Reasoning about Actions with Temporal Answer Sets *

Laura Giordano[1], Alberto Martelli[2], and Daniele Theseider Dupré[1]

[1] Università del Piemonte Orientale, Dipartimento di Informatica, Alessandria, Italy
[2] Università di Torino, Dipartimento di Informatica, Torino, Italy

Abstract. In this paper we define a Temporal Action Theory through a combination of Answer Set Programming and Dynamic Linear Time Temporal Logic (DLTL). DLTL extends propositional temporal logic of linear time with regular programs of propositional dynamic logic, which are used for indexing temporal modalities. In our language, general temporal constraints can be included in domain descriptions. We define the notion of Temporal Answer Set for domain descriptions, based on the usual notion of Answer Set. Bounded Model Checking techniques are used for the verification of DLTL formulas. The approach can deal with systems with infinite runs.

1 Introduction

In this paper we define a temporal answer set extension of ASP reasoning about action in a Dynamic Linear Time Temporal Logic (DLTL). DLTL extends propositional temporal logic of linear time with regular programs of propositional dynamic logic, which are used for indexing temporal modalities. Allowing program expressions within temporal formulas and including arbitrary temporal formulas in domain descriptions provides a simple way of constraining the (possibly infinite) evolutions of the system, using regular programs (as in Propositional Dynamic Logic). The need of temporally extended goals in a linear temporal logic has been motivated in the context of planning by Bacchus, Kabanza et al. [2, 20]. The formalization of properties of planning domains as temporal formulas in CTL has also been proposed in [17], where the idea of planning as model checking in a temporal logic has been explored, by formalizing planning domains as semantic models. In [13], a monotonic approach to the definition of a temporal action theory based on DLTL has been developed. Following [25], the extensions of a domain description are defined as the temporal models of a completion of the domain description. The temporal projection and planning problem are formalized as satisfiability problems in DLTL, which can be solved by model checking techniques. An algorithm for constructing on-the-fly a Büchi

* 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*".

automaton from a DLTL formula has been proposed in [14], generalizing the tableau-based algorithm for LTL [12].

In this paper we define an extension of ASP rules by allowing temporal modalities and we introduce a notion of temporal answer set, which naturally allows to deal with possibly infinite action sequences. The temporal answer sets which satisfy the constraints in the domain description are recognized as the extensions of the domain description.

We provide a translation into standard ASP rules of action laws, causal laws, etc. in the domain description. The temporal answer sets of an action theory can then be computed as the standard answer sets of the translation. To compute the extensions of a domain description, the temporal constraints which are part of the domain description, are then evaluated over temporal answer sets using *bounded model checking* techniques [5]. The approach proposed for the verification of DLTL formulas extends the one developed in [18] for bounded LTL model checking with Stable Models.

The proposed action theory can deal naturally with systems with infinite runs, and we provide an example concerning a controlled system from the automotive domain.

2 Dynamic Linear Time Temporal Logic

In this section we briefly define the syntax and semantics of DLTL as introduced in [19]. In such a linear time temporal logic the next state modality is indexed by actions. Moreover, (and this is the extension to LTL) the until operator $\mathcal{U}^\pi$ is indexed by a program π as in Propositional Dynamic Logic (PDL). In addition to the usual $\Box$ (always) and $\Diamond$ (eventually) temporal modalities of LTL, new modalities $[\pi]$ and $\langle\pi\rangle$ are allowed. 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. The program π can be any regular expression built from atomic actions using sequence $(;)$, non-deterministic choice $(+)$ and finite iteration $(*)$. The modalities $\Box$, $\Diamond$ and $\bigcirc$ (next) of linear temporal logic can be seen to be derivable.

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 $[[\ ]] : Prg(\Sigma) \rightarrow 2^{\Sigma^*}$, 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 $\bot$. The set of $DLTL$ formulas over Σ is defined as follows:

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

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

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^{\omega}$ and $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ 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 $\mathcal{U}$ (until), $\bigcirc$ (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 [19], 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 Action theories in Temporal ASP

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 denote by $\bar{l}$ the complementary literal of l (namely, $\overline{p} = \neg p$ and $\overline{\neg p} = p$). Also, we denote by Lit the set of all simple fluent literals. 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 $a \in \Sigma$). 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*.

