TLP **13** (2): 201–225, 2013. © Cambridge University Press 2012
doi:10.1017/S1471068411000639 First published online 25 January 2012

Reasoning about actions with Temporal Answer Sets

LAURA GIORDANO
Dipartimento di Informatica, Università del Piemonte Orientale, Italy
(*e-mail:* laura@mfn.unipmn.it)

ALBERTO MARTELLI
Dipartimento di Informatica, Università di Torino, Italy
(*e-mail:* mrt@di.unito.it)

DANIELE THESEIDER DUPRÉ
Dipartimento di Informatica, Università del Piemonte Orientale, Italy
(*e-mail:* dtd@mfn.unipmn.it)

submitted 15 November 2010; revised 16 April 2011; accepted 31 August 2011

Abstract

In this paper, we combine Answer Set Programming (ASP) with Dynamic Linear Time Temporal Logic (DLTL) to define a temporal logic programming language for reasoning about complex actions and infinite computations. DLTL extends propositional temporal logic of linear time with regular programs of propositional dynamic logic, which are used for indexing temporal modalities. The action language allows general DLTL formulas to be included in domain descriptions to constrain the space of possible extensions. We introduce a notion of Temporal Answer Set for domain descriptions, based on the usual notion of Answer Set. Also, we provide a translation of domain descriptions into standard ASP and use Bounded Model Checking (BMC) techniques for the verification of DLTL constraints.

KEYWORDS: Answer Set Programming, Temporal Logic, Bounded Model Checking

1 Introduction

Temporal logic is one of the main tools used in the verification of dynamic systems. In the last decades, temporal logic has been widely used also in AI in the context of planning, diagnosis, web service verification, agent interaction, and, in general, in most of those areas having to do with some form of reasoning about actions.

The need of temporally extended goals in the context of planning has been first motivated in Bacchus and Kabanza (1998), Kabanza *et al.* (1997), and Giunchiglia and Traverso (1999). In particular, Giunchiglia and Traverso (1999) developed the idea of planning as model checking in a temporal logic, where the properties of planning domains are formalized as temporal formulas in CTL. In general, temporal formulas can be usefully exploited both in the specification of a domain and in

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

the verification of its properties. This has been done, for instance, for modeling the interaction of services on the web (Pistore *et al.* 2005), as well as for the specification and verification of agent communication protocols (Giordano *et al.* 2007). Recently, Claßen and Lakemeyer (2008) have introduced a second order extension of the temporal logic CTL*, $\mathcal{ESG}$, to express and reason about non-terminating Golog programs. The ability to capture infinite computations is important as agents and robots usually fulfill non-terminating tasks.

In this paper, we combine Answer Set Programming (ASP) (Gelfond 2007) with Dynamic Linear Time Temporal Logic (DLTL) (Henriksen and Thiagarajan 1999) to define a temporal logic programming language for reasoning about complex actions and infinite computations. 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 provide a simple way of constraining the (possibly infinite) evolutions of a system, as in Propositional Dynamic Logic (PDL). To combine ASP and DLTL, we define a temporal extension of ASP by allowing temporal modalities to occur within rules and we introduce a notion of Temporal Answer Set, which captures the temporal dimension of the language as a linear structure and naturally allows to deal with infinite computations. A domain description consists of two parts: a set of temporal rules (action laws, causal laws, etc.) and a set of constraints (arbitrary DLTL formulas). The temporal answer sets of the rules in the domain description which also satisfy the constraints are defined to be the extensions of the domain description.

We provide a translation into standard ASP for the temporal rules of 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 are evaluated over temporal answer sets using *bounded model checking* (BMC) techniques (Biere *et al.* 2003). The approach proposed for the verification of DLTL formulas extends the one developed in Heljanko and Niemelä (2003) for bounded LTL model checking with Stable Models.

The outline of the paper is as follows: In Section 2, we recall the temporal logic DLTL. In Section 3, we introduce our action theory in temporal ASP. In Section 4, we define the notions of temporal answer set and extension of a domain description. Section 5 describes the reasoning tasks, while Sections 6 and 7 describe the model checking problem and provide a translation of temporal domain descriptions into ASP. Section 8 is devoted to conclusions and related work.

2 Dynamic linear time temporal logic

In this section, we briefly define the syntax and semantics of DLTL as introduced in Henriksen and Thiagarajan (1999). 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 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

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

model (a sequence of propositional interpretations) 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 usual modalities $\Box$, $\Diamond$, and $\bigcirc$ (next) of LTL are definable.

Let Σ be a finite non-empty alphabet representing actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, and 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^*. For $u \in \Sigma^\infty$, we denote by $prf(u)$ the set of finite prefixes of u. Moreover, we denote by $\leqslant$ the usual prefix ordering over Σ^* namely, $\tau \leqslant \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$, and $\tau < \tau'$ iff $\tau \leqslant \tau'$ and $\tau \neq \tau'$.

The set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ is:

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

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 as follows:

- $[[a]] = \{a\}$;
- $[[\pi_1 + \pi_2]] = [[\pi_1]] \cup [[\pi_2]]$;
- $[[\pi_1;\pi_2]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi_1]] \text{ and } \tau_2 \in [[\pi_2]]\}$;
- $[[\pi^*]] = \bigcup[[\pi^i]]$, where
  - $[[\pi^0]] = \{\varepsilon\}$
  - $[[\pi^{i+1}]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi]] \text{ and } \tau_2 \in [[\pi^i]]\}$, for every $i \in \omega$

where ε is the empty word (the empty action sequence).

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions containing $\top$ and $\bot$ (standing for *true* and *false*), and let $\mathrm{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha\mathcal{U}^\pi\beta$, where $p \in \mathcal{P}$ and α, β range over $\mathrm{DLTL}(\Sigma)$.

A model of $\mathrm{DLTL}(\Sigma)$ 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:

- $M, \tau \models \top$;
- $M, \tau \not\models \bot$;
- $M, \tau \models p$ 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 $\varepsilon \leqslant \tau'' < \tau'$, $M, \tau\tau'' \models \alpha$.

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 classical connectives $\supset$ and $\wedge$ are defined as usual. 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,

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

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: $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, and $\Box\alpha \equiv \neg\Diamond\neg\alpha$ respectively, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \cdots + a_n$. Hence, LTL(Σ) is a fragment of DLTL(Σ). As shown in Henriksen and Thiagarajan (1999), 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 Lit_S the set of all simple fluent literals and, for each $l \in Lit_S$, we denote by $\bar{l}$ the complementary literal (namely, $\bar{p} = \neg p$ and $\overline{\neg p} = p$). Lit_T is the set of *temporal fluent literals*: if $l \in Lit_S$, then $[a]l, \bigcirc l \in Lit_T$ (for $a \in \Sigma$). Let $Lit = Lit_S \cup Lit_T \cup \{\bot\}$. 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 *state* is a set of fluent literals in Lit_S. A state is *consistent* if it is not the case that both f and $\neg f$ belong to the state, or that $\bot$ belongs to the state. A state is *complete* if, for each fluent name $p \in \mathcal{P}$, either p or $\neg p$ belongs to it. The execution of an action in a state may possibly change the values of fluents in the state through its direct and indirect effects, thus giving rise to a new state.

Given a set of actions Σ, a *domain description D over* Σ is defined as a tuple $(\Pi, \mathcal{C})$, where Π is a set of laws (*action laws*, *causal laws*, *precondition laws*, etc.) describing the preconditions and effects of actions, and $\mathcal{C}$ is a set of *DLTL constraints*. While Π contains the laws that are usually included in a domain description, which define the executability conditions for actions, their direct and indirect effects as well as conditions on the initial state, $\mathcal{C}$ contains general DLTL constraints which must be satisfied by the intended interpretations of the domain description. While the laws in Π define conditions on single states or on pairs of consecutive states, DLTL constraints define more general conditions on possible sequences of states and actions. Let us first describe the laws occurring in Π.

