Achieving completeness in the verification of action theories by Bounded Model Checking in ASP

L. Giordano, A. Martelli and D. Theseider Dupre'

(preliminary version)

published in

Journal of Logic and Computation 25(6), pp 1307-1330 (2015)

http://logcom.oxfordjournals.org/content/25/6/1307

Achieving completeness in the verification of action theories by Bounded Model Checking in ASP

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

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

Abstract. Temporal logics are well suited for reasoning about actions, as they allow for the specification of domain descriptions including temporal constraints as well as for the verification of temporal properties. The paper deals with verification of action theories defined in a temporal extension of *answer set programming* which combines ASP with a dynamic linear time temporal logic (DLTL). The paper proposes an approach to bounded model checking which exploits the Büchi automaton construction while searching for a counterexample, with the aim of achieving completeness. The paper provides an encoding in ASP of the temporal action domains and of Bounded Model Checking of DLTL formulas. The paper also deals with reasoning about epistemic knowledge and incomplete states.

1 Introduction

Temporal logics have been extensively used in the specification and verification of action domains in many fields, from planning to web services. In planning, both CTL [29, 34] and LTL [6,3] have been used in the specification of temporally extended goals. The need for state trajectory constraints has been advocated in PDDL3 [20]. [2] exploits a first order linear temporal logic for defining domain dependent search control knowledge in the planner TLPlan, and in [13] strong fairness constraints expressed in LTL are used to restrict nondeterminism in generalized planning. LTL has been used in the verification of agent interaction protocols [24] and for enforcing regulations in automated Web service composition [35]. In the context of reasoning about action, [11] introduced a second order extension of the temporal logic CTL*, $\mathcal{ESG}$, to reason about non-terminating Golog programs.

In this paper, we start from the temporal action theories introduced in [26], formulated in a temporal extension of *answer set programming* (ASP [18]) based on Dynamic Linear Time Temporal Logic (DLTL [31]) and we exploit Bounded Model Checking (BMC) techniques for the verification of properties of such action theories. BMC [7] is an efficient model checking technique which does not require a tableau or automaton construction. Given a system model (a transition system) and a property to be checked, it searches for a counterexample of the property as a path of length k, generating a propositional formula that is satisfiable iff such a counterexample exists. The bound k on the length of the path is iteratively increased and, if no counterexample exists, the procedure never stops, i.e., it is a partial decision procedure for checking validity. Techniques for achieving completeness have been described e.g. in [7], where upper bounds

for k are determined for some classes of properties, namely unnested properties. To deal with completeness, [10] proposes a *semantic* translation scheme, based on Büchi automata.

In [30] Helianko and Niemelä developed a compact encoding of bounded model checking of LTL formulas as the problem of finding stable models of logic programs. Since ASP naturally accommodates for reasoning about actions, in [26] this encoding is extended to DLTL formulas, for reasoning about theories including complex actions and programs. These papers do not address the problem of achieving completeness.

In this paper we propose an alternative encoding of BMC of DLTL formulas in ASP, with the aim of achieving completeness. Unlike [30, 26], the search for a counterexample exploits the Büchi automaton construction [21] as well as the transition system. Unlike [10], a "counterexample" path is searched for, without assuming that the Büchi automaton is constructed in advance. Our counterexample is an accepting path of the product Büchi automaton which can be finitely represented as a *(k,l)-loop*, i.e., a finite path of length k, in which the states are all distinct from each other, and terminating in a loop back to a previous state l. The procedure for verifying a given property searches for a *(k,l)-loop* providing a counterexample to the property, increasing k until either a counterexample is found, or no path of length k of distinct states can be found.

As in [26], verification is performed on a transition system provided by a domain description in a temporal action theory, and our BMC approach is used for proving properties of domain descriptions. The action theory is given in a temporal extension of ASP, based on the generalization of the notion of *answer set* [18] to *temporal answer sets*. The temporal properties of a domain description can be proved by combining the construction of temporal extensions of the domain with the verification of their properties, according to a tableaux-based procedure which provides an encoding of BMC in ASP. In particular, an incremental encoding in iClingo [17] is provided. The proposed approach provides a decision procedure for the verification of satisfiability and validity properties of an action domain in a temporal action theory. The paper also addresses the problem of verification of domain descriptions with incomplete knowledge introducing epistemic modalities.

