))$ for some $a \in \Sigma$;*
- *$R_d \subseteq \mathcal{S} \times \mathcal{S}$ is a* deontic accessibility relation *among states, such that if $((n, w), (n', w')) \in R_d$, then $n = n'$; moreover, R_d is serial, i.e., for all $s \in \mathcal{S}$ there is $s' \in \mathcal{S}$ such that $(s, s') \in R_d$;*
- *$V : \mathcal{S} \to 2^{\mathcal{P}}$ is a valuation function.*

For each $n \in \mathbb{N}$, W_n is the set of of worlds accessible at time point n. The fact that the sets W_n are increasing, means that at time n there may be a world which was not present at time $n-1$. Time is linear and, when applied to a state $s \in \mathcal{S}$, the function R_t returns a pair (a, s') where a is the action executed in s, and s' is the state obtained by executing a in s. As in standard deontic logic, we have assumed the accessibility relation R_d to be serial.

In the following, we denote by $\tau, \tau', \ldots$ the finite action sequences in Σ^* (including the empty one). For any program expression π, we let $[[\pi]]$ to be the set of finite sequences associated with π. For all finite action sequences τ, we write $s \Rightarrow^\tau s'$ to mean that s' is reachable from s through the action sequence τ. We define $s \Rightarrow^\tau s'$ by induction on the length of τ as follows: (i) $s \Rightarrow^\varepsilon s$ and (ii) $s \Rightarrow^{\tau a} s'$ iff $s \Rightarrow^\tau s''$ and $R_t(s'') = (a, s')$, for some s''. In the following, we denote by $\leq$ the usual prefix ordering over Σ^* namely, $\tau \leq \tau'$ (τ is a prefix of τ') iff $\exists \tau''$ such that $\tau\tau'' = \tau'$. Moreover, we let $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

Given a model $M = (\mathcal{S}, R_t, R_d, V)$ over Σ, a state $s \in \mathcal{S}$ and a formula α, the *satisfiability of a formula α in s*, written $M, s \models \alpha$, is defined as follows:

- $M, s \models p$ iff $p \in V(s)$;
- $M, s \not\models \bot$;
- $M, s \models \neg\alpha$ iff $M, s \not\models \alpha$;
- $M, s \models \alpha \vee \beta$ iff $M, s \models \alpha$ or $M, s \models \beta$;
- $M, s \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau \in [[\pi]]$ such that $s \Rightarrow^\tau s'$ and $M, s' \models \beta$. Moreover, for every τ' such that $\varepsilon \leq \tau' < \tau$ and and for every $s'' \in \mathcal{S}$ such that $s \Rightarrow^{\tau'} s''$, $M, s'' \models \alpha$;

- $M, s \models \mathbf{O}(\alpha)$ iff for all $s' \in W$ such that $(s, s') \in R_d$, $M, s' \models \alpha$

A formula $\alpha \mathcal{U}^{\pi} \beta$ is true in a state s if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π, starting from s. The operator $\mathbf{O}$ is read "it is obligatory that" and $\mathbf{O}(\alpha)$ is true in a state s if α holds in all the states s' that are ideal with respect to s. The possibility operator $\mathbf{P}$, which is read "it is permitted that", can be defined as the dual of $\mathbf{O}$, i.e., $\mathbf{P}(\alpha) \equiv \neg\mathbf{O}\neg(\alpha)$.

As in DLTL, from the until operator, the derived modalities $\langle\pi\rangle$, $[\pi]$, X (next), $\mathcal{U}$, $\Diamond$ and $\Box$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^{\pi}\alpha$, $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$, $X\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Informally, a formula $[\pi]\alpha$ is true in a state s of a linear temporal model if α holds in all the states of the model which are reachable from s through any execution of the program π. A formula $\langle\pi\rangle\alpha$ is true in a state s of a linear temporal model if there exists a state of the model reachable from s through an execution of the program π, in which α holds.

Observe that for obligations of the form $\mathbf{O}(X\phi)$ the semantics in [6] satisfies the "perfect recall property" $\mathbf{O}(X\phi) \rightarrow X\mathbf{O}(\phi)$, stating that no obligation is 'forgotten' over time. This axiom, however, is not a valid formula in DDLTL. We assume that each obligation can be cancelled by an explicit cancellation action. Hence, the semantics above neither satisfies the perfect recall property nor the propagation property in [6]. However, in the following we will introduce weaker (non-monotonic) propagation properties for obligations in the temporal deontic action language.

Observe also that DDLTL is a conservative extension of DLTL. It can be shown that DDLTL is decidable.

PROPOSITION 1. *Satisfiability of a formula in DDLTL can be solved in* EXPTIME.

PROOF. Satisfiability in DDLTL can be polinomially reduced to concept satisfiability in the description logic $\mathcal{ALCF}_{reg}$, which extends $\mathcal{ALC}$ with functional roles and role operators from Propositional Dynamic Logic [25], namely, composition, union, reflexive-transitive closure and test. Satisfiability in $\mathcal{ALCF}_{reg}$ is known to be in EXPTIME [18]. The reduction proof is similar to the one in [36] and, more specifically, to the proof of Lemma 1 in [31] for $\text{LTL}_{\mathcal{ALC}}$ with expanding domains. □

The reduction of satisfiability in DDLTL to satisfiability in the description logic $\mathcal{ALCF}_{reg}$ makes it possible to exploit $\mathcal{ALCF}_{reg}$ inference procedures for the verification of the satisfiability and validity of formulas in DDLTL. However, this would require a monotonic solution to model persistence of the effects of actions, including obligations. In the following we develop a clausal non monotonic fragment of DDLTL, adopting the ASP paradigm and extending both ASP rules and the notion of answer set to deontic temporal formulas in the line of the temporal answer set programming language in [20]. In this approach we adopt a nonmonotonic solution for the frame problem where default negation in ASP is exploited to deal with persistency of fluents and with the propagation properties of obligations in a flexible way, as well as to capture the defeasible nature of norms. To reason about properties of action theories, we exploit bounded model checking techniques in ASP following [26, 20].

