Verifying Business Process Compliance by Reasoning about Actions *

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 address the problem of verifying business process compliance with norms. To this end, we employ reasoning about actions in a temporal action theory. The action theory is defined through a combination of Answer Set Programming and Dynamic Linear Time Temporal Logic (DLTL). The temporal action theory allows us to formalize a business process as a temporal domain description, possibly including temporal constraints. Obligations in norms are captured by the notion of commitment, which is borrowed from the social approach to agent communication. Norms are represented using (possibly) non monotonic causal laws which (possibly) enforce new obligations. In this context, verifying compliance amounts to verify that no execution of the business process leaves some commitment unfulfilled. Compliance verification can be performed by Bounded Model Checking.

1 Introduction

Verifying the compliance of business processes towards normative regulations has become an important issue to be addressed. 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. In the financial domain, in particular, 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 investor 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, in order to address the problem of business process compliance verification, we introduce a language for reasoning about action which extends Answer Set Programming. Temporal logic can be usefully exploited both in the

* 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*".

specification of an action domain and in the verification of its properties (see, e.g., [15]). In this paper, we provide a way to specify a business process as a temporal action domain and then we reason about it in the temporal action theory. The same formalism is used for the representation of both processes and norms towards which the process has to be compliant. In particular, causal laws of the action theory are well suited to model norms as directional rules, and defeasible negation of ASP can be exploited to model exceptions to the norms, by allowing norms to be defeasible. To represent the obligations which can be enforced by the application of norms, we make use of a notion of commitment, which is borrowed from the area of multi-agent communication [24, 9, 15].

For the specification and verification of business processes, we rely on a Temporal Action Theory [13], which combines Answer Set Programming with Dynamic Linear Time Temporal Logic (DLTL) [20]. DLTL extends propositional temporal logic of linear time with regular programs of propositional dynamic logic, that are used for indexing temporal modalities. The action language allows for general temporal constraints to be included in the domain description. The definition of action theories based on ASP presents several advantages over the approach in [14], which is based on a monotonic solution for the frame problem. First, the adoption of a non-monotonic solution to the frame problem, based on ASP default negation, allows to avoid the limitation of the completion solution in [14], which requires action and causal laws to be stratified to avoid unexpected extensions in case of cyclic dependencies. Second, ASP allows for a simple definition of defeasible action laws and defeasible causal laws, using default negation. Defeasibility of causal laws is needed if they are used to model norms with exceptions. Finally, bounded model checking [4] can be used for the verification of temporal properties of domain descriptions in temporal ASP. The approach developed in [19] for bounded LTL model checking with Stable Models, has been extended in [13] to deal with DLTL bounded model checking.

Given the specification of a business process as an action domain and the specification of norms as a set of (defeasible) causal rules generating commitments, the problem of compliance verification consists in verifying that there is no execution of the business process which leaves some commitment unfulfilled. This verification can be done using bounded model checking thechniques.

2 Running example

As a running example we consider a fragment of the business process of an investment firm, where the firm offers financial instruments to an investor. The description of the business process in YAWL is given in Figure 1. We chose YAWL (Yet Another Workflow Language) [25] as specification language for our running business process example, since it provides a number of advantages with respect to several available alternatives:

– YAWL has been implemented in an open source workflow system and can be seen as a reference implementation of the workflow patterns (the outcome of an analysis activity based on business process modeling practice) [26].

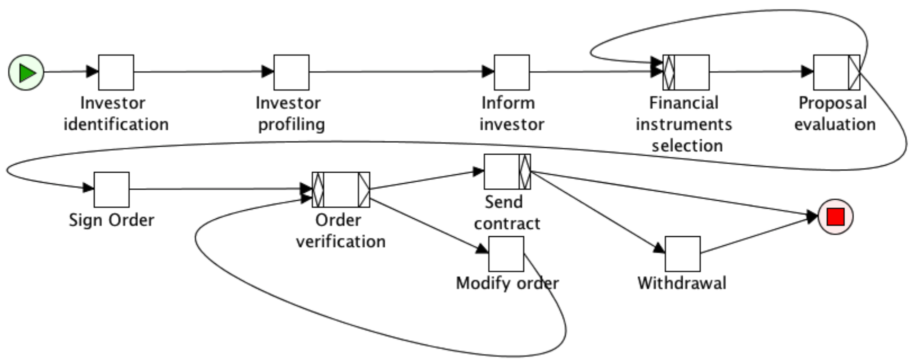

Fig. 1. Example business process in YAWL

- It is the most powerful business process modeling language, with respect to control-flow, data and resource perspectives, the three orthogonal views in a business process specification.
- It has been defined free from commercial interests.
- It provides a graphical user interface, based on a few basic elements, for business process specification needs; this implies a better learning curve.
- It is heavily XML-based, which facilitates interoperability.
- It comes with a formal foundation, which gives the possibility to perform formal analysis for achieving validation and verification goals.