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

The *action laws* in Π have the form:

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

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.

In case of deterministic actions, there is a single disjunct in the head of the action law. For instance, from the Russian turkey problem domain, we have: $\Box([shoot]\neg alive \leftarrow loaded)$ (the action of shooting the turkey makes the turkey dead if the gun is loaded) and $\Box[load]loaded$ (loading the gun makes the gun loaded). An example of non-deterministic action is the action of spinning the gun, after which the gun may be loaded or not: $\Box([spin]loaded \; or \; [spin]\neg loaded \leftarrow true)$.

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

$$\Box(l \leftarrow l_1, \ldots, l_m),$$

where $l, l_1, \ldots, l_m$ are simple fluent literals. Their meaning is that: if $l_1, \ldots, l_m$ hold in a state, l is also caused to hold in that state. For instance, the causal law $\Box(frightened \leftarrow turkey\_in\_sight, alive)$ says that the turkey being in sight causes it to be frightened, if it is alive.

Dynamic causal laws in Π have the form:

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

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

Precondition laws have the form:

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

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

The *initial state*, $Init$, is a (possibly incomplete) set of simple fluent literals, i.e., the fluents which are known to hold initially.

For instance, $Init = \{alive, \neg turkey\_in\_sight, \neg frightened\}$.

As in [22, 21] 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 $p \in \mathcal{P}$ 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 non-deterministically, when a is executed.

Unlike [13], 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:

$$\Box([a]l \leftarrow l, not \; [a]\bar{l}),$$

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 no hold in the resulting state.

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

$$\Box([a]p \; or \; [a]\neg p \leftarrow true)$$

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 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 p, the domain description contains the law:

$$p \; or \; \neg p \leftarrow true$$

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.

The *temporal constraints* in $\mathcal{C}$ are arbitrary temporal formulas of DLTL. They are used to restrict the space of the possible extensions. For instance, $\neg loaded \; \mathcal{U} \; turkey\_in\_sight$ states that the hunter does not load the gun until the turkey is in sight. A temporal constraint can also require a complex behavior to be performed.

The program $\pi = load; ((\neg turkey\_in\_sight)?; wait)^*; (turkey\_in\_sight)?; spin; shoot$ describes the behavior of the hunter who loads the gun, waits for a turkey until it appears and, when there is one in sight, spins the gun and shoots. The action $in\_sight?$ is a *test action*. DLTL does not include test actions. We define a test action $p?$ as an atomic action with no effect, which is executable only if p holds: $\Box([p?]\bot \leftarrow \neg p)$.

4 Example system model

As a further example, we describe an adaptation of the qualitative causal model of the "common rail" diesel injection system from [6, 24] where:

– Pressurized fuel is stored in a container, the *rail*, in order to be injected at high pressure into the cylinders. We ignore in the model the output flow through the injectors.
– Fuel from the tank is input to the rail through a *pump*.
– A regulating system, including, in the physical system, a *pressure sensor*, a *pressure regulator* and an *Electronic Control Unit*, controls pressure in the rail; in particular, the pressure regulator, commanded by the ECU based on the measured pressure, outputs fuel back to the tank.
– The control system repeatedly executes the *sense_pressure* action while the physical system evolves through internal events[4].

Examples of formulas from the model are as follows:

$$\Box([pump\_weak\_fault]f\_in\_low)$$

shows the effect of the fault event $pump\_weak\_fault$.