3. A TEMPORAL DEONTIC ACTION LANGUAGE

In this section, we introduce a temporal deontic extension of Answer Set Programming (ASP) which combines ASP with DDLTL, as a language for defining temporal deontic action theories. Action theories will consist of two components: a set of deontic temporal rules (namely ASP rules which may contain deontic and temporal literals), and a set of DDLTL formulas constraining the domain description. While ASP rules are used in the definition of action effects and preconditions and, in particular, in the specification of business rules, the DDLTL component can be used to define temporal constraints on the action domain as well as to encode temporal/deontic properties to be checked.

Both in ASP rules and in the set of constraints, we restrict the language of DDLTL by limiting the nesting of temporal and deontic operator to allow only a restricted choice of temporal formulas to occur within the deontic operators. As we will see, with this restriction, the verification techniques based on LTL Bounded Model Checking [4] and their ASP encoding [26] can be adapted to deal with temporal deontic formulas without the need to reason on temporal deontic Kripke structures. In this respect, our approach is related with the proposals in reasoning about actions and planning, which deal with epistemic knowledge by explicitly introducing knowledge literals within the states of the action domain [11, 3, 33]. However, here we allow deontic literals (including some restricted kinds of temporal formulas) to occur in the states of the action domain.

Let $\mathcal{L}$ be a first order language which includes a finite number of constants and variables, but no function symbol. Let $\mathcal{P}$ be the set of predicate symbols, Var the set of variables and C the set of constant symbols. We call *fluents* the atomic literals of the form $p(t_1, \ldots, t_n)$, where, for each i, $t_i \in Var \cup C$.

A *simple fluent literal* l is an atomic literal $p(t_1, \ldots, t_n)$ or its negation $\neg p(t_1, \ldots, t_n)$. We denote by Lit_S the set of all simple fluent literals, and we assume that the fluent $\bot$, representing the inconsistency, and the fluent $\top$ (true) are included in Lit_S.

A *deontic fluent literal* has the form $\mathbf{O}(\alpha)$ or $\neg\mathbf{O}(\alpha)$, where α is a restricted temporal formula defined as follows:

$$\alpha ::= l \mid Xl \mid l_1 \mathcal{U} l_2 \mid l_1 \mathcal{U} \langle a \rangle \top$$

where $l, l_1, l_2 \in Lit_S$, $l_2 \neq \bot$ and $a \in \Sigma$. The meaning of the formula $l_1 \mathcal{U} \langle a \rangle \top$ is that l_1 holds until eventually action a is executed. As we will see we allow this kind of temporal formulas in deontic literals to model the obligation to execute an action a within a certain deadline. We denote by Lit_D the set of all deontic fluent literals.

A *temporal fluent literal* has the form $[a]l$, $\langle a\rangle l$ or Xl, where $l \in Lit_S \cup Lit_D$ and a is an action name (an atomic proposition, possibly containing variables).

Given a (simple, deontic or temporal) fluent literal l, $not\ l$ represents the default negation of l. A (simple, deontic or temporal)

fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*.

Observe that fluent literals do not contain nested deontic operators and they contain very limited nesting of temporal operators.

The *laws* of a domain description are formulated as rules of a temporally extended logic programming language having the form

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \quad (1)$$

where the l_i's are simple, deontic or temporal fluent literals, with $l_0 \neq \langle a \rangle l$. As usual in ASP, rules with variables are a shorthand for the set of their ground instances; and we let Σ be the set of ground instances of atomic actions in the domain description.

A *state*, informally, is a set of ground fluent simple and deontic literals.

The execution of an action in a state may change the values of fluents through its direct and indirect effects. In particular, an action can generate or cancel obligations.

We assume that a law as (1) can be applied in all states while, when prefixed with **Init**, it only applies to the initial state.

A *domain description* D is a pair $(\Pi, \mathcal{C})$, where Π is a set of laws describing the effects and executability preconditions of actions (as described below), and $\mathcal{C}$ is a set of *temporal deontic constraints*, namely DDLTL formulas, built on the set of ground fluents, in which deontic formulas are restricted to deontic fluent literals.

Π is a set of laws such as action laws, causal laws, precondition laws, persistency laws, initial state laws, that are normally used in action theories, and can all be defined as instances of (1).

Action laws describe the effects of atomic actions. The meaning of an action law

$$[a]l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n,$$

(where $l_0 \in Lit_S \cup L_D$ and $l_1, \ldots, l_n$ are either simple or deontic fluent literals or temporal fluent literals of the form $[a]l$) is that executing action a in a state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold makes the effect l_0 to hold in the state obtained after the action execution. For instance, in

$$[accept\_price]\mathbf{O}(\top\ \mathcal{U}\ \langle pay \rangle \top)$$
$$[cancel\_payment]\neg\mathbf{O}(\top\ \mathcal{U}\ \langle pay \rangle \top)$$

the action $accept\_price$ creates an obligation to pay, while the action $cancel\_payment$ cancels that obligation[1]. As discussed in [5], obligations should come with a deadline; we refer to sections 6 and 7 for our treatment of deadlines.

Precondition laws have the form:

$$[a]\bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$$

(where $l_1, \ldots, l_n$ are either simple or deontic fluent literals) meaning that a cannot be executed (it has an inconsistent effect) in case $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold. For instance

$$[send\_contract]\ \bot \leftarrow \neg confirmed$$

states that a contract cannot be sent if it has not been confirmed.

Causal laws define causal dependencies among propositions, which are used to derive indirect effect of actions, called *ramifications* in the literature of reasoning about actions and change, where it is well known that causal dependencies among propositions are not suitably represented by material implication in classical logic. *Static causal laws* have the form:

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$$

where the l_i's are simple or deontic fluent literals. Their meaning is: if $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold in a state, then l_0 is caused to hold in that state. An example is

$$\neg order\_confirmed \leftarrow order\_deleted$$

in a business process where a customer can delete her order after confirmation by the provider.

Dynamic causal laws have the form:

$$Xl_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n$$

where l_0 is a simple or deontic fluent literal and the t_i's are either simple or deontic fluent literals, or temporal fluent literals of the form Xl_i (meaning that the fluent literal l_i holds in the next state). Their meaning is: if $t_1, \ldots, t_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold, then l_0 is caused to hold in the next state. For instance, dynamic causal laws can be used to define persistency:

$$XF \leftarrow F, not\ X\neg F$$

(where F is a fluent) while the rule:

$$X\neg\mathbf{O}(\top\ \mathcal{U}\ \langle A \rangle \top) \leftarrow \mathbf{O}(\top\ \mathcal{U}\ \langle A \rangle \top), \langle A \rangle \top$$

cancels the obligation to execute action A after the execution of A (see Section 5).