Let us consider a regulation containing the following norms:

(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 may generate obligations to be fulfilled. For instance, according to the second norm, signing an order generates for the firm the obligation to provide 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. We first introduce the action language used, which is based on a temporal extension of answer set programming.

3 Action theories in Temporal ASP

A domain description is defined as a set of laws describing the effects of actions as well as their executability preconditions. 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 properties (fluent) which hold in s and are not affected by the action a, still hold in s'. Let us first describe the notions of fluent and fluent literal.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluent names*. A *simple fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. We will denote by Lit the set of all simple fluent literals. In the language we also make use of temporal literal, that is literals that are prefixed by temporal modalities, as $[a]l$ and $\bigcirc l$. Their intended meaning is the following: $[a]l$ holds in a state when l holds in the state obtained after the execution of action a; $\bigcirc l$ holds in a state if l holds in the next state.

Lit_T is the set of *(temporal) fluent literals*: if $l \in Lit$, then $l \in Lit_T$; if $l \in Lit$, then $[a]l, \bigcirc l \in Lit_T$ (for $a \in \Sigma$, the set of actions). Given a (temporal) fluent literal l, *not* l represents the default negation of l. A (temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*.

A *domain description* D is defined as a tuple $(\Pi, Frame, \mathcal{C})$, where Π contains *action laws*, *causal laws*, *precondition laws* and the *initial state*, $Init$; $Frame$ provides a classification of fluents as frame fluents and non-frame fluents; $\mathcal{C}$ is a set of *temporal constraints*.

The *action laws* in Π have the form:

$$\Box([a]l_1 \ or \ldots or \ [a]l_k \leftarrow l'_1 \wedge \ldots \wedge l'_m)$$

where $l_1, \ldots, l_m$ and $l'_1, \ldots, l'_k$ are simple fluent literals. Its meaning is that executing action a in a state in which the conditions $l'_1, \ldots, l'_m$ hold causes either the effect l_1 or ... or the effect l_k to hold. Consider, for instance, the nondeterministic action of $order\_verification(T, C)$, which checks if the order of the financial product T by customer C is correct or not. In the first case, the order is accepted, otherwise it is not:

$$\begin{array}{l}\Box([order\_verification(T, C)]confirmed(T, C) \ or \\ \quad [order\_verification(T, C)]\neg confirmed(T, C)\end{array}$$

In case of deterministic actions, there is a single disjunct in the head of the action law. For instance, the action of informing the investor has the effect that the investor has acquired information:

$$\Box([inform(C)]informed(C)$$

Causal laws are intended to express "causal" dependencies among fluents. *Static Causal laws* in Π have the form:

$$\Box(l \leftarrow l_1 \wedge \ldots \wedge l_m \wedge not \ l'_1 \wedge \ldots \wedge not \ l'_r)$$

where $l, l_1, \dots, l_m$ are simple fluent literals. Their meaning is that: if $l_1, \dots, l_m$ hold in a state, l is also caused to hold in that state. For instance,

$$\Box(\neg order\_confirmed(T, C) \leftarrow order\_deleted(T, C))$$

where "confirmed" means "confirmed by the firm" and is a possible effect of order verification, while "deleted" means "withdrawn by the customer", models the fact that the direct effect "deleted" of withdrawal has the indirect effect of making the order no longer effective for the firm as well.

Dynamic causal laws in Π have the form:

$$\Box(\bigcirc l \leftarrow l_1, \dots, l_m, \bigcirc l_{m+1}, \dots, \bigcirc l_k)$$

meaning that: if $l_1, \dots, l_m$ hold in a state and $l_{m+1}, \dots, l_k$ hold in the next state, then l is caused to hold in the next state.

Precondition laws have the form:

$$\Box([a]\bot \leftarrow l_1, \dots, l_m)$$

with $a \in \Sigma$ and $l_1, \dots, l_k$ are simple fluent literals. The meaning is that the execution of an action a is not possible if $l_1, \dots, l_k$ hold (that is, no state results from the execution of a in a state in which $l_1, \dots, l_k$ holds). An action for which there is no precondition law is always executable. The precondition law

$$\Box([proposal\_evaluation(T, C)]\bot \leftarrow \neg selected(T, C) \vee \neg informed(C))$$

states that an investor can be requested to evaluate a proposed investment only if the proposal has been selected and the investor has been already informed of the firm policy. Similar preconditions can either be asserted in the model, or verified to be true. The second option is suitable for the case where the process explicitly includes, as in figure 1, activities that make the precondition true; the first one is suitable for the case where such activities are abstracted away.

The *initial state*, $Init$, is a (possibly incomplete) set of simple fluent literals, the fluent which are known to hold initially. For instance, $Init = \{investor, \neg informed, \neg signed, etc.\}$.