The **action laws** describe the immediate effects of actions. They have the form:

$$\Box([a]l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n) \tag{1}$$

where l_0 is a simple fluent literal and the t_i's are either simple fluent literals or temporal fluent literals of the form $[a]l$. Its meaning is that executing action a in a state in which the conditions $t_1, \ldots, t_m$ hold and conditions $t_{m+1}, \ldots, t_n$ do not hold causes the effect l_0 to hold. Observe that a temporal literal $[a]l$ is true in a state when the execution of action a in that state causes l to become true in the next state. For instance, the following action laws describe the deterministic effect of the actions *shoot* and *load* for the Russian Turkey problem:

$$\Box([shoot]\neg alive \leftarrow loaded)$$
$$\Box[load]loaded$$

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

Non-deterministic actions can be defined using default negation in the body of action laws. In the example, after spinning the gun, it may be loaded or not:

$$\Box([spin]loaded \leftarrow not\ [spin]\neg loaded)$$
$$\Box([spin]\neg loaded \leftarrow not\ [spin]loaded)$$

Observe that, in this case, temporal fluent literals occur in the body of action laws.

Causal laws are intended to express "causal" dependencies among fluents. In Π, we allow two kinds of causal laws. **Static causal laws** have the form:

$$\Box(l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n) \tag{2}$$

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

Dynamic causal laws have the form:

$$\Box(\bigcirc l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n) \tag{3}$$

where l_0 is a simple fluent literal and the t_i's are either simple fluent literals or temporal fluent literals of the form $\bigcirc l_i$. Their meaning is: if $t_1, \ldots, t_m$ hold and $t_{m+1}, \ldots, t_n$ do not hold in a state, then l_0 is caused to hold in the next state. Observe that $t_i = \bigcirc l_i$ holds in a state when l_i holds in the next state.

For instance, the static causal law $\Box(frightened \leftarrow in_sight, alive)$ states that the turkey being in sight of the hunter causes it to be frightened, if it is alive; alternatively, the dynamic causal law $\Box(\bigcirc frightened \leftarrow \bigcirc in_sight, \neg in_sight, alive)$ states that if the turkey is alive, it *becomes* frightened (if it is not already) when it *starts* seeing the hunter; but it can possibly become non-frightened later, due to other events, while still being in sight of the hunter.[1]

Besides action laws and causal laws, that apply to all states, we also allow for laws in Π that only apply to the initial state. They are called **initial state laws** and have the form:

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{4}$$

where the l_i's are simple fluent literals. Observe that initial state laws, unlike static causal laws, only apply to the initial state as they are not prefixed by the $\Box$ modality. As a special case, the initial state can be defined as a set of simple fluent literals. For instance, the initial state $\{alive, \neg in_sight, \neg frightened\}$ is defined by the initial state laws:

$$alive \qquad \neg in_sight \qquad \neg frightened$$

Given the laws introduced above, all the usual ingredients of action theories can be introduced in Π. In particular, let us consider the case when $\bot$ can occur as a literal in the head of those laws.

Precondition laws are special kinds of action laws (1) with $\bot$ as effect. They have the form:

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

[1] Shorthands like those in Denecker *et al.* (1998) could be used, even though we do not introduce them in this paper, to express that a fluent or a complex formula is initiated (i.e., it is false in the current state and caused true in the next one).

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

where $a \in \Sigma$ and the l_i's are simple fluent literals. The meaning is that the execution of an action a is not possible in a state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold (that is, no state may result from the execution of a in a state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold).

State constraints that apply to the initial state or to all states can be obtained when $\bot$ occurs in the head of initial state laws (4) or static causal laws (2):

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

The first one means that it is not the case that, in the initial state, $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold. The second one means that there is no state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold.

As in (Lifschitz 1990) we call *frame* fluents those fluents to which the law of inertia applies. The persistency of frame fluents from a state to the next one can be enforced by introducing in Π a set of laws, called **persistency laws**,

$$\Box(\bigcirc f \leftarrow f,\ not \bigcirc \neg f)$$
$$\Box(\bigcirc \neg f \leftarrow \neg f,\ not \bigcirc f)$$

for each simple fluent f to which inertia applies. Their meaning is that, if f holds in a state, then f will hold in the next state, unless its complement $\neg f$ is caused to hold, and similarly for $\neg f$. Note that persistency laws are instances of dynamic causal laws (3). In the following, we use **inertial f** as a shorthand for the persistency laws for f.

The persistency of a fluent from a state to the next one is blocked by the execution of an action which causes the value of the fluent to change, or by a non-deterministic action which may cause it to change. For instance, the persistency of $\neg loaded$ is blocked by *load* and by *spin*.

Examples of non-inertial fluents, for which persistency laws are not included, are those taking a default truth value, as for a spring door which is normally closed:

$$\Box(closed \leftarrow not \neg closed)$$

or those which always change, at least by default, e.g., in case of a pendulum (see Giunchiglia *et al.* 2004) always switching between left and right position:

$$\Box(\bigcirc right \leftarrow \neg right,\ not \bigcirc \neg right)$$
$$\Box(\bigcirc \neg right \leftarrow right,\ not \bigcirc right)$$

Such default action laws play a role similar to that of *inertia rules* in $\mathcal{C}$ (Giunchiglia and Lifschitz 1998), $\mathcal{C}^+$ (Giunchiglia *et al.* 2004), and $\mathcal{K}$ (Eiter *et al.* 2004).

Initial state laws may incompletely specify the initial state. In this paper, we want to reason about complete states so that the execution of an infinite sequence of actions gives rise to a linear model as defined in Section 2. For this reason, we assume that, for each fluent f, Π contains the laws:

$$f \leftarrow not \neg f$$
$$\neg f \leftarrow not\ f$$

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

As we will see later, this assumption in general is not sufficient to guarantee that all the states are complete.

Test actions, useful for checking the value of a fluent in a state in the definition of complex actions, can be defined through suitable laws as follows. Given a simple fluent literal $l \in Lit_S$, the test action $l?$ is executable only if l holds, and it has no effect on any fluent f:

$$\Box([l?]\bot \leftarrow not\ l)$$
$$\Box([l?]f \leftarrow f)$$
$$\Box([l?]\neg f \leftarrow \neg f)$$

The second component of a domain description is the set $\mathcal{C}$ of **DLTL constraints**, which allow very general temporal conditions to be imposed on the executions of the domain description (we will call them extensions). Their effect is that of restricting the space of the possible executions. For instance, the constraint:

$$\neg loaded\ \mathcal{U}\ in_sight$$

states that the gun is not loaded until the turkey is in sight. Its addition filters out all the executions in which the gun is loaded before the turkey is in sight.

A temporal constraint can also require a complex behavior to be performed. The program

$$(\neg in_sight?; wait)^*; in_sight?; load; shoot \tag{5}$$

describes the behavior of the hunter who waits for a turkey until it appears and, when it is in sight, loads the gun and shoots. Actions $in_sight?$ and $\neg in_sight?$ are test actions, as introduced before. If the constraint

$$\langle(\neg in_sight?; wait)^*; in_sight?; load; shoot\rangle\top$$

is included in $\mathcal{C}$, then all the runs of the domain description which do not start with an execution of the given program will be filtered out. For instance, an extension in which in the initial state the turkey is not in sight and the hunter loads the gun and shoots is not allowed. In general, the inclusion of a constraint $\langle\pi\rangle\top$ in $\mathcal{C}$ requires that there is an execution of the program π starting from the initial state.

Example 1

Let us consider a variant of the Yale shooting problem including some of the laws above, and some more stating that: if the hunter is in sight and the turkey is alive, the turkey is frightened; the turkey may come in sight or out of sight (non-deterministically) during waiting.

Let $\Sigma = \{load, shoot, spin, wait\}$ and $\mathcal{P} = \{alive, loaded, in_sight, frightened\}$. We define a domain description (Π,$\mathcal{C}$), where Π contains the following laws:

Immediate effects:

$\Box([shoot]\neg alive \leftarrow loaded)$ $\quad\quad$ $\Box[load]loaded$

$\Box([spin]loaded \leftarrow not[spin]\neg loaded)$ $\quad\quad$ $\Box([spin]\neg loaded \leftarrow not[spin]loaded)$

$\Box([wait]in_sight \leftarrow not[wait]\neg in_sight)$ $\quad\quad$ $\Box([wait]\neg in_sight \leftarrow not[wait]in_sight)$

Causal laws: $\quad\quad$ $\Box(frightened \leftarrow in_sight, alive)$

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

Initial state laws: $alive \quad \neg in_sight \quad \neg frightened$

Precondition laws: $\Box([load]\ \bot \leftarrow loaded)$

All fluents in $\mathcal{P}$ are inertial: **inertial** alive, **inertial** loaded, **inertial** in_sight, **inertial** frightened; and $\mathcal{C} = \{\neg loaded\ \mathcal{U}\ in_sight\}$.