Flows influence the (derivative of) the pressure in the rail, and the derivative influences pressure, e.g.:

$$\Box(p\_decr \leftarrow f\_out\_ok, f\_in\_low)$$
$$\Box(p\_incr \leftarrow f\_out\_very\_low, f\_in\_low)$$
$$\Box(p\_steady \leftarrow f\_out\_low, f\_in\_low)$$
$$\Box([p\_change]p\_low \leftarrow p\_ok, p\_decr)$$
$$\Box([p\_change]p\_ok \leftarrow p\_low, p\_incr)$$
$$\Box([p\_change]\bot \leftarrow p\_steady)$$
$$\Box([p\_change]\bot \leftarrow p\_decr, p\_low)$$
$$\Box([p\_change]\bot \leftarrow p\_incr, p\_normal)$$

The model of the pressure regulating subsystem includes:

$$\Box([sense\_pressure]p\_obs\_ok \leftarrow p\_ok)$$
$$\Box([sense\_pressure]p\_obs\_low \leftarrow p\_low)$$
$$\Box(f\_out\_ok \leftarrow normal\_mode, p\_obs\_ok)$$
$$\Box([switch\_mode]comp\_mode)$$
$$\Box(f\_out\_very\_low \leftarrow comp\_mode, p\_obs\_low)$$
$$\Box(f\_out\_low \leftarrow comp\_mode, p\_obs\_ok)$$

[4] We make several further abstraction wrt [6, 24]. In particular, we consider "normal" and "low" abstractions for the pressure value, and we assume that the qualitative abstraction of the "normal" range of numeric values is large enough, the frequency (in the real system) of readings from the pressure sensor is high enough, and the reaction of the regulating system is fast and strong enough so that, when the system is driven away from the nominal behavior by a change which is small enough, it quickly evolves to a state which is steady in the qualitative abstraction. Alternatively, temporal constraints can be used to state, e.g., that a *sense_pressure* action is executed at most every k internal events. This choice has, of course, a major influence on properties that hold for the system model. Finding proper abstractions is a key problem in qualitative reasoning (see, e.g., [26]) and we do not aim at solving it here.

Initially, everything is normal and pressure is steady:

p_ok, p_steady, f_in_ok, f_out_ok, $normal\_mode$

We have the following constraints in $\mathcal{C}$:

$\Box(\langle p\_change\rangle true \leftarrow (p\_ok \wedge p\_decr) \vee (p\_low \wedge p\_incr)$

$\Box(\langle switch\_mode\rangle true \leftarrow normal\_mode \wedge p\_obs\_low)$

$[sense\_pressure]\langle(\Sigma - \{sense\_pressure\})^*\rangle\langle sense\_pressure\rangle true$

$\Box[pump\_weak\_fault]\neg\Diamond\langle pump\_weak\_fault\rangle True$

The first models conditions under which a pressure change has to occur (sufficient preconditions). The 2nd models the fact that a mode switch occurs when the system is operating in normal mode and the pressure measured is low. The 3rd one models the fact that the control system repeatedly executes *sense_pressure*, but other actions may occur in between. The 4th imposes that at most one fault may occur in a run.

Given this specification, we can, for instance, check that if pressure is low in one state, it will be normal in the 3rd next one, namely, that the temporal formula $\Box(p\_low \rightarrow \bigcirc\bigcirc\bigcirc p\_ok)$ is satisfied in all the extensions.

Given the observation p_obs_low in a state, we can ask if there is an extension of the domain description which explains it. A diagnosis of the fault is a run from the initial state to a state in which p_obs_low holds and which does not contain previous fault observation in the previous states. In general, we can compute a diagnosis of the fault obs_f by finding an extension of the domain description which satisfies the formula: $(\neg obs_1 \wedge \ldots \wedge \neg obs_n)\ \mathcal{U}\ obs_f$, where $obs_1, \ldots, obs_n$ are all the possible observations of fault. Here, p_obs_low is the only possible fault observation, hence a diagnosis for it is an extension of the domain description which satisfies $\Diamond p\_obs\_low$.

5 Temporal answer sets and extensions for domain descriptions

Given a domain description $D = (\Pi, Frame, \mathcal{C})$, the set of rules in $\Pi \cup Frame_D \cup Init_D$ 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. In order to define the extensions of a domain description, we introduce a notion of *temporal answer set*, extending the notion of *answer set* [11]. The extensions of a domain description will be defined as the temporal answer sets of $\Pi \cup Frame_D \cup Init_D$ satisfying the integrity constraints $\mathcal{C}$.