The *temporal constraints* in $\mathcal{C}$ are arbitrary temporal formulas of DLTL. They are used to restrict the space of the possible extensions. DLTL [20] extends LTL by allowing the until operator $\mathcal{U}^\pi$ to be indexed by a program π, an expression of Propositional Dynamic Logic (PDL). The usual LTL modalities $\Box$ (always), $\Diamond$ (eventually) 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 τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π. The program π can be any regular expression built from atomic actions using sequence (;), non-deterministic choice (+) and finite iteration ($*$).

As an example of temporal constraint, $\neg \mathit{sent\_contract}\ \mathcal{U}\ \mathit{signed}$ states that the contract is not sent to the customer until it has been signed. A temporal constraint can also require a complex behavior to be performed, through the specification of a program. For instance (the complete version of the program for the process in figure 1 will be given in Section 4), the program

$$\begin{aligned}\pi = {} & inform(C); select\_financial\_instrument(T, C);\\ & ((sign\_order(T, C); send\_contract) + withdraw(T, C))\end{aligned}$$

describes a process in which: the investor C is informed, a financial instrument T is selected for C, C either signs the contract and a copy of the contract is set to him, or C withdraws. The formula $\langle \pi \rangle true$ requires that there is an execution of the program π starting from the initial state.

As in [23, 22] we call *frame* fluents those fluents to which the law of inertia applies. We consider frame fluents as being dependent on the actions. *Frame* is a set of pairs (p, a), where $p \in \mathcal{P}$ is a fluent and $a \in \Sigma$, meaning that p is a frame fluent for action a, that is, p is a fluent to which persistency applies when action a is executed. Instead, non-frame fluents with respect to a do non persist and may change value in a non-deterministically, when a is executed.

Unlike [14], we adopt a non-monotonic solution to the frame problem, as usual in the context of ASP. The persistency of frame fluents from a state to the next one can be enforced by introducing *persistency laws* of the form:

$$\Box([a]l \leftarrow l, \mathit{not}\ [a]\neg\, l),$$

for each simple fluent literal l and action $a \in \Sigma$, such that $(|l|, a) \in Frame$. Its meaning is that, if l holds in a state, then l holds in the state obtained by executing action a, if it can be assumed that $\neg l$ does not hold in the resulting state.

For capturing the fact that a fluent literal l which is non-frame with respect to $a \in \Sigma$ may change its value non-deterministically when a is executed, we introduce the axiom:

$$\Box([a]p\ \mathit{or}\ [a]\neg p \leftarrow true)$$

for all p and a such that $(p, a) \notin Frame$. When a is executed, either the non-frame fluent literal p holds in the resulting state, or $\neg p$ holds. We will call $Frame_D$ the set of laws introduced above for dealing with frame and non-frame fluents. They have the same structure as action laws, but frame axioms contain default negation in their bodies. Indeed, both action laws and causal laws can be extended for free by allowing default negation in their body. This extension has been considered for instance in [8].

As concerns the initial state, we assume that its specification is, in general, incomplete. However, we reason on complete initial states obtained by completing the initial state in all the possible ways. and we assume that, for each fluent literal p, the domain description contains the law:

$$p\ \mathit{or}\ \neg p \leftarrow true$$

meaning that either p is assumed to hold in the initial state, or $\neg p$ is assumed to hold. This approach is in accordance with our treatment of non-deterministic actions and, as we will see, gives rise to extensions in which all states are complete, where each extension represents a run, i.e. a possible evolution of the world from the initial state. We will call $Init_D$ the set of laws introduced above for completing the initial state in all the possibile ways.

4 Specifying a business process as an action domain

In the following, we provide the specification of a business process as an action domain description. 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*, *order_verif* stands for *order verification*. The following action and causal laws describe the effect of the actions in the process:

$$\begin{aligned}
&\Box([investor\_identification(C)]investor(C))\\
&\Box([profiling(C)]investor\_classified(C,D))\\
&\Box([profiling(C)](risk\_averse(C) \textit{ or } risk\_neutral(C) \textit{ or } risk\_seeking(C)))\\
&\Box([inform(C)]informed(C))\\
&\Box([fi\_selection(t_1,C)]selected(t_1,C) \textit{ or} \ldots\\
&\qquad \ldots \textit{or } [fi\_selection(t_n,C)]selected(t_n,C) \leftarrow\\
&\qquad\qquad financial\_instr(t_1) \wedge \ldots \wedge financial\_instr(t_n) \wedge risk\_averse(C))\\
&\ldots\\
&\Box([p\_eval(T,C)]accepted(T,C) \textit{ or } [p\_eval(T,C)]\neg accepted(T,C))\\
&\Box([sign\_order(T,C)]order\_signed(T,C))\\
&\Box([order\_verif](T,C)]order\_confirmed(T,C) \textit{ or}\\
&\qquad [order\_verif](T,C)]\neg order\_confirmed(T,C))\\
&\Box([send\_contract(T,C)]sent\_contract(T,C))\\
&\Box([withdraw(T,C)]order\_deleted(T,C))\\
&\Box(\neg order\_confirmed(T,C) \leftarrow order\_deleted(T,C))\\
&\Box([end\_procedure]end)
\end{aligned}$$

The following precondition laws can either be asserted or verified (see sect. 3):

$$\begin{aligned}
&\Box([p\_eval(T,C)]\bot \leftarrow \neg selected(T,C) \vee \neg informed(C))\\
&\Box([send\_contract(T,C)]\bot \leftarrow \neg confirmed(T,C))
\end{aligned}$$

The meaning of the second one is that it is possible to send a contract to the investor only if the contract has been confirmed.