Given this domain description, we may want to ask if it is possible for the hunter to execute a behavior described by program π in (5) so that the turkey is not alive after that execution. The intended answer to the query $\langle\pi\rangle\neg alive$ would be yes, since there is a possible scenario in which this can happen.

Example 2

In order to see that the action theory in this paper is well suited to deal with infinite executions, consider a mail delivery agent, which repeatedly checks if there is mail in the mailboxes of a and b and then it delivers the mail to a or to b, if there is any; otherwise, it waits. Then, the agent starts again the cycle. The actions in Σ are: *begin*, *sense_mail*(a) (the agent verifies if there is mail in the mailbox of a), *sense_mail*(b), *deliver*(a) (the agent delivers the mail to a), *deliver*(b), *wait* (the agent waits). The fluent names are *mail*(a) (there is mail in the mailbox of a) and *mail*(b). The domain description contains the following laws for a:

Immediate effects:

$\Box[deliver(a)]\neg mail(a)$

$\Box([sense_mail(a)]mail(a) \leftarrow\ not\ [sense_mail(a)]\neg mail(a))$

Precondition laws:

$\Box([deliver(a)]\ \bot \leftarrow \neg mail(a))$

$\Box([wait]\ \bot \leftarrow mail(a))$

Their meaning is (in the order) that: after delivering mail to a, there is no mail for a anymore; the action *sense_mail*(a) of verifying if there is mail for a, may (non-monotonically) cause *mail*(a) to become true; in case there is no mail for a, *deliver*(a) is not executable; in case there is mail for a, wait is not executable. The same laws are also introduced for the actions involving b.

All fluents in $\mathcal{P}$ are inertial: **inertial** mail(a), **inertial** mail(b). Observe that, the persistency laws for inertial fluents interact with the immediate effect laws above. The execution of *sense_mail*(a) in a state in which there is no mail for a ($\neg mail(a)$), may either lead to a state in which *mail*(a) holds (by the second action law) or to a state in which $\neg mail(a)$ holds (by the persistency of $\neg mail(a)$).

$\mathcal{C}$ contains the following constraints:

$\langle begin\rangle\top$

$\Box[begin]\langle sense(a); sense(b); (deliver(a) + deliver(b) + wait); begin\rangle\top$

The first one means that the action *begin* must be executed in the initial state. The second one means that, after any execution of action *begin*, the agent must execute *sense*(a) and *sense*(b) in the order, then either deliver the mail to a or to b or wait and, then, execute action *begin* again, to start a new cycle.

We may want to check that if there is mail for b, the agent will eventually deliver it to b. This property, which can be formalized by the formula $\Box(mail(b) \supset \Diamond\neg mail(b))$,

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

does not hold as there is a possible scenario in which there is mail for b, but the mail is repeatedly delivered to a and never to b. The mail delivery agent we have described is not fair.

Example 3

As an example of modeling a controlled system and its possible faults, we describe an adaptation of the qualitative causal model of the "common rail" diesel injection system from Panati and Theseider Dupré (2001) 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_p* (sense pressure) action while the physical system evolves through internal events.

Examples of formulas from the model are contained in Π:

$$\Box([pump_weak_fault]f_in_low)$$

shows the effect of the fault event $pump_weak_fault$. Flows influence the derivative of the pressure in the rail, and the pressure derivative influences pressure via the event p_change:

$\Box(p_decr \leftarrow f_out_ok, f_in_low)$ $\quad$ $\Box(p_incr \leftarrow f_out_very_low, f_in_low)$
$\Box(p_steady \leftarrow f_out_low, f_in_low)$ $\quad$ $\Box([p_change]p_low \leftarrow p_ok, p_decr)$
$\Box([p_change]p_ok \leftarrow p_low, p_incr)$ $\quad$ $\Box([p_change]\bot \leftarrow p_steady)$
$\Box([p_change]\bot \leftarrow p_decr, p_low)$ $\quad$ $\Box([p_change]\bot \leftarrow p_incr, p_ok)$

The model of the pressure regulating subsystem includes:

$\Box([sense_p]p_obs_ok \leftarrow p_ok)$ $\quad$ $\Box(f_out_ok \leftarrow normal_mode, p_obs_ok)$
$\Box([sense_p]p_obs_low \leftarrow p_low)$ $\quad$ $\Box(f_out_low \leftarrow comp_mode, p_obs_ok)$
$\Box([switch_mode]comp_mode)$ $\quad$ $\Box(f_out_very_low \leftarrow comp_mode, p_obs_low)$

with the obvious mutual exclusion constraints among fluents. Initially, everything is normal and pressure is steady: p_ok, p_steady, f_in_ok, f_out_ok, $normal_mode$.

All fluents are inertial. We have the following temporal constraints in $\mathcal{C}$:

$$\Box((p_ok \wedge p_decr) \vee (p_low \wedge p_incr) \supset \langle p_change\rangle\top)$$
$$\Box(normal_mode \wedge p_obs_low \supset \langle switch_mode\rangle\top)$$
$$[sense_p]\langle(\Sigma - \{sense_p\})^*\rangle\langle sense_p\rangle\top$$
$$\Box[pump_weak_fault]\neg\Diamond\langle pump_weak_fault\rangle\top$$

The first one model conditions which imply a pressure change. The second one models the fact that a mode switch occurs when the system is operating in normal mode and the measured pressure is low. The third one models the fact that the control system repeatedly executes *sense_p*, but other actions may occur in between. The fourth one imposes that at most one fault may occur in a run.

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

Given this specification we can, for instance, check that if pressure is low in one state, it will be normal in the third next one, namely, that the temporal formula $\Box(p_low \supset \bigcirc\bigcirc\bigcirc p_ok)$ is satisfied in all the possible scenarios admitted by the domain description. That is, the system tolerates a weak fault of the pump — the only fault included in this model. In general, we could, e.g., be interested in proving properties that hold if at most one fault occurs, or at most one fault in a set of "weak" faults occurs.

As we have seen from the examples, our formalism allows naturally to deal with infinite executions of actions. Such infinite executions define the models over which temporal formulas can be evaluated. In order to deal with cases (e.g., in planning) where we want to reason on finite action sequences, it is easy to see that any finite action sequence can be represented as an infinite one, adding to the domain description an action *dummy*, and the constraints $\Diamond\langle dummy\rangle\top$ and $\Box[dummy]\langle dummy\rangle\top$ stating that action *dummy* is eventually executed and, from that point on, only the action *dummy* is executed. In the following, we will restrict our attention to infinite executions, assuming that the dummy action is introduced when needed.

4 Temporal answer sets and extensions for domain descriptions

Given a domain description $D = (\Pi, \mathcal{C})$, the laws in Π are rules of a general logic program extended with a restricted use of temporal modalities. In order to define the extensions of a domain description, we introduce a notion of *temporal answer set*, extending the usual notion of *answer set* (Gelfond 2007). The extensions of a domain description will then be defined as the temporal answer sets of Π satisfying the integrity constraints $\mathcal{C}$.

In the following, for conciseness, we call "simple (temporal) literals" the "simple (temporal) fluent literals". We call *rules* the laws in Π, having one of the two forms:

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{6}$$

where the l_i''s are simple literals, and

$$\Box(t_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n) \tag{7}$$

where the t_i's are simple or temporal literals, the first one capturing initial state laws, the second one all the other laws. To define the notion of extension, we also need to introduce rules of the form: $[a_1; \ldots; a_h](t_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n)$, where the t_i's are simple or temporal literals, which will be used to define the reduct of a program. The modality $[a_1; \ldots; a_h]$ means that the rule applies in the state obtained after the execution of actions $a_1, \ldots, a_h$. Conveniently, also the notion of temporal literal used so far needs to be extended to include literals of the form $[a_1; \ldots; a_h]l$, meaning that l holds after the action sequence $a_1, \ldots, a_h$.

As we have seen, temporal models of DLTL are linear models, consisting of 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). We extend the notion of answer set to capture this linear structure of temporal models, by defining a partial

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

temporal interpretation as a pair (σ, S), where $\sigma \in \Sigma^\omega$ and S is a set of literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ.