The language also includes constraints of the form

$$\bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$$

where the l_i's are simple, deontic or temporal fluent literals.

Although in the language we have only introduced the deontic modality $\mathbf{O}$, the other deontic modalities, such as $\mathbf{P}$ and $\mathbf{F}$, can be defined from $\mathbf{O}$ using static causal laws.

[1] In a richer modeling, obligations could have as additional parameters, as in case of *commitments* [37], an agent as a *debtor* and another as a *creditor*.

4. TEMPORAL DEONTIC ANSWER SETS

In [20], the semantics of a domain description is defined by extending the notion of *answer set* [15] to define *temporal answer sets*, which capture the linear structure of temporal models. In this section we adapt the notion of *temporal answer sets* to temporal deontic domain descriptions. The main concern we need to address is that of consistency of deontic formulas in each state of a temporal deontic answer set. In fact, by the seriality requirement on the deontic accessibility relation R_d in Kripke frames, it cannot be the case that an unsatisfiable set of deontic fluents may belong to a state.

In the following we shortly recall the definition of temporal answer set from [20] and we modify it to deal with consistency of deontic modalities. To this purpose, we let Π be the ground instantiation of the domain description, and Σ the set of all the ground instances of the action names in Π.

For conciseness, we call "simple (deontic, temporal) literals" the "simple (deontic,temporal) fluent literals". To define the notion of extension, we need to introduce additional rules of the form: $[a_1; \ldots; a_h](t_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n)$ that will be used to define the reduct of a program, where the t_i's are simple, deontic or temporal literals. The modality $[a_1; \ldots; a_h]$ in front of the rule says that the rule applies in the state obtained after the execution of the sequence of actions $a_1, \ldots, a_h$. Conveniently, also the notion of temporal literal used so far needs to be extended to include literals of the form $[a_1; \ldots; a_h]l$, meaning that the (simple or deontic) literal l holds after after the execution of the sequence of actions $a_1, \ldots, a_h$.

Generalizing the definition of temporal answer sets in [20], we define a temporal deontic answer set as a temporal deontic interpretation (σ, S), where $\sigma \in \Sigma^\omega$ is a sequence of actions and S is a consistent set of ground literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ and l is a ground simple or deontic fluent literal. Each prefix $a_1 \ldots a_k$ of σ corresponds to a state of the interpretation, the state obtained by the execution of the actions $a_1 \ldots a_k$, namely $w^{(\sigma,S)}_{a_1 \ldots a_k} = \{l : [a_1; \ldots; a_k]l \in S\}$.

A temporal interpretation (σ, S), is *propositionally 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 simple or deontic fluent literal l, and it is not the case that $[a_1; \ldots; a_k]\bot \in S$. 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 simple or deontic fluent atom p.

The notion of satisfiability of a rule in a temporal deontic interpretation (σ, S), as well as the notion of *reduct* $\Pi^{(\sigma,S)}$ *of (a domain description)* Π *relative to* (σ, S) can be defined as natural extensions of Gelfond and Lifschitz' ones [15]. With these notions, a *temporal deontic answer set* of Π is defined as a consistent temporal deontic interpretation (σ, S) such that S is minimal (in the sense of set inclusion) among the S' such that (σ, S') is a partial interpretation satisfying the rules in the reduct $\Pi^{(\sigma,S)}$.

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

$(\sigma, S), a_1 \ldots a_k \models \top$
$(\sigma, S), a_1 \ldots a_k \not\models \bot$
$(\sigma, S), a_1 \ldots a_k \models l$ *iff* $[a_1; \ldots; a_k]l \in S$, for l simple literal
$(\sigma, S), a_1 \ldots a_k \models [a]l$ *iff* $[a_1; \ldots; a_k; a]l \in S$, or $a_1 \ldots a_k a$ is not a prefix of σ
$(\sigma, S), a_1 \ldots a_k \models \langle a \rangle l$ *iff* $[a_1; \ldots; a_k; a]l \in S$, and $a_1 \ldots a_k a$ is a prefix of σ
$(\sigma, S), a_1 \ldots a_k \models Xl$ *iff* $[a_1; \ldots; a_k; b]l \in S$, and $a_1 \ldots a_k b$ is a prefix of σ

The satisfiability of rule bodies in a deontic temporal interpretation is defined as usual. A rule $H \leftarrow Body$ is satisfied in a deontic temporal interpretation (σ, S) if, for all action sequences $a_1 \ldots a_k$ (including the empty action sequence ε), $(\sigma, S), a_1 \ldots a_k \models Body$ implies $(\sigma, S), a_1 \ldots a_k \models H$. A rule **Init** $H \leftarrow Body$ is satisfied in a deontic temporal interpretation (σ, S) if, $(\sigma, S), \varepsilon \models Body$ implies $(\sigma, S), \varepsilon \models H$, where ε is the empty action sequence. A rule $[a_1; \ldots; a_h](H \leftarrow Body)$ is satisfied in a deontic temporal interpretation (σ, S) if, $(\sigma, S), a_1 \ldots a_k \models Body$ implies $(\sigma, S), a_1 \ldots a_k \models H$.

Let Π be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$. The following definitions generalize the corresponding definitions in [20].

DEFINITION 2. *A deontic temporal interpretation* (σ, S) *is a* deontic 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 deontic temporal interpretation satisfying the rules in* Π.

To define deontic temporal answer sets of a program Π containing negation, given a deontic 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 [16] to compute a different reduct of Π for each prefix $a_1, \ldots, a_h$ of σ.

DEFINITION 3. *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 4. *A deontic temporal interpretation* (σ, S) *is a* deontic temporal answer set of Π *if* (σ, S) *is a deontic temporal answer set of the reduct* $\Pi^{(\sigma,S)}$.

The notion of propositional consistency introduced above is too weak to guarantee that the set of obligations occurring in the states of a deontic temporal answer set are consistent with each other, according to the notion of consistency in Section 2. We say that a temporal interpretation (σ, S), is *deontically consistent* iff for each prefix $a_1 \ldots a_k$, $w^{(\sigma,S)}_{a_1 \ldots a_k}$ is satisfiable in the logic DDLTL. Domain descriptions Π such that all the answer sets of Π are total and deontically consistent are of special interest as their answer sets (σ, S) corresponds to a path in a DDLTL model.