In order to specify business processes as programs, we introduce *test actions*. DLTL does not include test actions. We define test actions as atomic actions with no effects. For example, $accepted(T,C)?$ is an action testing the result of proposal evaluation in figure 1. Preconditions of $accepted(T,C)?$ are given by the following precondition law:

$$\Box([accepted(T,C)?]\bot \leftarrow \neg accepted(T,C)$$

stating that $accepted(T, C)?$ is not executable in a state in which the investor has not accepted the proposal.

A loop **repeat** *activity* **until** *test* can then be written as follows:

$activity;$
$(\neg test?; activity;)^*$
$test?;$

Note that a regular expression e^* represents the *infinite* set of strings where each string is formed by a *finite* number of occurrences of e. Therefore, only the (infinite) set of finite executions of the loop is represented. This interpretation is consistent with the combination of "classical soundness" and "strong fairness" in Workflow Nets, the class of Petri Nets which form the basis for the semantics of YAWL; see, e.g., the comments in [27] after Definition 8 ("without this assumption, all nets allowing loops in their execution sequences would be called unsound, which is clearly not desirable.").

The control flow of the process in figure 1, which is defined quite rigidly, can be modeled by the following program π:

$investor\_identification(C);$
$profiling(C);$
$inform(C);$
$fi\_selection(T, C);$
$p\_eval(T, C);$
$(\neg accepted(T, C)?; fi\_selection(T, C); p\_eval(T, C))^*;$
$accepted(T, C)?;$
$sign\_order(T, C);$
$order\_verif(T, C);$
$(\neg order\_confirmed(T, C)?; modify\_order(T, C); order\_verif(T, C))^*;$
$order\_confirmed(T, C)?;$
$send\_contract(T, C);$
$(withdraw(T, C) + skip);$
$end\_procedure$

where the action *skip* is defined as the empty action, with no effect.
Given the specification of the program π given above, we introduce the following constraints in $\mathcal{C}$:

$\langle \pi \rangle True$

meaning that in each extension of the domain description, there exists a state reachable from the initial one through the execution of the program π (in which of course $True$ holds). Namely, the sequence of the actions executed must start with an execution of the program π.

The approach we adopt in this paper for reasoning about actions is well suited for reasoning about infinite action sequences. 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). We introduce the following constraints:

$\Diamond\langle dummy\rangle True$
$\Box[dummy]\langle dummy\rangle True$

stating that: the dummy action must eventually be executed and, from that point on, the dummy action is executed repeatedly. The precondition law:

$\Box[dummy]\bot \leftarrow \neg end$

states that the dummy action cannot be executed if the process has not reached the end point.

Although the above specification of the process is very rigid, as it is given through the program expression π, in general a more flexible specification can be provided by encoding the control flow of the process through the specification of action effects and preconditions (as done, for instance, for flexibly encoding agent protocols in [15]). This is, in general, the approach for translating YAWL processes into a domain description with action effects and preconditions, even though in this paper we do not propose an approach to cover all the expressiveness of the YAWL language.

5 Normative Specification

As we have seen in the previous section, the action theory provides a specification of the business process which allows the effect of atomic tasks in the process to be made explicit. According to the normative specification, the execution of each task in the business process can, in addition, trigger some normative position (obligation, permission, prohibition). For instance, as said above, the *identification* task in the business process in Figure 1, 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 [21]. Their use in social approaches to agent communication is essentially motivated by requirements of verifiability. Among the most significant proposals to use commitments in the specification of protocols (or more generally, in agent communication) are [24, 18, 9]. A notion of commitment for reasoning about agent protocols in a temporal action logic have been proposed in [15]. In [1] an alternative notion to commitment called *expectation* is proposed. We refer to section 7 for a discussion of this approach.

Following [15], we introduce two kinds of commitments (which are regarded as special fluent propositions): *Base-level commitments* having the form $C(i, j, A)$ 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 $CC(i, j, B, A)$ and meaning that agent i is committed to agent j to bring about A, if condition B is brought about.

A base level commitment $C(i,j,A)$ can be naturally regarded as an obligation (namely, $O\ A$, "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 [29]. Our present choice is different from the one in [18], 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 task in the business process and they are "discharged" when they have been fulfilled. A commitment $C(i,j,A)$, 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 committment is fullfilled in a run, it is considered to be satisfied and no longer active: it can be discharged.