Definition 1
Let $\sigma \in \Sigma^\omega$. A *partial temporal interpretation over* σ is a pair (σ, S), where S is a set of temporal literals of the form $[a_1; \ldots; a_k]l$, such that $a_1 \ldots a_k$ 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 or that $[a_1; \ldots; a_k]\bot$ belongs to S (namely, S is a *consistent* set of temporal literals).

A temporal interpretation (σ, S) is said to be *total* if either $[a_1; \ldots; a_k]p \in S$ or $[a_1; \ldots; a_k]\neg p \in S$, for each $a_1 \ldots a_k$ prefix of σ and for each fluent name p.

Observe that a partial interpretation (σ, S) provides, for each prefix $a_1 \ldots a_k$, a partial evaluation of fluents in the state corresponding to that prefix. The (partial) state $w^{(\sigma,S)}_{a_1 \ldots a_k}$ obtained by the execution of the actions $a_1 \ldots a_k$ in the sequence can be defined as follows:

$$w^{(\sigma,S)}_{a_1 \ldots a_k} = \{l : [a_1; \ldots; a_k]l \in S\}$$

Given a partial temporal interpretation (σ, S) and a prefix $a_1 \ldots a_k$ of σ, we define the *satisfiability of a simple, temporal and extended literal* t *in* (σ, S) at $a_1 \ldots a_k$ (written $(\sigma, S), a_1 \ldots a_k \models t$) as follows:

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

The satisfiability of rule bodies in a partial interpretation is defined as usual:

$(\sigma, S), a_1 \ldots a_k \models t_1, \ldots, t_n$ *iff* $(\sigma, S), a_1 \ldots a_k \models t_i$ for $i = 1, \ldots, n$.

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

A rule $\Box(H \leftarrow Body)$ is satisfied in a partial temporal interpretation (σ, S) if, for all action sequences $a_1 \ldots a_k$ (including the empty one), $(\sigma, S), a_1 \ldots a_k \models Body$ implies $(\sigma, S), a_1 \ldots a_k \models H$.

A rule $[a_1; \ldots; a_h](H \leftarrow Body)$ is satisfied in a partial temporal interpretation (σ, S) if $(\sigma, S), a_1 \ldots a_h \models Body$ implies $(\sigma, S), a_1 \ldots a_h \models H$.

We are now ready to define the notion of answer set for a set P of rules that does 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 (σ, S) is a *temporal answer set of* P if S is minimal (in the sense of set inclusion) among the S' such that (σ, S') is a partial interpretation satisfying the rules in P.

In order to define answer sets of a program P possibly containing negation, given a partial temporal interpretation (σ, S) over $\sigma \in \Sigma^\omega$, we define the *reduct,* $P^{(\sigma,S)}$*, of* P

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

relative to (σ, S) extending Gelfond and Lifschitz' transform (Gelfond and Lifschitz 1988) to compute a different reduct of P for each prefix $a_1, \ldots, a_h$ of σ.

Definition 3
The *reduct,* $P^{(\sigma,S)}_{a_1,\ldots,a_h}$, *of* P *relative to* (σ, S) *and to the prefix* $a_1, \ldots, a_h$ *of* σ is the set of all the rules

$$[a_1; \ldots; a_h](H \leftarrow t_1, \ldots, t_m)$$

such that $\Box(H \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n)$ is in P and $(\sigma, S), a_1, \ldots, a_h \not\models t_i$, for all $i = m+1, \ldots, n$. The *reduct* $P^{(\sigma,S)}$ *of* P *relative to* (σ, S) is the union of all reducts $P^{(\sigma,S)}_{a_1,\ldots,a_h}$ for all prefixes $a_1, \ldots, a_h$ of σ.

In essence, given (σ, S), a different reduct is defined for each finite prefix of σ, i.e., for each possible state corresponding to a prefix of σ.

Definition 4
A partial temporal interpretation (σ, S) is a *temporal answer set of* P if (σ, S) is a temporal answer set of the reduct $P^{(\sigma,S)}$.

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 $P^{(\sigma,S)}$ is infinite as well as its answer sets. This is in accordance with the fact that temporal models are infinite.

In the following, we will devote our attention to those domain descriptions $D = (\Pi, \mathcal{C})$ such that Π has total temporal answer sets. We will call such domain descriptions *well-defined domain descriptions*. As we will see below, total temporal answer sets can indeed be regarded as temporal models (according to the definition of model in Section 2). Although it is not possible to define general syntactic conditions which guarantee that the temporal answer sets of Π are total, this can be done in some specific case. It is possible to prove the following:

Proposition 1
Let $D = (\Pi, \mathcal{C})$ be a domain description over Σ, such that all fluents are inertial. Let $\sigma \in \Sigma^\omega$. Any answer set of Π over σ is a total answer set over σ.

This result is not surprising, since, as we have assumed in the previous section, the laws for completing the initial state are implicitly added to Π, so that the initial state is complete. Moreover, it can be shown that (under the conditions, stated in Proposition 1, that all fluents are inertial) the execution of an action in a complete state produces (non-deterministically, due to the presence of non-deterministic actions) a new complete state, which can be 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 well-defined domain description $D = (\Pi, \mathcal{C})$ over Σ in two steps: first, we find the temporal answer sets of Π; second, we filter out all the temporal 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 (σ, S). Observe that a total answer set (σ, S) can be easily seen as a temporal model, as defined in Section 2. Given a total answer set (σ, S) we define the corresponding temporal model as

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

$M_S = (\sigma, V_S)$, 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 well-defined domain description* $D = (\Pi, \mathcal{C})$ *over* Σ is a (total) answer set (σ, S) of Π which satisfies 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 non-deterministic actions.

Example 4
Assume the *dummy* action is added to the RussianTurkey domain in Section 3. Given the infinite sequence $\sigma_1 = \neg in_sight?; wait; in_sight?; load; shoot; dummy; \ldots$, the domain description has (among the others) an extension (σ_1, S_1) over σ_1 containing the following temporal literals (for the sake of brevity, we write $[a_1; \ldots; a_n](l_1 \wedge \ldots \wedge l_k)$ to mean that $[a_1; \ldots; a_n]l_i$ holds in S_1 for all i's):
$[\varepsilon](alive \wedge \neg in_sight \wedge \neg frightened \wedge \neg loaded)$,
$[\neg in_sight?](alive \wedge \neg in_sight \wedge \neg frightened \wedge \neg loaded)$,
$[\neg in_sight?; wait](alive \wedge in_sight \wedge frightened \wedge \neg loaded)$,
$[\neg in_sight?; wait; in_sight?](alive \wedge in_sight \wedge frightened \wedge \neg loaded)$,
$[\neg in_sight?; wait; in_sight?; load](alive \wedge in_sight \wedge frightened \wedge loaded)$,
$[\neg in_sight?; wait; in_sight?; load; shoot](\neg alive \wedge in_sight \wedge frightened \wedge loaded)$,
$[\neg in_sight?; wait; in_sight?; load; shoot; dummy](\neg alive \wedge in_sight \wedge frightened \wedge loaded)$
and so on. This extension satisfies the constraints in the domain description and corresponds to a linear temporal model $M_{S_1} = (\sigma_1, V_S)$.

To conclude this section, we would like to point out that, given a domain description $D = (\Pi, \mathcal{C})$ over Σ such that Π only admits total answer sets, a *transition system* (W, I, T) can be associated with Π, as follows:

- W is the set of all the possible consistent and complete states of the domain description;
- I is the set of all the states in W satisfying the initial state laws in Π;
- $T \subseteq W \times \Sigma \times W$ is the set of all triples (w, a, w') such that: $w, w' \in W$, $a \in \Sigma$, and for some total answer set (σ, S) of Π: $w = w^{(\sigma,S)}_{[a_1;\ldots;a_h]}$ and $w' = w^{(\sigma,S)}_{[a_1;\ldots;a_h;a]}$

Intuitively, T is the set of transitions between states. A transition labelled a from w to w' (represented by the triple (w, a, w')) is present in T if, there is a (total) answer set of Π, in which w is a state and the execution of action a in w leads to the state w'.

5 Reasoning tasks

Given a domain description $D = (\Pi, \mathcal{C})$ over Σ and a temporal *goal* α (a DLTL formula), we are interested in finding out the extensions of $D = (\Pi, \mathcal{C})$

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

satisfying/falsifying α. While in the next section, we will focus on the use of BMC techniques for answering this question, in this one we show that many reasoning problems, including temporal projection, planning and diagnosis can be characterized in this way.