PROPOSITION 2. *Given a deontically consistent temporal interpretation (σ, S) with $\sigma = a_1 a_2 a_3 \ldots$, there is a DDLTL model M, with valuation function V, such that:*

- *M contains the path $w_0 \Rightarrow^{a_1} w_1 \Rightarrow^{a_2} w_2 \Rightarrow^{a_3} \ldots$;*
- *$V(w_k) = \{p \ : \ p$ simple literal and $[a_1, \ldots, a_k]p \in S\}$*
- *$M, w_k \models \mathbf{O}(\alpha)$, for each $\mathbf{O}(\alpha)$ such that $[a_1, \ldots, a_k]\mathbf{O}(\alpha) \in S$.*

Note that the deontic consistency of the answer set is essential to guarantee the existence of the model M.

Given the restriction on the deontic literals admitted in the action language, we can introduce a set of conditions on temporal deontic answer sets which guarantee its (deontic) consistency.

PROPOSITION 3. *A temporal deontic interpretation (σ, S) is deontically consistent iff it is propositionally consistent and, for each prefix $a_1 \ldots a_k$ of σ, the state $w^{(\sigma,S)}_{a_1 \ldots a_k}$ does not contain any of the following sets of deontic literals, for all simple fluents p, simple literals l_1, l_2 and distinct action names a_1, a_2, a_3, a_4:*

- $\{\mathbf{O}(p), \mathbf{O}(\neg p)\}$
- $\{\mathbf{O}(Xp), \mathbf{O}(X\neg p)\}$
- $\{\mathbf{O}(l_1 \mathcal{U} l_2), \mathbf{O}(\neg l_2), \mathbf{O}(\neg l_1)\}$
- $\{\mathbf{O}(l_1 \mathcal{U} l_2), \mathbf{O}(\neg l_2), \mathbf{O}(X\neg l_2), \mathbf{O}(X\neg l_1)\}$
- $\{\mathbf{O}(l_1 \mathcal{U} l_2), \mathbf{O}(\neg l_2), \mathbf{O}(\neg l_1 \mathcal{U} l_2)\}$
- $\{\mathbf{O}(l_1 \mathcal{U} l_2), \mathbf{O}(\neg l_2), \mathbf{O}(\neg l_1 \mathcal{U} \langle a_1 \rangle \top), \mathbf{O}(\neg l_1 \mathcal{U} \langle a_2 \rangle \top)\}$
- $\{\mathbf{O}(l_1 \mathcal{U} \langle a_1 \rangle \top), \mathbf{O}(l_1 \mathcal{U} \langle a_2 \rangle \top), \mathbf{O}(\neg l_1)\}$
- $\{\mathbf{O}(l_1 \mathcal{U} \langle a_1 \rangle \top), \mathbf{O}(l_1 \mathcal{U} \langle a_2 \rangle \top), \mathbf{O}(l_1 \mathcal{U} \langle a_3 \rangle \top), \mathbf{O}(X\neg l_1)\}$
- $\{\mathbf{O}(l_1 \mathcal{U} \langle a_1 \rangle \top), \mathbf{O}(l_1 \mathcal{U} \langle a_2 \rangle \top), \mathbf{O}(\neg l_1 \mathcal{U} \langle a_3 \rangle \top), \mathbf{O}(\neg l_1 \mathcal{U} \langle a_4 \rangle \top)\}$

The (Only if) part is obvious. The (If) part of this proposition can be proved by showing that a state $w^{(\sigma,S)}_{a_1 \ldots a_k}$ satisfying the above condition is satisfiable in a DDLTL model. We omit the proof. Observe that, although the obligation $\mathbf{O}(l_1 \mathcal{U} l_2)$ is inconsistent with the obligation $\mathbf{O}(\neg l_2 \mathcal{U} \neg(l_1 \wedge l_2))$ (as $\neg l_2 \mathcal{U}(\neg l_1 \wedge \neg l_2)$ entails $\neg(l_1 \mathcal{U} l_2)$), the second obligation is not admitted in our action language.

The conditions above can be formulated by introducing in the domain description causal laws to identify the answer sets falsifying these conditions (see section 7). The violation of the above consistency conditions discloses an inconsistency in the definition of the rules describing the obligations, which has to be resolved by modifying the rules (and possibly introducing preferences among them).

5. PROPAGATION PROPERTIES FOR OBLIGATIONS

In the definition of DDLTL we have not assumed any propagation property for obligations, to admit situations in which actions may cancel an obligation. However, some kind of propagation is needed. If we have an obligation to pay within a deadline, encoded as $\mathbf{O}(\neg deadline \mathcal{U} paid)$[2], then we expect that, if the obligation has not been fulfilled in a given state and the deadline has not yet occurred, the obligation is still valid in the next state. However, we do not want the obligation to be propagated in case the obligation itself is canceled. We then define for an obligation $\mathbf{O}(l_1 \mathcal{U} l_2)$ a defeasible propagation property as follows:

$$X\mathbf{O}(l_1 \mathcal{U} l_2) \leftarrow \mathbf{O}(l_1 \mathcal{U} l_2), l_1, not\ X\neg\mathbf{O}(l_1 \mathcal{U} l_2)$$

meaning that, in any state, if there is an obligation $\mathbf{O}(l_1 \mathcal{U} l_2)$, which is not fulfilled in that state and it is still possible to fulfill it (l_1 is true), then the obligation persists to the next state, unless the obligation is cancelled. An action which cancels an obligation has the effect of making its negation $\neg\mathbf{O}(l_1 \mathcal{U} l_2)$ hold. Note that $\neg l_2$ is not included in the body of the propagation property. In fact, propagation is blocked when the obligation is fulfilled: indeed, we assume that both the fulfillment and the violation of a commitment discharge the obligation, thus blocking its persistency:

$$X\neg\mathbf{O}(l_1 \mathcal{U} l_2) \leftarrow \mathbf{O}(l_1 \mathcal{U} l_2), l_2 \quad \text{(fulfillment)}$$
$$X\neg\mathbf{O}(l_1 \mathcal{U} l_2) \leftarrow \mathbf{O}(l_1 \mathcal{U} l_2), \neg l_1, \neg l_2 \quad \text{(violation)}$$