Given the notion of commitment introduced above, norms can be modeled as precondition and causal laws which trigger new commitments/obligations. For instance, we can encode the norms in Section 2 by the following precondition and causal laws:

$$\Box([sign\_order(T,C)]\bot \leftarrow \neg informed(C))$$
$$\Box(C(firm,C,sent\_contract(T,C)) \leftarrow order\_signed(T,C))$$

The first one is a precondition for $sign\_order(T,C)$, and it is quite obviously true in the example process model, because $informed(C)$ is the effect of the action $inform(C)$ which is always executed before $sign\_order(T,C)$ is reached (and there is no action making $informed(C)$ false). Verifying preconditions may be more interesting in more complex processes where the action may be reached via several paths.

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. The commitment remains active until some action is executed, which makes $sent\_contract(T,C)$ true. In the business process, the commitment is fulfilled by the execution of the action $send\_contract(T,C)$.

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

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

A commitment to bring about α is considered fulfilled and is discharged (i) as soon as α holds. A conditional commitment $CC(i,j,\beta,\alpha)$ becomes a base-level commitment $C(i,j,\alpha)$ 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 disambiguation. Consider the following example from [17]:

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

Rule r_1 says that a seller has the obligation to apply a discount to premium customers. Rule r_2 says that premium customer are not entitled for a discount in case the order (O) is a special order. Suppose that rule r_2 is explicitly given priority over r_1 $(r_2 > r_1)$. 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:

$$\Box(C(s,m,o,\mathit{discount}) \leftarrow \mathit{sells}(s,m,o) \wedge \mathit{premium}(m) \wedge \mathit{not}\ \mathit{bl}(r_1))$$
$$\Box(\neg C(s,m,o,\mathit{discount}) \leftarrow \mathit{sells}(s,m,o) \wedge \mathit{special\_order}(c) \wedge \mathit{not}\ \mathit{bl}(r_2))$$
$$\Box(\mathit{bl}(r_1) \leftarrow \mathit{sells}(s,m,o) \wedge \mathit{special\_order}(c) \wedge \mathit{not}\ \mathit{bl}(r_2))$$
$$\Box(\neg \mathit{bl}(r_1) \leftarrow \mathit{not}\ \mathit{bl}(r_1))$$
$$\Box(\neg \mathit{bl}(r_2) \leftarrow \mathit{not}\ \mathit{bl}(r_2))$$

(where $\mathit{bl}(r_i)$ means that r_i is blocked) so that rule r_2, when applicable, blocks the application of r_1, but not vice-versa.

In the context of ASP, more general and complex encodings of prioritized rules into standard rules with default negation have been studied in [7] and in [5]. We expect that a similar approach can be exploited in this setting to model defeasible norms as prioritized defeasible causal laws.

Another issue to be addressed when modeling norms is that of formalizing violations and reparation obligations. When an obligation is violated, another obligation can be generated as a reparation of that violation. In [17] this problem is addressed through the definition of reparation chains $OA \otimes OB \otimes OC$, where OB is the reparation of the violation of OA, and OC is the reparation of the violation of OB, so that the rules representing norms can have a reparation chain in the conclusion. For instance, the norm $OPay\_in\_time \otimes OPay\_with\_Interest \leftarrow Invoice$ says that, after the invoice has been received, the obligation to pay in time is generated but, if this obligation is not fulfilled, the obligation to pay with interest is generated. We can represent this norm with the following laws:

$$r_1\colon C(i,j,Pay\_in\_time) \leftarrow Invoice(j,i)$$
$$r_2\colon C(i,j,Pay\_with\_Interest) \leftarrow$$
$$Invoice(j,i) \wedge C(i,j,Pay\_in\_time) \wedge \neg Pay\_in\_time$$
$$r_3\colon \neg C(i,j,Pay\_in\_time) \leftarrow$$
$$Invoice(j,i) \wedge C(i,j,Pay\_in\_time) \wedge \neg Pay\_in\_time$$

The first law states that after i receives the invoice from j, i is committed to j to pay in time; r_2 and r_3 state that, if there is a commitment to pay in time and it is violated, then this commitment is discharged and the commitment to pay with interest is generated. Rule r_3 has priority over r_1.

6 Temporal answer sets and compliance verification

Once the specification of the business process has been given as a domain description in an action theory, the problem of verifying its compliance with some regulation can be modeled as the problem of verifying that all the executions of the business process fulfill the obligations that are generated during the execution of the process. In order to characterize the executions of the business process and to check if they violate some obligation, we introduce the notion of extension of a domain description.

A domain description $D = (\Pi, Frame, \mathcal{C})$, is a general logic program extended with a restricted use of temporal modalities. The action modalities $[a]$ and $\bigcirc$ may occur in front of simple literals within rules and the $\Box$ modality occurs in front of all rules in Π. Following [13], we introduce a notion of *temporal answer set*, extending the notion of *answer set* [10]. The extensions of a domain description are then defined as the temporal answer sets of $\Pi \cup Frame_D \cup Init_D$ satisfying the integrity constraints $\mathcal{C}$.