In the following, for conciseness, we will speak of simple literals and of temporal literal rather than of simple fluent literals or temporal fluent literal. Also, we will call *rules* the laws in $\Pi \cup Frame_D \cup Init_D$, which have the general form:

$$\Box(l'_1\ or \ldots or\ l'_h \leftarrow l_1 \wedge \ldots \wedge l_m \wedge not\ l_{m+1} \wedge \ldots \wedge not\ l_k),$$

where the l'_i, l_j's are simple or temporal literals. Additional rules of the form $[a_1; \ldots; a_h](l'_1\ or \ldots or\ l'_h \leftarrow l_1 \wedge \ldots \wedge l_m)$, where the l'_i, l_j's are simple or temporal literals, will be used in the following.

As we have seen, temporal models of DLTL are linear models, consisting in an action sequence σ and a valuation function V associating a propositional evaluation with each state in the sequence (denoted by a prefix of σ) . To capture this linear structure of temporal models, the notion of answer set has to be extended accordingly. We define a partial temporal interpretation S as a set of literals of the form $[a_1; \ldots; a_k]l$ where $a_, \ldots, a_k \in \Sigma$, meaning that literal l holds in S in the state obtained by executing the actions $a_1; \ldots; a_k$ in the order. Let us define a notion of partial interpretation over σ.

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 $[a_1; \ldots; a_k]\bar{l} \in S$; 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. Given a partial temporal interpretation S over σ, and a prefix $a_1 \ldots a_k$ of σ we say that

$S, a_1 \ldots a_k \models_t [a]l$ 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 $[a_1; \ldots; a_k; a]\bar{l} \in S$ or $a_1 \ldots a_k, a$ is not a prefix of σ

$S, a_1 \ldots a_k \models_u [a]l$, otherwise.

For the temporal literals of the form $\bigcirc l$:

$S, a_1 \ldots a_k \models_t \bigcirc l$ if $[a_1; \ldots; a_k; b]l \in S$, where $a_1 \ldots a_k b$ is a prefix of σ.

$S, a_1 \ldots a_k \models_f \bigcirc l$ if $[a_1; \ldots; a_k; b]\bar{l} \in S$, where $a_1 \ldots a_k b$ is a prefix of σ.

$S, a_1 \ldots a_k \models_u \bigcirc l$, otherwise.

For default negation of a simple or temporal literal l:

$S, a_1 \ldots a_k \models_t not\ l$ if $S, a_1 \ldots a_k \models_f l$ or $S, a_1 \ldots a_k \models_u l$;

$S, a_1 \ldots a_k \models_f not\ l$, otherwise.

The three valued evaluation of conjunctions and disjunctions of literals is defined as usual in ASP (see, for instance, [11]). 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 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 [11] to compute a different reduct of P for each prefix $a_1, \ldots, a_h$ of σ.

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

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 easily transformed into a temporal model as defined in Section 2. Given a total answer set S over σ we define the corresponding temporal model as $M_S = (\sigma, V_S)$, where $p \in V_S(a_1, \ldots, a_h)$ if and only if $[a_1; \ldots; a_h]p \in S$, for all atomic propositions p. We say that a total answer set S over σ satisfies a DLTL formula α if $M_S, \varepsilon \models \alpha$.

Definition 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.

6 Translation to ASP

We now show how to translate a domain description to standard ASP.

A model is a sequence of actions and a valuation function giving the value of fluents in the states of the model. It can be represented in ASP by the predicates $next(State, State')$, $occurs(Action, State)$ and $holds(Literal, State)$. Predicate *next* must be defined so that each state has exactly one successor state. An *action law* $\Box([a]l \leftarrow l_1, \ldots, l_m)$ can be easily translated as $holds(l, S') \leftarrow next(S, S'), occurs(a, S), holds(l_1, S), \ldots, holds(l_m, S)$ and similarly for the other kind of formulas of the action theory[5]. Occurrence of exactly one action in each state is also encoded. To deal with constraints, we use the predicate $sat(Form, S)$, to express satisfiability of a formula DLTL α in a state of a model, where $Form$ is an ASP atom representing formula α, and S is a state. Let $at[\alpha]$ be the ASP atom representing the formula. We exploit the equivalence between regular expressions and finite automata and we make use of an equivalent formulation of DLTL formulas in which *until* formulas are indexed with finite automata rather than regular expressions [14]. Thus we have $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ instead of $\alpha \mathcal{U}^{\pi} \beta$, where $\mathcal{L}(\mathcal{A}(q)) = [[\pi]]$. More precisely, let $\mathcal{A} = (Q, \delta, Q_F)$ be an ϵ-free nondeterministic finite automaton over the alphabet Σ without an initial state, where Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function, and Q_F is the set of final states. Given a state $q \in Q$, we denote with $\mathcal{A}(q)$ an automaton $\mathcal{A}$ with initial state q.