Suppose that, in Example 1, we want to know if there is a scenario in which the turkey is not alive after the action sequence $\neg in_sight?, wait; in_sight?, load, shoot$. We can solve this *temporal projection problem* by finding out an extension of the domain description which satisfies the temporal formula

$$\langle \neg in_sight?; wait; in_sight?; load; shoot \rangle \neg alive$$

The extension S_1 in Example 4 indeed satisfies the temporal formula above, since $\langle \neg in_sight?; wait; in_sight?; load; shoot \rangle \neg alive$ is true in the linear model $M_{S_1} = (\sigma_1, V_S)$ associated with the extension S_1.

It is well known that a planning problem can be formulated as a satisfiability problem (Giunchiglia and Traverso 1999). In case of complete state and deterministic actions, the problem of finding a plan which makes the turkey not alive and the gun loaded, can be stated as the problem of finding out an extension of the domain description in which the formula $\Diamond(\neg alive \wedge loaded)$ is satisfied. The extension provides a plan for achieving the goal $\neg alive \wedge loaded$.

With an incomplete initial state, or with non-deterministic actions, the problem of finding a conformant/universal plan which works for all the possible completions of the initial state and for all the possible outcomes of non-deterministic actions cannot be simply solved by checking the satisfiability of the formula above. The computed plan must also be tested to be a conformant plan. On the one hand, it must be verified that the computed plan π always achieves the given *Goal*, i.e., there is no extension of the domain description satisfying the formula $\langle \pi \rangle \neg Goal$. On the other hand, it must be verified that π is executable in all initial states. This can be done, for instance, adopting techniques similar to those in Giunchiglia (2000). Eiter *et al.* (2003) addresses the problem of conformant planning in the DLV$^{\mathcal{K}}$ system. (Tu *et al.* 2011) develops conformant planners based on a notion of approximation of action theories in the action language $\mathcal{AL}$ (Baral and Gelfond 2000).

As concerns diagnosis, consider systems like the one in Example 3. A diagnosis of a fault observation obs_f is a run from the initial state to a state in which obs_f holds and which does not contain fault observations in the previous states (Panati and Theseider Dupré 2000), i.e., an extension satisfying 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. In Example 3, *p_obs_low* is the only possible fault observation, hence a diagnosis for it is an extension of the domain description which satisfies $\Diamond p_obs_low$.

As concerns property verification, an example has been given in Example 2. We observe that the verification that a domain description D is well-defined can be done by adding to the domain description a static law $\Box(undefined_fluent \leftarrow not\ f \wedge not\ \neg f)$, for each fluent literal f, and by verifying that there are no extensions in which $\Diamond undefined_fluent$ holds in the initial state.

Other reasoning tasks which can be addressed by checking the satisfiability or validity of formulas in a temporal action theory are multiagent protocol verification

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

(Giordano *et al.* 2007), and verification of the compliance of business processes to norms (D'Aprile *et al.* 2010).

6 Model checking and BMC

LTL is widely used to prove properties of systems by means of *model checking*. The property to be verified can be represented as an LTL formula φ, whereas a Kripke structure provides the model of the system to be verified (in the current case, the transition system associated with the domain description). A standard approach to verification is based on the construction of the Büchi automaton for the negated property and on the computation of the product of such automaton with the model of the system. The property is verified when the language accepted by the product automaton is empty, whereas any infinite word accepted by the product automaton provides a counterexample to the validity of φ. This approach is also feasible for DLTL, as it is possible to construct a Büchi automaton for a given DLTL formula (Henriksen and Thiagarajan 1999). In particular, as for LTL, the construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton (Giordano and Martelli 2006).

In Biere *et al.* (2003) it has been shown that, in some cases, model checking can be more efficient if, instead of building the product automaton and checking for an accepting run on it, we build only an accepting run of the automaton (if there is one). This technique is called *Bounded Model Checking* (BMC), since it looks for paths whose length is bounded by some integer k, by iteratively increasing the length k until a run satisfying $\neg\varphi$ is found (if one exists). The paths considered are infinite paths which can be finitely represented as paths of length k with a back loop from state k to a previous state in the path: it can be shown that, if a Büchi automaton has an accepting run, it has one which can be represented in this way.

A BMC problem can be efficiently reduced to a propositional satisfiability problem (Biere *et al.* 2003) or to an ASP problem (Heljanko and Niemelä 2003). BMC provides a partial decision procedure for checking validity: if no model exists, the iterative procedure will never stop. Techniques for achieving completeness are described in Biere *et al.* (2003).

In the next section, we address the problem of defining a translation of a domain description into standard ASP, so that BMC techniques can be used to check if a temporal goal (a DLTL formula) is satisfiable in some extension of the domain description. The approach, we propose for the verification of DLTL formulas, extends the one developed in (Heljanko and Niemelä 2003) for bounded LTL model checking with Stable Models.

7 Translation to ASP

In this section, we show how to translate a domain description to standard ASP.

A temporal model consists of an infinite sequence of actions and a valuation function giving the value of fluents in the states of the model. States are represented

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

in ASP as integers, starting with the initial state 0. We will use the predicates $occurs(Action, State)$ and $holds(Literal, State)$. Occurrence of exactly one action in each state must be encoded:

$\neg occurs(A,S) \leftarrow occurs(A1,S), action(A), action(A1), A \neq A1, state(S).$

$occurs(A,S) \leftarrow not\ \neg occurs(A,S), action(A), state(S).$

Given a *domain description* $(\Pi, \mathcal{C})$, the rules in Π can be translated as follows. Action laws $\Box([a](\neg)f_0 \leftarrow t_1,\ldots,t_m, not\ t_{m+1},\ldots,not\ t_n)$ are translated to

$$(\neg)holds(f_0,S') \leftarrow state(S), S' = S+1, occurs(a,S), h_1 \ldots h_m, not\ h_{m+1} \ldots not\ h_n$$

where $h_i = (\neg)holds((f_i,S')$ if $t_i = [a](\neg)f_i$ and $h_i = (\neg)holds(f_i,S)$ if $t_i = (\neg)f_i$. Dynamic causal laws $\Box(\bigcirc(\neg)f_0 \leftarrow t_1,\ldots,t_m, not\ t_{m+1},\ldots,not\ t_n)$ are translated to

$$(\neg)holds(f_0,S') \leftarrow state(S), S' = S+1, h_1 \ldots h_m, not\ h_{m+1} \ldots not\ h_n$$

where $h_i = (\neg)holds(f_i,S')$ if $t_i = \bigcirc(\neg)f_i$ and $h_i = (\neg)holds(f_i,S)$ if $t_i = (\neg)f_i$.

Static causal laws (2) are translated in a similar way (replacing S' with S in the head), while *initial state laws* are evaluated in state 0.

Precondition laws $\Box([a]\ \bot\leftarrow l_1,\ldots,l_m, not\ l_{m+1},\ldots,not\ l_n)$ are translated to ASP constraints

$$\leftarrow state(S), occurs(a,S), h_1 \ldots h_m, not\ h_{m+1} \ldots not\ h_n$$

where $h_i = holds(l_i,S)$.

As described in the previous section, we are interested in infinite models represented as *k-loops*, i.e., finite sequences of states from 0 to k with a back loop from state k to a previous state. Thus, we assume a bound k on the number of states.

The above rules compute a finite model from state 0 to state $k+1$. To detect the loop, we must find a state j, $0 \leqslant j \leqslant k$, equal to state $k+1$. This can be achieved by defining a predicate $eq_last(S)$ to check if state S, i.e., equal to state $k+1$, and a predicate $next(S1,S2)$ such that $next(i,i+1)$ for $0 \leqslant i \leqslant k-1$, and $next(k,j)$.

$diff_last(S) \leftarrow state(S), S <= k, fluent(F), holds(F,S), \neg holds(F,k+1).$

$diff_last(S) : -state(S), S <= k, fluent(F), holds(F,k+1), \neg holds(F,S).$

$eq_last(S) : -state(S), S <= k, not\ diff_last(S).$

$next(S,SN) \leftarrow state(S), S < k, SN = S+1.$

$\neg next(k,S) \leftarrow next(k,SS), state(S), state(SS), S \neq SS.$

$next(k,S) \leftarrow state(S), S <= k, not\ \neg next(k,S).$

$\leftarrow next(k,S), not\ eq_last(S).$

The second and third rule for *next* impose that there is exactly one state next to state k; the last constraint imposes that such a state is equal to state $k+1$.