We define a partial temporal interpretation S as a set of literals of the form $[a_1; \ldots; a_k]l$ where $a_1, \ldots, a_k \in \Sigma$, meaning that literal l holds in S in the state obtained by executing the actions $a_1, \ldots, a_k$ in the order.

Definition 1. *Let $\sigma \in \Sigma^\omega$. A* partial temporal interpretation S over σ *is a set of temporal literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ, and it is not the case that both $[a_1; \ldots; a_k]l$ and $[a_1; \ldots; a_k]\neg l$ belong to S (namely, S is a* consistent *set of temporal literals).*

We define a notion of *satisfiability of a literal l in a temporal interpretation S in the state* $a_1 \ldots a_k$ as follows. A literal l is *true* in a partial temporal interpretation S in the state $a_1 \ldots a_k$ (and we write $S, a_1 \ldots a_k \models_t l$), if $[a_1; \ldots; a_k]l \in S$; a literal l is *false* in a partial temporal interpretation S in the state $a_1 \ldots a_k$ (and we write $S, a_1 \ldots a_k \models_f l$), if $[a_1; \ldots; a_k]\overline{l} \in S$; and, finally, a literal l is *unknown* in a partial temporal interpretation S in the state $a_1 \ldots a_k$ (and we write $S, a_1 \ldots a_k \models_u l$), otherwise.

The notion of satisfiability of a literal in a partial temporal interpretation in a given state, can be extended to temporal literals and to rules in a natural way.

For temporal literals $[a]l$, we have: $S, a_1 \ldots a_k \models_t [a]l$ if $[a_1; \ldots; a_k; a]l \in S$ or $a_1 \ldots a_k, a$ is not a prefix of σ; $S, a_1 \ldots a_k \models_f [a]l$ if $[a_1; \ldots; a_k; a]\overline{l} \in S$ or $a_1 \ldots a_k, a$ is not a prefix of σ; and $S, a_1 \ldots a_k \models_u [a]l$, otherwise.

For temporal literals of the form $\bigcirc l$: $S, a_1 \ldots a_k \models_t \bigcirc l$ if $[a_1; \ldots; a_k; b]l \in S$, for some b with $a_1 \ldots a_k b$ prefix of σ; $S, a_1 \ldots a_k \models_f \bigcirc l$ if $[a_1; \ldots; a_k; b]\overline{l} \in S$, for some b with $a_1 \ldots a_k b$ prefix of σ; $S, a_1 \ldots a_k \models_u \bigcirc l$, otherwise.

For default negation, we have: $S, a_1 \ldots a_k \models_t$ *not* l if $S, a_1 \ldots a_k \models_f l$ or $S, a_1 \ldots a_k \models_u l$; and $S, a_1 \ldots a_k \models_f$ *not* l, otherwise.

The three valued evaluation of conjunctions and disjunctions of literals is defined as usual in ASP (see, for instance, [10]). Finally, we say that a rule $\Box(H \leftarrow Body)$ is satisfied in a partial temporal interpretation S if, for all action sequences $a_1 \ldots a_k$ (including the empty one), $S, a_1 \ldots a_k \models_t Body$ implies $S, a_1 \ldots a_k \models_t H$. We say that a rule $[a_1; \ldots; a_h](H \leftarrow Body)$, is satisfied in a partial temporal interpretation S if $S, a_1 \ldots a_h \models_t Body$ implies $S, a_1 \ldots a_h \models_t H$.

We are now ready to define the notion of answer set for a set of P of rules that do not contain default negation. Let P be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$.

Definition 2. *A partial temporal interpretation S over σ is a* temporal answer set of P *if S is minimal (in the sense of set inclusion) among the partial interpretations satisfying the rules in P.*

We want to define answer sets of a program P possibly containing negation. Given a partial temporal interpretation S over $\sigma \in \Sigma^\omega$, we define the *reduct, P^S, of P relative to S* extending the transformation in [10] to compute a different reduct of P for each prefix $a_1, \dots, a_h$ of σ.

Definition 3. *The* reduct, $P^S_{a_1,\dots,a_h}$, of P relative to S and to the prefix $a_1, \dots, a_h$ of σ *, is the set of all the rules $[a_1; \dots; a_h](H \leftarrow l_1 \wedge \dots \wedge l_m)$, such that $\Box(H \leftarrow l_1 \wedge \dots \wedge l_m \wedge not\ l_{m+1} \wedge \dots \wedge not\ l_k)$ is in P and, for all $i = m+1, \dots, k$, either $S, a_1, \dots, a_h \models_f l_i$ or $S, a_1, \dots, a_h \models_u l_i$, (where $l, l_1, \dots, l_k$ are simple or temporal literals). The* reduct P^S of P relative to S over σ *is the union of all reducts $P^S_{a_1,\dots,a_h}$ for all prefixes $a_1, \dots, a_h$ of σ.*

Definition 4. *A partial temporal interpretation S over σ is an* answer set of P *if S is an answer set of the reduct P^S.*

The definition above is a natural generalization of the usual notion of answer set to programs with temporal rules. Observe that, σ has infinitely many prefixes, so that the reduct P^S is infinite and answer sets are infinite. This is in accordance with the fact that temporal models are infinite. Given to the laws for completing the initial state in $Init_D$, we can prove the following:

Proposition 1. *Given a domain description D over Σ and an infinite sequence σ, any answer set of $\Pi \cup Frame_D \cup Init_D$ over σ is a total answer set over σ.*

It can be shown that, by persistency laws, the execution of an action in a complete state produces a new complete state, which is only determined by the action laws, causal laws and persistency laws executed in that state.