In the definition of predicate *sat* for *until* formulas, we make use of the following axioms [19]:

$\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \equiv (\beta \vee (\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta))$
(q is a final state of $\mathcal{A}$)
$\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \equiv \alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta$
(q is not a final state of $\mathcal{A}$)

The definition of *sat* is:

fluent: $sat(at[f], S) \leftarrow holds(f, S).$

or:
$sat(at[\alpha \vee \beta], S) \leftarrow sat(at[\alpha], S).$
$sat(at[\alpha \vee \beta], S) \leftarrow sat(at[\beta], S).$

neg: $sat(at[\neg\alpha], S) \leftarrow not\ sat(at[\alpha], S).$

until:
$sat(at[\alpha \mathcal{U}^{\mathcal{A}(q)} \beta], S) \leftarrow$
$sat(at[\alpha], S),$
$occurs(a, S),$
$next(S, S'),$
$sat(at[\alpha \mathcal{U}^{\mathcal{A}(q')} \beta], S').$

[5] For instance, the causal law $\Box(\bigcirc l \leftarrow l_1 \wedge \bigcirc l_2)$ is trasnslated to $holds(l, S') \leftarrow next(S, S'), holds(l_1, S), holds(l_2, S')$; the precondition law $\Box([a]\bot \leftarrow l_1, \ldots, l_m)$, to $\leftarrow occurs(a, S),\ holds(l_1, S), \ldots,\ holds(l_m, S)$.

(for each $a \in \Sigma$ and $q' \in \delta(q, a)$)
$sat(at[\alpha \mathcal{U}^{\mathcal{A}(q)} \beta], S) \leftarrow$
$sat(at[\beta], S)$.
(if q is a final state of $\mathcal{A}$)

Since states are complete, we can identify negation as failure with classical negation, thus having a two valued interpretation of DLTL formulas.

We must also add a constraint $\leftarrow$ *not* $sat(at[\alpha], 0)$ for each temporal constraint α in the domain description.

Given a DLTL formula α, we can generate the corresponding ASP rules with the above transformation. It is easy to see that the number of the rules will be finite, since each rule will be applied to a subformula of α, or to a formula derived from an *until* subformula. We say that a formula $\gamma \mathcal{U}^{\mathcal{A}(q')} \beta$ is *derived* from a formula $\gamma \mathcal{U}^{\mathcal{A}(q)} \beta$ if q' is reachable from q in $\mathcal{A}$.

It can be proved that there is a one to one correspondence between the extensions of a temporal domain description and the answer sets of its translation in ASP.

Available ASP systems allow to deal only with finite sets of ground literals, and thus they do not allow to represent infinite sequences of states. To cope with this problem we exploit the property that any infinite LTL (and DLTL) model can be finitely represented by a finite sequence of states with a loop. Of course, the definition of predicate *next* must be modified accordingly, to capture the fact that each state has a unique next state and the last state has one of its predecessors in the sequence as the next state[6]

We show that the above definition of *sat* works also in this case by considering, in particular, the case of *until* formulas. If S is a state belonging to the loop the goal $sat(at[\alpha \mathcal{U}^{\mathcal{A}(q)} \beta], S)$ can depend cyclically on itself. This happens if the only rule which can be applied to prove the satisfiability of $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ or one of its derived formulas in each state of the loop is the first rule of *until*. In this case $sat(\alpha \mathcal{U}^{\mathcal{A}(q)} \beta, S)$ will be undefined, which amounts to say that $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ is false. This is correct, since, if this happens, α must be true in each state of the loop, and β must be false in all states of the loop corresponding to final states of $\mathcal{A}$. Thus, by unfolding the cyclic sequence into an infinite sequence, $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ will never be satisfied.