The fulfillment rule says that, the *fulfillment* of the obligation causes the obligation to be discharged. The violation rule says that the *violation* of the obligation causes the obligation to be discharged. In order to keep track of the fulfillment or violation of obligations during the execution of a business process, and to express in a uniform way the formulae to be verified (see section 7) we introduce, for each obligation $\mathbf{O}(A)$, the simple fluents: $fulfilled_{\mathbf{O}(A)}$ and $violated_{\mathbf{O}(A)}$. For the obligation $\mathbf{O}(l_1 \mathcal{U} l_2)$, we add the rules:

$$fulfilled_{\mathbf{O}(l_1 \mathcal{U} l_2)} \leftarrow \mathbf{O}(l_1 \mathcal{U} l_2), l_2$$
$$violated_{\mathbf{O}(l_1 \mathcal{U} l_2)} \leftarrow \mathbf{O}(l_1 \mathcal{U} l_2), \neg l_1, \neg l_2$$

Similar rules are introduced for obligations of the form $\mathbf{O}(l_1 \mathcal{U} \langle a \rangle l_2)$, replacing l_2 in the rules above with $\langle a \rangle l_2$.

The persistency, fulfillment and violation rules for obligations of the form $\mathbf{O}(Xl)$ are the following:

$$X\mathbf{O}(l) \leftarrow \mathbf{O}(Xl), not\ \neg X\mathbf{O}(l)$$
$$Xfulfilled_{\mathbf{O}(Xl)} \leftarrow \mathbf{O}(Xl), Xl, not\ \neg X\mathbf{O}(l)$$
$$Xviolated_{\mathbf{O}(Xl)} \leftarrow \mathbf{O}(Xl), X\overline{l}, not\ \neg X\mathbf{O}(l)$$

where $\overline{l}$ is the complement of l. The first rule is a defeasible version of the perfect recall property in [6]: if the obligation $\mathbf{O}(Xl)$ occurs in a state, the obligation $\mathbf{O}(l)$ is added to the next state, unless cancelled. According to the 2nd and 3rd rules, an obligation $\mathbf{O}(Xl)$ is fulfilled if l holds in the next state, and is violated if $\neg l$ holds in the next state, unless the obligation is cancelled. Observe that, given the fulfillment and violation rules for $\mathbf{O}(Xl)$ the propagation axiom for $\mathbf{O}(Xl)$ can be omitted. Unlike for the obligation $\mathbf{O}(l_1 \mathcal{U} l_2)$, we can decide about the fulfillment or violation of the obligation $\mathbf{O}(Xl)$ without propagating it to the next state.

For obligations of the form $\mathbf{O}(l)$, we only introduce the fulfillment and violation rules:

$$fulfilled_{\mathbf{O}(l)} \leftarrow \mathbf{O}(l), l$$
$$violated_{\mathbf{O}(l)} \leftarrow \mathbf{O}(l), \neg l$$

[2]We discuss in the next section the representation of obligations with deadlines.

Observe also that causal laws are essential to deal with the dynamic of obligations and to encode persistency, fulfillment and violation of obligations.

6. SPECIFYING OBLIGATIONS

In this section we show that temporal deontic ASP is well suited to formalize the different kinds of obligations introduced in [22, 21] where three main classes are identified, namely, achievement, maintenance and punctual obligations. Achievement obligations are further classified as persistent/non-persistent obligations, preemptive/ non-preemptive obligations.

Achievement obligations. *Achievement obligations* require a given condition to occur at least once before a deadline. Consider the example from [21]: customers must pay before the delivery of the goods, after receiving the invoice. The action of receiving the invoice has the effect of generating an obligation to pay within a deadline, i.e., before receiving the goods. This can be modeled by the action law (with empty body):

$$[receive\_invoice]\mathbf{O}(\neg goods\mathcal{U}pay)$$

According to the persistency, fulfillment and violation laws in the previous section, the achievement obligation $\mathbf{O}(\neg goods\mathcal{U}pay)$ persists until it is fulfilled, cancelled or violated. If the customer executes the action of canceling the order, the obligation is canceled (and discharged):

$$[cancel\_order]\neg\mathbf{O}(\neg goods\mathcal{U}pay)$$
$$[cancel\_order]cancelled_{\mathbf{O}(\neg goods\mathcal{U}pay)}$$

This blocks the persistency of the obligation, and the cancellation is, again, recorded explicitly as a $cancelled_{\mathbf{O}(A)}$ fluent. If not fulfilled or canceled, the obligation persists until the deadline.

Observe that, in case the deadline does not occur (goods are never sent) the obligation $\mathbf{O}(\neg goods\mathcal{U}pay)$ requires, anyhow, that the payment is eventually done. A weaker notion of obligation could be defined by replacing the until operator with the *weak until* one, or by modeling deadline obligations as $\mathbf{O}(\neg(\neg pay\mathcal{U}\neg goods))$. Such alternative formulations are closer to the notion of deadline obligation studied in [7]. Such obligations, however, cannot be encoded in the restricted syntax of our action language.

Nevertheless, according to the propagation properties, an obligation $\mathbf{O}(\neg goods\mathcal{U}pay)$ which has neither been violated not fulfilled (goods have not been sent and the payment has not been done), persists and is not canceled. It can be considered as being "pending" and its presence can be detected.

It is then a matter of choice, during compliance verification, to consider the presence of such "pending" obligations as pointing out a violation or not, according to a stronger or weaker notion of compliance we want to verify (we refer to Section 7).