In the following, we define the notion of *extension* of a domain description $D = (\Pi, Frame, \mathcal{C})$ over Σ in two steps: first, we find the answer sets of $\Pi \cup Frame_D \cup Init_D$; second, we filter out all the answer sets which do not satisfy the temporal constraints in $\mathcal{C}$. 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 [20]. Given a total answer set S over σ we define the corresponding temporal model as $M_S = (\sigma, V_S)$, where $p \in V_S(a_1, \dots, a_h)$ if and only if $[a_1; \dots; a_h]p \in S$, 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 5. *An* extension of a domain description $D = (\Pi, Frame, \mathcal{C})$ over *Σ, is any (total) answer set S of $\Pi \cup Frame_D \cup Init_D$ satisfying the constraints in $\mathcal{C}$.*

Notice that, in general, a domain description may have more than one extension even for the same action sequence σ: the different extensions of D with the same σ account for the different possible initial states (when the initial state is incompletely specified) as well as for the different possible effects of nondeterministic actions.

The extensions of the domain description define all the possible executions of the business process. To check if there is an execution which violates some obligation we model the need to fulfil a commitment $C(i, j, \alpha)$ as the temporal formula:

$$\Box(C(i, j, \alpha) \rightarrow \Diamond\alpha)$$

Such formulae, together with the precondition formulae corresponding to norms, is the set of formulae to be verified in order to check compliance. We can then introduce the following definition.

Definition 6. *Let D_B the domain description providing the specification of a business process B and let P_N and C_N be, respectively, the set of precondition laws and causal laws in a set N of norms. The business process B is compliant with N if for each extension S of the domain description $D_B \cup C_N$:*

– *for each precondition law P in P_N, S satisfies P;*
– *for each commitment $C(i, j, \alpha)$ occurring in C_N, S satisfies the formula $\Box(C(i, j, \alpha) \rightarrow \Diamond\alpha)$.*

Consider the domain description $D_B \cup C_N$, including the specification D_B of the business problem example and the causal law

$$\Box(C(firm, C, sent\_contract(T, C)) \leftarrow order\_signed(T, C))$$

Each extension S of the domain description satisfies the temporal formulas

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

Hence, the business process is compliant with the norms.

Observe that a weaker notion of compliance can be defined by weakening the fulfilment condition to $\Box(C(i, j, \alpha) \rightarrow \Diamond(\alpha \vee \neg C(i, j, \alpha))$, meaning that a commitment occurring on a run is weakly fulfilled if there ia a later state in which either α holds or the commitment has been discharged. This notion of fulfillment can be used to deal with reparation chains [17].

In [13] we describe bounded model checking techniques for computing the extensions of a temporal domain description and for verifying temporal properties of a domain description. 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. Temporal constraints, which are part of the domain description, are evaluated over temporal answer sets using *bounded model checking* techniques [4]. The approach proposed for the verification of DLTL formulas extends the one developed in [19] for bounded LTL model checking with Stable Models.

7 Conclusions and related work

The paper deals with the problem of verifying the compliance of business processes with norms. Our approach is based on a temporal extension of ASP for reasoning about actions. Both the business process and the norms are given a specification in a domain description. 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 bounded model checking techniques, which generalize LTL bounded model checking [19].

Temporal rule patterns for regulatory policies are introduced in [12], where regulatory requirements are formalized as sets of compliance rules in a real-time temporal object logic. The proposed REALM approach is based on three central pillars: (1) the domain of discourse of a regulation is captured by a concept model expressed as UML class diagrams (concept model); (2) the regulatory requirements are formalized as a set of logical formulas expressed in a real-time temporal object logic, namely a combination of Alur and Henzinger's Timed Propositional Temporal Logic and many-sorted first-order logic (compliance rule set). (3) Information about the structure of the legal source as well as life-cycle data are captured as separate metadata, which allow to annotate both the concept model and the compliance rule set to ensure traceability from regulatory requirements to model elements and vice versa. The approach is used essentially for event monitoring.

[11] proposes an approach based on annotations of business process models. In detail, the tasks of a business process model in BPMN are annotated with their effects. A suitable mechanism to propagate and cumulate the effects of a task to next ones is presented. Process compliance verification establishes that a business process model is consistent with a set of compliance rules. This approach does not introduce normative concepts.