Given a domain description $(\Pi, \mathcal{C})$, we denote by $tr(\Pi)$, the set of rules containing the translation of each law in Π as well as the definitions of *eq_last*, *diff_last* and *next*, as defined above. Observe that an answer set R of $tr(\Pi)$ such that, for each state $i = 1,\ldots,k$, either $holds(p,i) \in R$ or $\neg holds(p,i) \in R$, represents a temporal model as a *k-loop*. The temporal model, $M_R = (\sigma_R, V_R)$ associated with R can be defined as follows:

$\sigma_R = a_1 a_2 \ldots a_j a_{j+1} \ldots a_{k+1} a_{j+1} \ldots a_{k+1} \ldots$

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

where $occurs(a_1,0), occurs(a_2,1), \ldots, occurs(a_{j+1},j), \ldots, occurs(a_{k+1},k)$, $next(k,j)$ (i.e., a_{k+1} leads back to state j) belong to R, and, for all atomic propositions $p \in \mathcal{P}$:

$p \in V_R(\varepsilon)$ if and only if $holds(p,0) \in R$

$p \in V_R(a_1 \ldots a_h)$ if and only if $holds(p,h) \in R$, for $0 < h \leqslant k$

$p \in V_R(a_1 \ldots a_{k+1})$ if and only if $holds(p,j) \in R$.

We can show that there is a one to one correspondence between the temporal answer sets of Π and the answer sets of the translation $tr(\Pi)$. Let $(\Pi, \mathcal{C})$ be a well-defined domain description over Σ.

Theorem 1

(1) Given a temporal answer set (σ, S) of Π such that σ can be finitely represented as a finite path with a k-loop, there is a consistent answer set R of $tr(\Pi)$ such that R and S correspond to the same temporal model.

(2) Given a consistent answer set R of $tr(\Pi)$, there is a temporal answer set (σ, S) of Π (that can be finitely represented as a finite path with a back loop) such that R and S correspond to the same temporal model.

We refer to online Appendix A for the proof.

Let us now come to the problem of evaluating a DLTL formula over the models associated with the answer sets of $tr(\Pi)$. To deal with DLTL formulas, we use the predicate $sat(alpha, S)$, to express satisfiability of a DLTL formula α in a state of a model. As in Giordano and Martelli (2006), we assume that *until* formulas are indexed with finite automata rather than regular expressions, by exploiting the equivalence between regular expressions and finite automata. 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 non-deterministic 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 refer to the following axioms (Henriksen and Thiagarajan 1999):

$$\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \equiv (\beta \vee (\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta)) \quad (q \text{ is a final state of } \mathcal{A})$$

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

In the translation to ASP, DLTL formulas will be represented with terms. In particular, the formula $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ will be represented as $until(A, q, alpha, beta)$. Furthermore, we assume the automaton $\mathcal{A}$ to be described with the predicates $trans(A, Q1, Act, Q2)$ defining the transitions, and $final(A, Q)$ defining the final states. The definition of *sat* is the following:

fluent: $sat(F,S) \leftarrow fluent(F), holds(F,S).$

or: $sat(or(Alpha, Beta), S) \leftarrow sat(Alpha, S).$

$sat(or(Alpha, Beta), S) \leftarrow sat(Beta, S).$

neg: $sat(neg(Alpha), S) \leftarrow not\ sat(Alpha, S).$

until: $sat(until(Aut, Q, Alpha, Beta), S) \leftarrow final(Aut, Q), sat(Beta, S).$

$sat(until(Aut, Q, Alpha, Beta), S) \leftarrow$

$sat(Alpha, S), trans(Aut, Q, Act, Q1), occurs(Act, S),$

$next(S, S1), sat(until(Aut, Q1, Alpha, Beta), S1).$

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

Similar definitions can be given for derived connectives and modalities. For instance, the temporal formulas $\Diamond\alpha$, $\langle a\rangle\alpha$ and $[a]\alpha$ are represented, respectively, by the terms $eventually(t_alpha)$, $diamond(a, t_alpha)$, and $box(a, t_alpha)$, where t_alpha is the term encoding the formula α. The definition of *sat* for such formulas is the following:

eventually: $sat(eventually(Alpha), S) \leftarrow sat(Alpha, S).$
eventually: $sat(eventually(Alpha), S) \leftarrow next(S, S1), sat(eventually(Alpha), S1).$
$\langle a\rangle$: $sat(diamond(A, Alpha), S) \leftarrow occurs(A, S), next(S, S1), sat(Alpha, S1).$
$[a]$: $sat(box(A, Alpha), S) \leftarrow action(A), occurs(B, S), A! = B.$
$[a]$: $sat(box(A, Alpha), S) \leftarrow occurs(A, S), next(S, S1), sat(Alpha, S1).$

Since states are complete, we can identify default negation with classical negation, thus having a two valued interpretation of DLTL formulas. We must also add a constraint $\leftarrow not\ sat(t_alpha, 0)$, for each temporal constraint α in the domain description, where states are represented by numbers, 0 is the initial state and t_alpha is the term encoding the formula α. The presence of the constraint $\leftarrow not\ sat(t_alpha, 0)$, in the translation of the domain description guarantees that α must be satisfied, as the negated formula $not\ sat(t_alpha, 0)$ is not allowed to be true in the answer set.

As an example, the encoding of the temporal constraint

$$\Box[begin]\langle sense(a); sense(b); (deliver(a) + deliver(b) + wait); begin\rangle\top$$

in Example 2, is given by the following rules:

$\leftarrow not\ sat(neg(ev(neg(box(begin, until(aut, q1, true, true))))), 0).$
$trans(aut, q1, sense(a), q2).$
$trans(aut, q2, sense(b), q3).$
$trans(aut, q3, deliver(a), q4).$
$trans(aut, q3, deliver(b), q4).$
$trans(aut, q3, wait, q4).$
$trans(aut, q4, begin, q5).$
$final(aut, q5).$

The first rule encodes the constraint, while the following ones encode the definition of the automaton *aut*, which is equivalent to the regular expression indexing the until formula in the constraint.

It is easy to see that the computation of the satisfiability of a formula α in a given state depends only on a finite set of formulas consisting of the subformulas of α and the formulas 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 is possible to see that the definition of the predicate *sat*, as given above for the base cases (fluent, or, neg, until), provides a correct evaluation of the temporal formulas over the temporal models associated with the translation $tr(\Pi)$ of Π. Let $tr'(\Pi)$ be the set of rules extending the rules in $tr(\Pi)$ with the definition of predicate *sat* above. Let $(\Pi, \mathcal{C})$ be a well-defined domain description over Σ. We can prove the following theorem (the proof can be found in online Appendix A):

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

Theorem 2
Let Π be the set of laws of a well-defined domain description, R an answer set of $tr(\Pi)$ and α a DLTL formula. The temporal model $M_R = (\sigma, V)$ associated with R satisfies α if and only if there is an answer set R' of $tr'(\Pi)$ such that $R \subset R'$ and $sat(t_alpha, 0) \in R'$ (where t_alpha is the term representing the formula α and *trans* and *final* encode the automata indexing the until formulas in α).

The above formulation of *sat* is indeed the direct translation of the semantics of DLTL, which is given for infinite models. Intuitively, we can show that it works also when the model is represented as a *k-loop*, by considering the case of *until* formulas. If S is a state belonging to the loop, the goal $sat(\alpha \mathcal{U}^{\mathcal{A}(q)} \beta, S)$ 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 second 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 not true. 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.

Given a domain description $D = (\Pi, \mathcal{C})$, the translation $tr(D)$ of D contains: the translation $tr(\Pi)$ of Π, the definition of the predicates *sat*, *trans* and *final*, and, for each temporal formula α in $\mathcal{C}$, the constraint $\leftarrow not\ sat(t_alpha, 0)$.

Let $(\Pi, \mathcal{C})$ be a well-defined domain description over Σ. Given Theorems 1 and 2 above, it can be proved that:

Corollary 1
There is a one to one correspondence between the extensions of the domain description D and the answer sets of its translation $tr(D)$ in ASP.

More precisely, each extension of D is in a one to one correspondence with an answer set of $tr(D)$, and both of them are associated with the same temporal model.