Following the approach used in *bounded model checking*[5], satisfiability of a domain description D can be proved by iteratively increasing the length k of

[6] Special attention is also required in the computation of the fluents which hold in the state s_i of the sequence which is the next state of the last state (the target of the loop back). Such a state is the next state of two different states (the last state and the state s_{i-1}). To deal with this case, we have added a new predicate $holds\_next(L, S)$, which computes the literals L which are caused to hold in the state next to S, according to the application of the action, causal and persistency laws. The predicate *hold_next* is used to freeze the effects of action execution and is defined as *holds* in the translation above. Then, *holds* is defined from *hold_next*, by the rules: $holds(L, S) : -holds\_next(L, S)$ and $\neg holds(L, S) : -\neg holds\_next(L, S)$.

the sequence searched for, until a cyclic model is found (if one exists). On the other hand, validity of a formula α can be proved, as usual in model checking, by verifying that D extended with $\neg\alpha$ is not satisfiable. For instance, the translation of the domain description in section 4, for a given $k > 5$, has a finite number of answer sets, some in which no fault occurs, some in which the fault occurs in state $0, 1, \ldots, k-6$. Such extensions correspond to temporal answer sets of the given domain description and the formula $\Box(p\_low \rightarrow \bigcirc\bigcirc\bigcirc p\_ok)$ holds in these extensions. The techniques in [5] (in particular, section 5.1) can be used, in general, to achieve completeness in proving validity.

In many cases (e.g. planning) we want to reason only on finite prefixes of infinite models. This can be achieved by just adding to the domain description one action *dummy*, and the constraints $\Diamond\langle dummy\rangle true$ and $\Box(\langle dummy\rangle true \rightarrow \langle dummy\rangle\langle dummy\rangle true)$ stating that, from some point on, only *dummy* will be executed. If the ASP computation return a model, it consists of a finite sequence satisfying the domain description ending with a loop of *dummy* actions.

7 Conclusions and related work

In this paper we developed an extension of ASP for dealing with temporal action theories with general DLTL constraints, which include regular programs indexing temporal modalities. The approach naturally deals with non-terminating computations and relies on bounded model checking techniques for the verification of temporal DLTL formulas.

In the last decade, ASP has been shown to be well suited for reasoning about dynamic domains [11]. In [3] Baral and Gelfond provide an encoding in ASP of the action specification language $\mathcal{AL}$, which extends the action description language $\mathcal{A}$ [10] by allowing static and dynamic causal laws, executability conditions and concurrent actions. The proposed approach has been used for planning [23] and diagnosis (see[1]). In [8, 9] a logic-based planning language, $\mathcal{K}$, is presented which is well suited for reasoning about incomplete knowledge and is implemented on the top of the DLV system. In [16, 15] the languages $\mathcal{C}$ and $\mathcal{C}^+$ provide an account of causality and deal with actions with indirect and non-deterministic effects and with concurrent actions. While our action language does not deal with concurrent actions and incomplete knowledge, our proposal extends the ASP approach for reasoning about dynamic domains with infinite computations.

Bounded model checking [5] is based on the idea to search for a counterexample of the property to be checked in executions which are bounded by some integer k. The value of k is increased until a counterexample is found, the problem becomes intractable or a known upper bound is reached. SAT based bounded model checking methods do not suffer from the state explosion problem as the methods based on BDDs. Helianko and Niemelä [18] have developed a compact encoding of bounded model checking of LTL formulas as the problem of finding stable models of logic programs. In this paper, we have defined a similar approach for encoding bounded model checking of DLTL formulas in ASP. While

the construction of a Büchi automaton [19, 14] from a DLTL formula requires a specific machinery to deal with program expressions with respect to the usual construction for LTL, bounded LTL model checking can be naturally extended to deal with program expressions in temporal modalities, by a direct encoding in ASP of the recursive definition of the modalities.