Contrary-to-duty obligations. In some cases, an achievement obligation can persist even after the deadline (a *persistent obligation* in [21]). Actually, in a more complete version of the example above, there is a *contrary-to-duty obligation*: the violation of the obligation to pay before goods are sent causes a new obligation (to pay with a fine) within the end of the business process execution to be generated:

$$X\mathbf{O}(\neg end\mathcal{U}pay\_with\_fine) \leftarrow violated_{\mathbf{O}(\neg goods\mathcal{U}pay)}$$

meaning that, under the violation of the obligation to pay before goods are sent, a new obligation is added (in the next state) to pay with fine within the end of the business process execution. As here we check for the compliance of *finite* business process executions, we assume that all finite process executions reach an end state. The fulfillment of the obligation to pay with fine *compensates* the earlier violation, i.e.

$$compensated_{\mathbf{O}(\neg goods\mathcal{U}pay)} \leftarrow fulfilled_{\mathbf{O}(\neg end\mathcal{U}pay\_fine)}$$

Following [21], we want to identify those violations which are compensated, to recognize sub-ideal situations in which the process is not fully compliant, but all the violations have been compensated (see again section 7). We consider the original obligation compensated also in case the new obligation is itself (violated and) compensated, or it is cancelled:

$$compensated_{\mathbf{O}(\neg goods\mathcal{U}pay)} \leftarrow compensated_{\mathbf{O}(\neg end\mathcal{U}pay\_fine)}$$
$$compensated_{\mathbf{O}(\neg goods\mathcal{U}pay)} \leftarrow cancelled_{\mathbf{O}(\neg end\mathcal{U}pay\_fine)}$$

Note that $cancelled_{\mathbf{O}(A)}$ only holds as effect of some action (like $cancel\_order$ above) cancelling the obligation, not in case $\mathbf{O}(A)$ is discharged for other reasons (fulfillment or violation).

The easiness to model the fact that "a violation causes a new obligation" in a causal and temporal deontic framework, was already observed in [38], where a temporal deontic logic based on causal theories was introduced.

To represent contrary-to-duty obligations, [23, 21] exploit obligation chains of the form $OA \otimes OB \otimes OC$, (meaning that "OB is the reparation of the violation of OA and OC is the reparation of the violation of OB") which may occur in the head of rules. The causal rules above are well suited to represent such cascaded obligations.

Maintenance obligations. Maintenance obligations require a condition to obtain during all instants before a deadline. For instance, (from [21]) after opening a bank account, customers must keep a positive balance until bank charges are taken out. The effect of action *opening_account* is modeled by the action law:

$$[opening\_account]\mathbf{O}(pos\_balance\mathcal{U}charges\_taken\_out)$$

A maintenance obligation persists until the deadline is reached or the obligation is violated or cancelled. The dynamic of maintenance obligations is ruled by persistency, fulfillment and violation laws for until obligations in Section 5.

Punctual obligations. They can be modeled by obligations of the form $\mathbf{O}(Xl)$. Suppose, e.g., that when a system receives a

given message, it must immediately acknowledge it: $\mathbf{O}(Xack) \leftarrow received\_message$. According to the rules in Section 5, if not canceled, this obligation causes the obligation $\mathbf{O}(ack)$ to be added to the next state.

Preemptive and nonpreemptive obligations. [21] distinguishes preemptive and non-preemptive achievement obligations. Consider the following example: after a contract has been signed, a copy has to be sent within a given deadline. If the copy has been sent before the contract has been signed, this does not fulfill the obligation. In this case, the obligation is said to be *non-preemptive*. To capture non-preemptive obligations, we introduce an obligation to execute an action as follows: $\mathbf{O}(\neg deadline \mathcal{U} \langle send\_copy \rangle \top)$. This obligation is fulfilled if the action $send\_copy$ is executed within the deadline (and after the obligation has been generated).

Observe that *defeasibility* has been identified in [35] as one of the crucial aspects in the formalization of business rules. Business rules are inherently defeasible, due to the presence of exceptions. In the context of an ASP language, defeasibility is captured by default negation, and several approaches to the definition of preferences among ASP rules have been proposed in the literature. In particular, [10] introduces a general methodology for expressing preference information among rules by encoding prioritized programs into standard ASP programs. The approach proposed in [10] can be exploited in this setting to model defeasible norms as prioritized defeasible causal laws.

7. COMPLIANCE VERIFICATION

The action theory introduced in Section 3 does not only allow for the specification of the business rules (the norms), it is also well suited for the specification of the business process itself. In particular, the temporal action language can be used both in the specification of the business process workflow (by exploiting its capability to represent complex actions), as well as in the specification of the atomic tasks occurring in it (see [9]). Observe that, as DLTL is an extension of LTL, it is possible to provide an encoding of all ConDec [34] constraints into our action language. The additional expressivity which comes from the presence of program expressions in DLTL, allows for a very compact encoding of certain declarative properties of the domain dealing with finite iteration[3]. Furthermore, following the approach in [17, 24, 41], in which *annotations* are introduced to decorate the business process, we can exploit the temporal action language as an expressive formalism to formulate properties annotations: the effects and, possibly, the preconditions of the atomic tasks can be defined by introducing propositions representing the properties of the world that are affected by the execution of the atomic tasks and are subject to the norms.

Given the specification of the business process (including atomic tasks annotations) and of the business rules in the deontic temporal action language above, several verification tasks can be addressed within the proposed approach, including compliance verification. However, as a preliminary step before compliance verification, the consistency of the rules encoding the norms (and of the annotations describing the effects of atomic tasks) must be verified against the consistency conditions in Section 4. More precisely, we want to exclude that a state is reachable in which not all the conditions in Proposition 3 are satisfied, to avoid, for instance, that contradictory obligations are generated.

For consistency verification, we can introduce a new proposition d_i, representing a *deontic inconsistency* and, for each set of conflicting deontic literals in Proposition 3, we introduce a rule, for instance:

$$d\_i \leftarrow \mathbf{O}(l), \mathbf{O}(\neg l)$$

Then, we can check if there is a possible action sequence starting from an initial state in which $\neg d\_i$ holds and leading to a state in which d_i holds, i.e., an execution satisfying the formula $\neg d_i \wedge \Diamond d_i$. A reachable state in which d_i holds is a state in which there are conflicting obligations, which may have been generated by conflicting rules. Inconsistencies in the definition of business rules have then to be resolved, by modifying the business rules themselves and, possibly, by introducing preferences among them.

The verification that a business process is compliant with a set of business rules [17, 24, 41] consists in verifying that all the norms or business rules are satisfied in all the execution of the process. Here, we distinguish among business rules which can be encoded as temporal formulas not including obligations and business rules whose modeling involves the obligations.

The specification of the norms, the annotations and the business process together define the domain description on which verification is performed. For the rules which can be encoded as temporal formulas, the validity of the formulas has to be checked. As an example, consider, in the order-production-delivery process in [29], the rule "Premium customer status shall only be offered after a prior solvency check". It can be verified by checking validity of the temporal formula

$$\Box(\textit{solvency\_check\_done} \vee \neg\langle \textit{offer\_prem\_status} \rangle \top)$$

in all possible states of the business process, if the action *offer_prem_status* is executable, then *solvency_check_done* must be true. As in the verification we want to check only the run of the process reaching the *end*, we assume the program specification contains the constraint $\Diamond end$, which cuts out all the other unwanted executions (for instance, infinite iterations in internal loops).