Given a temporal formula α, we may want to check if there is an extension of the domain description D satisfying it. To this purpose, as for the temporal formulas in $\mathcal{C}$, we add to the translation $tr(D)$ of D the constraint $\leftarrow not\ sat(t_alpha, 0)$, so that the answer sets falsifying α are excluded.

According to the BMC technique, the search for an extension of the domain description satisfying α is done by iteratively increasing the length k of 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. Let us consider, from Example 2, the property $\Box(mail(b) \supset \Diamond \neg mail(b))$ (if there is mail for b, the agent will eventually deliver it to b). This formula is valid if its negation $\Diamond \neg (mail(b) \supset \Diamond \neg mail(b))$ is not satisfiable. We verify the satisfiability of this formula, by adding to the translation of the domain description the constraint

$\leftarrow not\ sat(ev(neg(impl(mail(b), ev(neg(mail(b)))))), 0)$.

and looking for an extension. The resulting set of rules indeed has extensions, which can be found for $k \geqslant 3$ and provide counterexamples to the validity of the property above. For instance, the extension in which $next(0,1), next(1,2), next(2,3)$,

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

next(3,0), *occurs*(*begin*, 0), *occurs*(*sense_mail*(*a*), 1), *occurs*(*sense_mail*(*b*), 2), *occurs*(*deliver_mail*(*a*),3), where *mail*(*b*) holds in all states, and `mail(a)` only in states 2 and 3, can be obtained for $k = 3$.

In online Appendix B, we provide the encoding of BMC and of Example 2 in the DLV-Complex extension (https://www.mat.unical.it/dlv-complex) of DLV (Leone *et al.* 2006). In online Appendix C, we report tests of our approach for BMC of DLTL formulas in the line of the LTL BMC experiments in (Heljanko and Niemelä 2003). Results are provided for a DLV encoding of BMC and of action domain descriptions for the dining philosophers problems considered in that paper. The scalability of the two approaches is similar.

8 Conclusions and related work

In this paper, we have described an action language which is based on a temporal extension of ASP, in which temporal modalities are included within rules. In the action language, general temporal DLTL formulas (possibly including regular programs indexing temporal modalities) are allowed in the domain description to constrain the space of possible extensions. The approach naturally deals with non-terminating computations and relies on BMC techniques for the verification of temporal formulas. In (Giordano *et al.* 2001) a temporal action theory has been developed, which is based on the linear temporal logic DLTL and adopts a monotonic solution to the frame problem based on completion. Due to the different treatment of the frame problem, even in the case when default negation is not present in the body of the laws in Π, the notion of extension defined here is not equivalent to the one in Giordano *et al.* (2001), which requires action and causal laws to be stratified to avoid unexpected extensions due to cyclic dependencies.

BMC (Biere *et al.* 2003) 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. SAT-based BMC methods do not suffer from the state explosion problem as the methods based on BDDs. Heljanko and Niemelä (2003) exploit BMC in the verification of asynchronous systems modeled by 1-safe Petri nets. They provide a translation of a Petri net to a logic program which captures the execution of the net up to n steps and they develop a compact encoding of BMC of LTL formulas as the problem of finding stable models of logic programs. As a difference, the work in this paper aims at verifying properties of a temporal action theory including DLTL temporal constraints. Hence, we provide a translation of the action theory into ASP and we extend the encoding of BMC in Heljanko and Niemelä (2003) to deal with DLTL formulas.

Our encoding of BMC of LTL formulas in ASP does not make use of the Büchi automaton construction to build the path satisfying a formula. As future work, we aim at exploring an alternative approach which exploits the Büchi automaton of the formula to achieve completeness of BMC.

Stemming from the seminal paper of Gelfond and Lifschitz on the action language $\mathcal{A}$ (Gelfond and Lifschitz 1993), a lot of work has been devoted to define logic-based action languages. In particular, ASP has been shown to be well suited

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

for reasoning about dynamic domains (Gelfond 2007). Baral and Gelfond (2000) provide an encoding in ASP of the action specification language $\mathcal{AL}$, which extends the action description language $\mathcal{A}$ by allowing static and dynamic causal laws, executability conditions and concurrent actions. The proposed approach has been used for planning (Tu *et al.* 2011) and diagnosis (Balduccini and Gelfond 2003).

The action language defined in this paper can be regarded as a temporal extension of the language $\mathcal{A}$ (Gelfond and Lifschitz 1993) which allows for general temporal constraints, complex actions, and infinite computations, but does not deal with concurrent actions nor with incomplete knowledge. As regards laws in Π, our temporal action language has strong relations with the action languages $\mathcal{K}$ and $\mathcal{C}$.

The logic-based planning language, $\mathcal{K}$ (Eiter *et al.* 2003, 2004) is well suited for planning under incomplete knowledge and allows for concurrent actions. The main construct of $\mathcal{K}$ are causation rules of the form: `caused f if B after A`, meaning "If `B` is known to be true in the current state and `A` is known to be true in the previous state, then `f` is known to be true in the current state". Default negation can be used in the body of the rules and `A` may contain action atoms. The semantics of planning domains is defined in terms of states and transitions.

The temporal action language introduced in Section 3 for defining the component Π of the domain description can be regarded, apart from minor differences, as a fragment of $\mathcal{K}$ in which concurrent actions are not allowed. In particular, action laws (1) and dynamic causal laws (3):

$$\Box([a]l_0 \leftarrow (not)l_1, \ldots, (not)l_m, (not)[a]l_{m+1}, \ldots, (not)[a]l_k)$$
$$\Box(\bigcirc l_0 \leftarrow (not)l_1, \ldots, (not)l_m, (not) \bigcirc l_{m+1}, \ldots, (not) \bigcirc l_k)$$

can be mapped to the causation rules:

$$\textbf{caused } l_0 \textbf{ if } (not)l_{m+1} \ldots, (not)l_k \textbf{ after } a, (not)l_1, \ldots, (not)\ l_m$$
$$\textbf{caused } l_0 \textbf{ if } (not)l_{m+1} \ldots, (not)l_k \textbf{ after } (not)l_1, \ldots, (not)\ l_m$$

with the proviso, for dynamic causal laws, that $m \geqslant 1$. In case the literals $(not)l_1, \ldots, (not)l_m$ are not present (and the **after** part of the causation rule is empty), our dynamic causal law does not produce any effect on the initial state (which is not the next state of any other state) while the causation rule does. For this reason, our static causal laws can then be mapped to causation rules with empty **after** part. A similar translation can be given to precondition laws, which are special kinds of action laws and to initial state laws, which can be mapped to initial state causation rules in $\mathcal{K}$. All actions are regarded as being always executable, i.e., **executable** `a`, holds for all actions `a`. The correctness of this mapping emerges form the ASP encoding of our temporal language, which is similar, apart from minor differences, to the translation of $\mathcal{K}$ to answer set programming (Eiter *et al.* 2003).

The system $DLV^{\mathcal{K}}$ provides an implementation of $\mathcal{K}$ on top of the disjunctive logic programming system DLV. $DLV^{\mathcal{K}}$ does not only solve optimistic planning problems but also secure planning problems under incomplete initial states (*conformant planning*). $DLV^{\mathcal{K}}$ does not appear to support other kinds of reasoning besides planning, and, in particular, does not allow to express or verify temporal properties nor to reason about infinite computations.

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

The languages $\mathcal{C}$ and $\mathcal{C}^+$ (Giunchiglia and Lifschitz 1998; Giunchiglia *et al.* 2004) also deal with actions with indirect and non-deterministic effects and with concurrent actions, and are based on non-monotonic causation rules syntactically similar to those of $\mathcal{K}$, where head and body of causation rules can be boolean combinations of atoms. Their semantics is based on a non-monotonic causal logic (Giunchiglia *et al.* 2004). Causation rules can be represented in this logic by indexing fluents and actions with an integer i ($i = 0, \ldots, n$), in such a way that models of the causal theory correspond to histories of length n. The semantics of causal logic requires states to be complete. Due to the differences between the underlying semantics, a mapping between our action language and the languages $\mathcal{C}$ and $\mathcal{C}^+$ appears not to be straightforward.

If a $\mathcal{C}^+$ causal theory is definite (the head of a rule is an atom), it is possible to reason about it by turning the theory into a set of propositional formulas by means of a completion process, and then invoke a satisfiability solver. In this way, it is possible to perform various kinds of reasoning such as prediction, postdiction, or planning. However, the language does not exploits standard temporal logic constructs to reason about actions.