[17] proposes an 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). FCL combines defeasible logic [3] and deontic logic of violations [16] which allows to represent exceptions as well as to express reparation chains, to reason with violations, and the obligations resulting from violations. The notions of ideal, sub-ideal and non-ideal situations are defined to describe two degrees of compliance between execution paths and FCL constraints. Compliance is verified by traversing the graph describing the process and identifying the effects of tasks and the obligations triggered by the task execution. Algorithms for propagating obligations through the process graph are defined. Obligations are discharged when they are fulfilled. In this paper we provide a characterization the compliance problem as a problem of reasoning about actions, which provides business process specification, normative specification, and compliance verification in an integrated representation and reasoning framework.

In [28] process models are considered in which individual activities are annotated with logical preconditions and effects. A formal execution semantics for annotated business processes is introduced and several verification tasks are defined to check whether the business process control flow interacts correctly with the behaviour of the individual activities.

An approach to compliance based on a commitment semantics in the context of multi-agent systems is proposed in [6]. In this work, the authors formalize notions of conformance, coverage, and interoperability, proving that they are orthogonal to each other. The basic idea of the authors is that for an agent to be compliant with a protocol, it must be conformant with it, and conformance can be checked at design time. Another approach to the verification of agents compliance with protocols, based on a temporal action theory, has been proposed in [15]. These papers do not address the problem of compliance with norms.

The notion of *expectation*, which is alternative to the notion of commitment, has been proposed in [1] for the specification of agent protocols. Expectations are used for modelling obligations and prohibitions in the abductive computational framework SOCS [2]. Intuitively, obligations and prohibitions are mapped into abducible predicates, expressing positive and negative expectations, respectively; norms are formalized by abductive integrity constraints. The paper points out that the abductive proof procedure $\mathcal{S}$CIFF can be used for on-the-fly verification of agents conformance to norms.

References

1. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and Verification of Agent Interaction Protocols in a Logic-based System. *SAC'04*, pages 72–78, 2004.
2. 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.
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. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
5. Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.
6. A.K. Chopra and M.P. Sing. Producing compliant interactions: Conformance, coverage and interoperability. *DALT IV, LNCS(LNAI) 4327*, pages 1–15, 2006.
7. 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.
8. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under Incomplete Knowledge. *Computational Logic*, pages 807–821, 2000.
9. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *AAMAS03*, pages 520–527, 2003.
10. M. Gelfond. Answer Sets. *Handbook of Knowledge Representation, chapter 7, Elsevier*, 2007.
11. A. Ghose and G. Koliadis. Auditing business process compliance. *ICSOC, LNCS 4749*, pages 169–180, 2007.

12. C. Giblin, S. Müller, and B. Pfitzmann. From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. *IBM Reasearch Report*, 2007.
13. 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.
14. L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. *The Logic Journal of the IGPL*, 9(2):289–303, 2001.
15. 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.
16. 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.
17. G. Governatori and S. Sadiq. The journey to business process compliance. *Handbook of Research on BPM, IGI Global*, pages 426–454, 2009.
18. F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems, Springer LNAI 2650*, 2003.
19. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.
20. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.
21. N.R. Jennings. Commitments and Conventions: the foundation of coordination in multi-agent systems. *The knowledge engineering review*, 8(3):233–250, 1993.
22. G.N. Kartha and V. Lifschitz. Actions with Indirect Effects (Preliminary Report). *KR'94*, pages 341–350, 1994.
23. V. Lifschitz. Frames in the Space of Situations. *Artif. Intellig.*, 46:365–376, 1990.
24. M. P. Singh. A social semantics for Agent Communication Languages. *Issues in Agent Communication, LNCS(LNAI) 1916*, pages 31–45, 2000.
25. W. van der Aalst and A. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
26. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.
27. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysiss. *BPM Center Report BPM-08-02, BPMcenter.org*, 2008.
28. Ingo Weber, Jörg Hoffmann, and Jan Mendling. Beyond soundness: On the verification of semantic business process models. *Distributed and Parallel Databases*, 27(3):271–343, 2010.
29. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. *AAMAS'02*, pages 527–534, 2002.

Appendix

A Dynamic Linear Time Temporal Logic

The appendix describes shortly the syntax and semantics of linear time logic DLTL [20].

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, where $\omega = \{0, 1, 2, \ldots\}$. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by *prf(u)* the set of finite prefixes of u.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows:

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

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \to 2^{\Sigma^*}$, which is defined in the standard way.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions containing $\top$ and $\bot$.

$$\mathrm{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^\pi \beta$$

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

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \to 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha \mathcal{U}^\pi \beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$[3], $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha \mathcal{U}^\pi \beta$ is true at τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π.

The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^\pi\alpha$ and $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$.

Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, O (next), $\Diamond$ and $\Box$ operators of LTL can be defined as follows: $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [20], DLTL(Σ) is strictly more expressive than LTL(Σ). In fact, DLTL has the full expressive power of the monadic second order theory of ω-sequences.

[3] We define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.