In the verification of compliance involving obligations, **full compliance** amounts to check that the obligations which have been generated during the business process execution have not been violated, i.e., that, for all obligations $\mathbf{O}(A)$ occurring in the specification, the formula

$$\Box\neg violated_{O(A)}$$

is valid. The existence of a run satisfying the negation of this formula, for some A, proves that the process is not fully compliant.

In this respect, we have also to consider the fact that there may be obligations of the form $\mathbf{O}(l_1 \mathcal{U} l_2)$ which have neither been violated nor fulfilled, and they are still pending in the *end* state. The possibility that an obligation with a deadline is triggered, but the end is reached without the deadline having occurred, nor the obligation being fulfilled or cancelled, may be evidence of a flaw in the model,

[3]For instance, the property "action b must be executed immediately after any even occurrence of action a in a run" can be expressed by the temporal constraint: $\Box[(a; \Sigma^*; a)^*]\langle b \rangle \top$), where Σ^* represents any finite action sequence.

therefore we may add to our notion of full compliance the requirement that there are no pending obligations in the *end* state, i.e., that for all obligations $\mathbf{O}(A)$ occurring in the specification, the formula

$$\neg\Diamond(end \wedge \mathbf{O}(A))$$

is valid. We can instead define a somewhat weaker notion of compliance by stipulating that the presence of obligations with deadline of the form $\mathbf{O}(\neg goods \mathcal{U} pay)$ in the end state can be accepted and does not affect the compliance.

The notion of **weak compliance** defined in [21] requires that all the violated obligations have been compensated. In our framework its verification requires to check that, for each obligation $\mathbf{O}(A)$ occurring in the specification, the formula

$$\Box(violated_{O(A)} \rightarrow \Diamond compensated_{O(A)})$$

is valid, i.e., if an obligation is violated at some stage, it is compensated later.

The verification task considered in [12], namely the verification of properties of a business process under the assumption that the process satisfies some given business rules, can also be addressed in our approach: the specification of the business rules and their fulfillment condition can be added to the domain specification. The executions of the resulting domain specification can then be verified against other temporal properties. Unlike [12], here we do not deal with data properties and with the verification of first order temporal properties.

In [20] Bounded Model Checking techniques are developed for the verification of DLTL properties of a temporal action theory. The approach in [20] extends the one developed in [26] for bounded LTL model checking with Stable Models. The approach can be used for checking satisfiability of temporal formulas over a temporal action domain, by providing an encoding of both the action domain (action, causal, precondition laws) and of the temporal formula in ASP. Satisfiability can then be checked by running an ASP solver, which computes the temporal answer sets of the action domain satisfying the temporal formula. To prove the validity of a formula, its negation is checked for satisfiability and, in case the formula is not valid, a counterexample is provided.

The same approach can be adopted for the verification of deontic temporal formulas in which deontic formulas are restricted to deontic fluent literals as in Section 3. In such a case, deontic literals play the role of the simple literals in the encoding in [20]. In addition, to guarantee the consistency of deontic fluent literals, a set of constraints has to be added to the encoding of the action domain to exclude those answer sets containing states which are not deontically consistent. The required constraints correspond to the conditions given in Proposition 3.

8. CONCLUSIONS AND RELATED WORK

This paper enhances the approach to business processes compliance verification in [9], where attention was limited to specific kinds of achievement obligations. In [9], obligations are represented as *commitments* (borrowed from the social approach to agent communication [37]), and no temporal formulas may occur within commitments. In this paper, we show that a deontic extension of the temporal ASP language in [20], with restricted kinds of temporal formulas occurring within deontic modalities, allows to model several different kinds of obligations and to capture different notions (full and weak) of compliance. The use of causal laws, both static and dynamic ones, is crucial for the representation of norms, and, in particular, for modeling the dynamics of obligations (such as deadlines and contrary-to-duty obligations). The Deontic Temporal ASP language can be encoded in standard ASP by extending the approach developed in [20] and bounded model checking techniques, extending those in [26], can be used for verification of temporal properties of the business process that go beyond the verification of the fulfillment of the generated obligations.

In [13] a Dynamic Deontic and Temporal Logic has been proposed to reason about obligations and deadlines. In particular, [13] gives a formalization of achievement obligations as obligations with an until formulas as argument. We exploit this idea in a simpler temporal dynamic deontic logic and show that the several kinds of obligations which are relevant for business process verification can be formulated.

In [7] Broersen et al. propose a semantics for deadline obligations in terms of CTL models and show that their operator obeys intuitive properties and avoids some counterintuitive ones, such as agglomeration. As we have observed in Section 6, while their encoding does not fit the syntactic restrictions of our action theory, our notion of deadline obligation $\mathbf{O}(\neg deadline \mathcal{U} p)$ requires that p is eventually true, even in case the deadline does not occur. Nevertheless, from a practical point of view, when evaluating the actual course of actions, we have to stipulate whether pending deadline obligations have to be regarded as violations or not. A discussion of this problem in a multiagent setting can be found in [5], where it is shown that conditional temporal order obligations can be made into deadline obligations when agents do not control avoidance of the deadline condition. In this paper, we do not address the problem of compliance verification of agent strategies.

[39] exploits the temporal logic CTL in the specification of commitment protocols. Temporal formulas can occur within commitments and commitments can be nested (metacommitments). Unlike our approach, [39] does not define an action theory for reasoning about the effects of action executions, and commitments are not regarded as modalities with an associated Kripke semantics.

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. [8] does not address the problem of business process compliance with norms.

Several proposals in the literature introduce annotations on business processes for dealing with compliance verification [17, 24, 41]. In particular, [24] proposes a logical approach to business process compliance based on the idea of annotating the business process. Annotations and normative specifications are provided in the same logical language, namely, the Formal Contract Language (FCL), which combines defeasible logic [2] and deontic logic of violations [23]. In [21] different deontic operators are introduced in PCL for representing the different kinds obligations identified in [22]. The process model is extended with a set of annotations describing the effects of the atomic tasks and the rules describing the obligations. Compliance is verified by traversing the process graph and identifying the effects of tasks and the obligations triggered by each task execution. Algorithms for propagating obligations through the process graph are defined. In our approach, the dynamic of commitments and the propagation properties of obligations are declaratively modeled by a set of causal laws, and verifi-

cation related to obligations is performed by checking the validity of temporal formulas, not differently from the verification of other requirements.