In the context of planning, temporally extended goals allow the specification of properties that have to be achieved in the states along the execution of the plan. The need for state trajectory constraints has been advocated, for instance, in PDDL3 (Gerevini and Long 2005), the domain description language used in the 2006 International Planning Competition. Bacchus and Kabanza (2000) exploit a first order linear temporal logic for defining domain dependent search control knowledge in the planner TLPlan. Pistore and Traverso (2001) define a planning algorithm that generates plans for extended goals in a non-deterministic domain, where extended goals are CTL formulas. Son *et al.* (2006) show that temporal control knowledge can be incorporated in a planner written in ASP. It provides a translation of a planning problem whose domain is defined in the action language $\mathcal{B}$ into ASP as well as a translation of the temporal constraints on the domain. The work on temporally extended goals in (Dal Lago *et al.* 2002; Baral and Zhao 2007) is concerned with expressing preferences among goals and exceptions in goal specification. Son and Pontelli (2006) and Bienvenu *et al.* (2006) introduce languages including temporal operators for expressing preferences on solutions of planning problems. Son and Pontelli (2006), in particular, build on answer set planning, i.e., computing plans in ASP; the computation of *preferred* plans is also mapped to ASP, relying on an optimization predicate. As a difference with the proposals above, in this paper, we do not specifically focus on planning. Our language is intended to address several different reasoning tasks (including property verification) on rich domain descriptions, allowing for ramifications, non-deterministic and complex actions, incomplete initial states, and, in particular, it can be used for reasoning about infinite computations. However, in this paper, we do not address the problem of expressing preferences among goals.

As our language includes program expressions in the temporal formulas, it is related to the Golog language (Levesque *et al.* 1997), in which complex actions (plans) can be formalized as Algol-like programs. $\mathcal{ESG}$ (Claßen and Lakemeyer 2008)

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

is a second order extension of CTL* for reasoning about non-terminating Golog programs. In $\mathcal{ESG}$ programs include, besides regular expressions, non-deterministic choice of arguments and concurrent composition. The paper presents a method for verification of a first order CTL fragment of $\mathcal{ESG}$, using model checking and regression based reasoning. Because of first order quantification, this fragment is in general undecidable. DLTL (Henriksen and Thiagarajan 1999) can be regarded as a decidable LTL fragment of $\mathcal{ESG}$. Satisfiability in DLTL is known to be PSPACE-complete (Henriksen and Thiagarajan 1999), as for LTL.

Observe that, although our temporal answer sets are, in general, infinite, we do not need to exploit specific techniques for reasoning about infinite answer sets (Bonatti 2004), due to the property that an infinite path can be finitely represented as a k-loop.

Acknowledgments

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

References

Bacchus, F. and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence 22*, 5–27.

Bacchus, F. and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence 116,* 1–2, 123–191.

Balduccini, M. and Gelfond, M. 2003. Diagnostic reasoning with A-prolog. *Theory and Practice of Logic Programming 3,* 4–5, 425–461.

Baral, C. and Gelfond, M. 2000. Reasoning agents in dynamic domains. In *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, Boston, 257–279.

Baral, C. and Zhao, J. 2007. Non-monotonic temporal logics for goal specification. In *Proceedings of IJCAI 2007*. AAAI Press/International Joint Conferences on Artificial Intelligence, Menlo Park, USA, 236–242.

Bienvenu, M., Fritz, C. and McIlraith, S. 2006. Planning with qualitative temporal preferences. In *Proceedings of KR 2006*. AAAI Press, Menlo Park, USA, 134–144.

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

Bonatti, P. 2004. Reasoning with infinite stable models. *Artificial Intelligence 156,* 1, 75–111.

Claßen, J. and Lakemeyer, G. 2008. A logic for non-terminating Golog programs. In *Proceedings of KR 2008*. AAAI Press, Menlo Park, USA, 589–599.

Dal Lago, U., Pistore, M. and Traverso, P. 2002. Planning with a language for extended goals. In *Proceedings of AAAI 2002*. AAAI Press, Menlo Park, USA, 447–454.

D'Aprile, D., Giordano, L., Gliozzi, V., Martelli, A., Pozzato, G. and Theseider Dupré, D. 2010. Verifying business process compliance by reasoning about actions. In *Proceedings of CLIMA 2010*, LNCS 6245. Springer-Verlag, Berlin, 99–116.

Denecker, M., Theseider Dupré, D. and Van Belleghem, K. 1998. An inductive definitions approach to ramifications. *Electronic Transactions on Artificial Intelligence 2*, 25–97.

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

Eiter, T., Faber, W., Leone, N., Pfeifer, G. and Polleres, A. 2003. A logic programming approach to knowledge-state planning, II: The DLV^{K} system. *Artificial Intelligence 144,* 1–2, 157–211.

Eiter, T., Faber, W., Leone, N., Pfeifer, G. and Polleres, A. 2004. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Transactions on Computational Logic 5,* 2, 206–263.

Gelfond, M. 2007. *Handbook of Knowledge Representation, chapter 7, Answer Sets*. Elsevier, Amsterdam.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the 5th International Conference and Symposium*. MIT Press, Boston, USA, 1070–1080.

Gelfond, M. and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming 17*, 301–322.

Gerevini, A. and Long, D. 2005. *Plan Constraints and Preferences in PDDL3*. Technical Report, Department of Electronics and Automation, University of Brescia, Italy.

Giordano, L. and Martelli, A. 2006. Tableau-based automata construction for dynamic linear time temporal logic. *Annals of Mathematics and Artificial Intelligence 46,* 3, 289–315.

Giordano, L., Martelli, A. and Schwind, C. 2001. Reasoning about actions in dynamic linear time temporal logic. *The Logic Journal of the IGPL 9,* 2, 289–303.

Giordano, L., Martelli, A. and Schwind, C. 2007. Specifying and verifying interaction protocols in a temporal action logic. *Journal of Applied Logic 5*, 214–234.

Giunchiglia, E. 2000. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Proceedings of KR 2000*. Morgan Kaufman, San Francisco, USA, 657–666.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N. and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence 153,* 1–2, 49–104.

Giunchiglia, E. and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proceedings of AAAI/IAAI 1998*. AAAI Press, Menlo Park, USA, 623–630.

Giunchiglia, F. and Traverso, P. 1999. Planning as model checking. In *Proceedings of 5th European Conference on Planning (ECP'99)*. Springer-Verlag, Berlin, 1–20.

Heljanko, K. and Niemelä, I. 2003. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming 3,* 4–5, 519–550.

Henriksen, J. and Thiagarajan, P. 1999. Dynamic linear time temporal logic. *Annals of Pure and Applied logic 96,* 1-3, 187–207.

Kabanza, F., Barbeau, M. and St-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence 95*, 67–113.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

Levesque, H., Reiter, R., Lespérance, Y., Lin, F. and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming 31,* 1–3, 59–83.

Lifschitz, V. 1990. Frames in the space of situations. *Artificial Intelligence 46*, 365–376.

Panati, A. and Theseider Dupré, D. 2000. State-based vs simulation-based diagnosis of dynamic systems. In *Proceedings of ECAI 2000*. IOS Press, Amsterdam, 176–180.

Panati, A. and Theseider Dupré, D. 2001. Causal simulation and diagnosis of dynamic systems. In *Proceedings of AI*IA 2001: Advances in Artificial Intelligence*, LNCS 2175. Springer-Verlag, Berlin, 135–146.

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press

Pistore, M. and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *Proceedings of IJCAI 2001*. Morgan Kaufman, San Francisco, USA, 479–486.

Pistore, M., Traverso, P. and Bertoli, P. 2005. Automated composition of web services by planning in asynchronous domains. In *Proceedings of ICAPS 2005*. AAAI Press, Menlo Park, USA, 2–11.

Son, T., Baral, C., Tran, N. and McIlraith, S. 2006. Domain-dependent knowledge in answer set planning. *ACM Transactions on Computational Logic 7,* 4, 613–657.

Son, T. C. and Pontelli, E. 2006. Planning with preferences using logic programming. *Theory and Practice of Logic Programming 6,* 5, 559–607.

Tu, P., Son, T., Gelfond, M. and Morales, R. 2011. Approximation of action theories and its application to conformant planning. *Artificial Intelligence 175,* 1, 79–119.

https://doi.org/10.1017/S1471068411000639 Published online by Cambridge University Press