The presence of temporal constraints in our action language has some relation with the work on temporally extended goals in [7, 4]. These papers, however, are concerned with the problem of expressing preferences among goals and exceptions in goal specification.

The action language presented in this work is an ASP formulation of the action theory defined in [13] which, instead, was based on a monotonic solution to the frame problem. Due to the different treatment of the frame problem, the notion of extension defined here is not equivalent to the one in [13]. In particular, the formalization of causal rules in [13] does not allow reasoning by cases. Moreover, the completion solution requires action and causal laws to be stratified to avoid unexpected extensions when action and causal laws contain cyclic dependencies.

References

1. M. Balduccini and M. Gelfond: Diagnostic Reasoning with A-Prolog, *Theory and Practice of Logic Programming*, 3(4-5):425–461, 2003.
2. F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *Annals of Mathematics and AI*, 22:5–27, 1998.
3. C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In *Logic-Based Artificial Intelligence*, 257–279, 2000.
4. C. Baral, J. Zhao: Non-monotonic Temporal Logics for Goal Specification. IJCAI 2007: 236–242.
5. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, Y. Zhu, Bounded model checking. *Advances in Computers* 58: 118–149, 2003.
6. F. Cascio, L. Console, M. Guagliumi, M. Osella, A. Panati, S. Sottano, and D. Theseider Dupré. Generating on-board diagnostics of dynamic automotive systems based on qualitative deviations. *AI Communications*, 12(1):33–44, 1999.
7. Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a Language for Extended Goals. In Proc. AAAI02.
8. T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, Planning under Incomplete Knowledge. *Computational Logic 2000*, 807–821.
9. T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, A logic programming approach to knowledge-state planning: Semantics and complexity. ACM Trans. Comput. Log. 5(2): 206–263, 2004.
10. M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of logic Programming*, 17:301–322, 1993.
11. M. Gelfond. Handbook of Knowledge Representation, chapter 7. Answer Sets. Elsevier, 2007.
12. R. Gerth, D. Peled, M.Y.Vardi and P. Wolper. Simple On-the-fly Automatic verification of Linear Temporal Logic. In *Proc. 15th Work. Protocol Specification, Testing and Verification*, Warsaw, June 1995, North Holland.

13. L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. In *The Logic Journal of the IGPL*, 9(2): 289–303, 2001.
14. L. Giordano and A. Martelli. Tableau-based Automata Construction for Dynamic Linear Time Temporal Logic. *Annals of Mathematics and Artificial Intelligence*, 46(3): 289–315, Springer 2006.
15. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artif. Intell.*, 153(1-2):49–104, 2004.
16. E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *AAAI/IAAI*, pages 623–630, 1998.
17. F. Giunchiglia and P. Traverso. Planning as Model Checking. In *Proc. The 5th European Conf. on Planning (ECP'99)*, 1–20, Durham (UK), 1999.
18. K. Heljanko, I. Niemelä, Bounded LTL model checking with stable models. TPLP 3(4-5): 519–550 (2003)
19. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. in *Annals of Pure and Applied logic*, vol.96, n.1-3, 187–207, 1999.
20. F. Kabanza, M. Barbeau and R.St-Denis Planning control rules for reactive agents. In *Artificial Intelligence*, 95(1997) 67–113.
21. G.N. Kartha and V. Lifschitz. Actions with Indirect Effects (Preliminary Report). In *Proc. KR'94* , 341–350, 1994.
22. V. Lifschitz. Frames in the Space of Situations. In *Artificial Intelligence* , Vol. 46, 365–376, 1990.
23. R. Morales, M. Gelfond, Tran Cao Son, and Phan Huy Tu. Approximation of Action Theories and Its Application to Conformant Planning. *Artificial Intelligence*, 2008.
24. A. Panati and D. Theseider Dupré. Causal simulation and diagnosis of dynamic systems. In *AI*IA 2001: Advances in Artificial Intelligence*, Springer Verlag LNCS 2175.
25. R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, V. Lifschitz, ed., 359–380, Academic Press, 1991.
26. M. Sachenbacher and P. Struss. Task-dependent qualitative domain abstraction. *Artificial Intelligence*, 162(1-2), 121–143, 2005.