[22] presents a conceptual analysis of several kinds of deadlines in Temporal Modal Defeasible Logic (which combines deontic modalities with temporal intervals), according to which different obligations require distinct compliance conditions. In this paper we have adopted the approach of defining different compliance conditions for the obligations $\mathbf{O}(\alpha \mathcal{U} \beta)$ and $\mathbf{O}(X\alpha)$ and we have used them to provide a characterization of the different kinds of obligations considered in [22]. [22] does not address the problem of propagation of obligations.

The idea of describing the effects of atomic tasks on data through preconditions and effects is already present in [28], where effects and preconditions are sets of atomic formulas, and the background knowledge consists of a theory in clausal form; I-Propagation [41] is exploited for computing annotations. In our approach the domain theory contains directional causal rules, building on work on reasoning about actions and change for adequately representing ramifications (i.e., sides effects of actions).

In [30] Lomuscio and Sergot explore a deontic extension of Interpreted Systems [14] to provide a grounded semantics to deontic concepts. They apply the formal machinery to the analysis of a protocol and show that violations and correct functioning behavior of parts of the system can be represented through normative and epistemic properties.

In [32] the Abductive Logic Programming framework SCIFF is exploited in the declarative specification of business processes as well as in the verification of their properties. In [1] expectations are used for modelling obligations and prohibitions and norms are formalized by abductive integrity constraints.

The approach to business process verification we have presented in this paper is also related with artifact-centric approach process verification in [12]. The problem of capturing data awareness in the approach to verification based on Temporal ASP has been addressed in [19].

9. 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] G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Representation results for defeasible logic. *ACM Trans. on Computational Logic*, 2:255–287, 2001.

[3] M. Baldoni, A. Martelli, V. Patti, and L. Giordano. Programming rational agents in a modal action logic. *Ann. Math. Artif. Intell.*, 41(2-4), 2004.

[4] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[5] J. Broersen. Strategic deontic temporal logic as a reduction to atl, with an application to chisholm's scenario. In *DEON 06, LNCS 4048*, pages 53–68, 2006.

[6] J. Broersen and J. Brunel. 'What I fail to do today, I have to do tomorrow': A logical study of the propagation of obligations. In *CLIMA, LNCS 5056*, pages 82–99, 2007.

[7] J. Broersen, F. Dignum, V. Dignum, and J.-J. Ch. Meyer. Designing a deontic logic of deadlines. In *DEON 04, LNCS 3065*, pages 43–56, 2004.

[8] A.K. Chopra and M.P. Singh. 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] J. P. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, 2003.

[11] R. Demolombe and M. del Pilar Pozos Parra. A simple and tractable extension of situation calculus to epistemic logic. In *ISMIS*, pages 515–524, 2000.

[12] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

[13] F. Dignum and R. Kuiper. Combining dynamic deontic logic and temporal logic for the specification of deadlines. In *HICSS (5)*, pages 336–346, 1997.

[14] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[15] M. Gelfond. Answer Sets. *Handbook of Knowledge Representation, chapter 7, Elsevier*, 2007.

[16] M. Gelfond and V. Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.

[17] A. Ghose and G. Koliadis. Auditing business process compliance. *ICSOC, LNCS 4749*, pages 169–180, 2007.

[18] G. De Giacomo and M. Lenzerini. Tbox and abox reasoning in expressive description logics. In *KR*, pages 316–327, 1996.

[19] L. Giordano, A. Martelli, M. Spiotta, and D. Theseider Dupré. Business processes verification with temporal ASP: from process annotations to data awareness. In *Proc. KIBP 2012*.

[20] L. Giordano, A. Martelli, and D. Theseider Dupré. Reasoning about actions with temporal answer sets. *Theory and Practice of Logic Programming*, 13:201–225, 2013.

[21] G. Governatori. Law, logic and business processes. In *Third International Workshop on Requirements Engineering and Law*. IEEE, 2010.

[22] G. Governatori, J. Hulstijn, R. Riveret, and A. Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Australian Conference on Artificial Intelligence, LNCS 4830*, pages 486–496, 2007.

[23] 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.

[24] G. Governatori and S. Sadiq. The journey to business process compliance. *Handbook of Research on BPM, IGI Global*, pages 426–454, 2009.

[25] D. Harel. Dynamic logic. In *Handbook of Philosophical Logic, vol. 2*, pages 497–604, 1984.

[26] K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.

[27] J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.

[28] J. Hoffmann, I. Weber, and G. Governatori. On compliance

checking for clausal constraints in annotated process models. *Information Systems Frontieres*, 2009.

[29] D. Knuplesch, L. T. Ly, S. Rinderle-Ma, H. Pfeifer, and P. Dadam. On enabling data-aware compliance checking of business process models. In *Proc. ER 2010, 29th International Conference on Conceptual Modeling*, pages 332–346, 2010.

[30] A. Lomuscio and M. J. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.

[31] C. Lutz, F. Wolter, and Michael Zakharyaschev. Temporal description logics: A survey. In *TIME*, pages 3–14, 2008.

[32] M. Montali, P. Torroni, F. Chesani, P. Mello, M. Alberti, and E. Lamma. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundamenta Informaticae*, 102(3-4):325–361, 2010.

[33] H. Palacios and H. Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *AAAI*, pages 900–905, 2006.

[34] M. Pesic and W. 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.

[35] H. Prakken. *Logical Tools for Modelling Legal Argument*. 1997.

[36] Klaus Schild. Combining terminological logics with tense logic. In *EPIA*, pages 105–120, 1993.

[37] M. P. Singh. A social semantics for Agent Communication Languages. *Issues in Agent Communication, LNCS(LNAI) 1916*, pages 31–45, 2000.

[38] L. van der Torre. Causal deontic logic. In *Deon'2000*, 2000.

[39] M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols. *Autonomous Agents and Multi-Agent Systems*, 2(3), 1999.

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

[41] I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: On the verification of semantic business process models. *Distributed and Parallel Databases (DAPD)*, 2010.