2 Dynamic Linear Time Temporal Logic

In this paper we refer to a formulation of DLTL (dynamic linear time temporal logic), in [31], where the until operator $\mathcal{U}^\pi$ is indexed by a program π which, as in PDL, can be any regular expression built from atomic actions using sequence (;), nondeterministic choice ($+$) and finite iteration ($*$).

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω, by τ, τ' the words over Σ^* and by ε the empty word. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^*, namely, $\tau \leq \tau'$ iff $\exists \tau''$ such that $\tau\tau'' = \tau'$, and $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$. For $u \in \Sigma^\infty$, we denote by *prf(u)* the set of finite prefixes of u.

Let the set of programs (regular expressions) generated by Σ be $Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$, where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite

words can be associated with each program by the mapping $[[]] : Prg(\Sigma) \rightarrow 2^{\Sigma^*}$ in the usual way. Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions. The set of formulas of DLTL(Σ) is defined as: DLTL(Σ) ::= $p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \mathcal{U}^{\pi} \beta$, where $p \in \mathcal{P}$, $\pi \in Prg(\Sigma)$ and α, β range over DLTL(Σ).

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^{\omega}$ and $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ 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 usual for boolean formulas and as follows for atomic and until formulas:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \alpha \mathcal{U}^{\pi} \beta$ 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 $\varepsilon \leq \tau'' < \tau'$, $M, \tau\tau'' \models \alpha$.

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 symbols $\top$ and $\bot$ can be defined as: $\top \equiv p \vee \neg p$ and $\bot \equiv \neg\top$. The derived modalities $\langle\pi\rangle\alpha$, $[\pi]\alpha$, $\bigcirc$ (next), $\mathcal{U}$, $\Diamond$ and $\Box$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^{\pi}\alpha$, $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$, $\bigcirc\alpha \equiv \bigvee_{a \in \Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where α is a formula and, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$.

3 Temporal action language

Let $\mathcal{L}$ be a first order language which includes a finite number of constants and variables, but no function symbol. Let $\mathcal{P}$ be the set of predicate symbols, Var the set of variables and C the set of constant symbols. We call *fluents* atomic literals of the form $p(t_1, \ldots, t_n)$, where, for each i, $t_i \in Var \cup C$. A *simple fluent literal* l is an atomic literal $p(t_1, \ldots, t_n)$ or its negation $\neg p(t_1, \ldots, t_n)$. We denote by Lit_S the set of all simple fluent literals, including $\bot$ and $\top$, the *false* and *true* literals. Lit_T is the set of *temporal fluent literals*: if $l \in Lit_S$, then $[a]l, \bigcirc l \in Lit_T$, where a is an *action name* (an atomic proposition, possibly containing variables), and $[a]$ and $\bigcirc$ are the temporal operators introduced in the previous section. Let $Lit = Lit_S \cup Lit_T$. Given a (simple or temporal) fluent literal l, $not\ l$ represents the default negation of l. A (simple or temporal) fluent literal, possibly preceded by default negation, will be called an *extended fluent literal*.

A *domain description* Π is a set of laws describing the effects of actions and their executability preconditions. The laws are formulated as rules of a temporally extended logic programming language. Rules have the form

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

where the l_i's are either simple fluent literals or temporal fluent literals, with the following constraints: (i) If l_0 is a simple fluent literal, the body cannot contain temporal literals (in this case, the rule is called *static*); (ii) If $l_0 = [a]l$, the temporal literals in the body must have the form $[a]l'$; (iii) If $l_0 = \bigcirc l$, the temporal literals in the body must have the form $\bigcirc l'$. With these restrictions, possible successors of a state w only depend on w (see Proposition 2 at the end of the Section 3.1). As usual in ASP, rules with variables are a shorthand for the set of their ground instances.

A *state*, informally, is a set of ground simple fluent literals closed with respect to the rules above (see Section 3.1). A state is *consistent* if it is not the case that both f and $\neg f$ (for some ground fluent f) belong to it, or that $\bot$ belongs to it. A state is *complete* if, for each ground fluent $f \in \mathcal{P}$, either f or $\neg f$ belongs to it. The execution of an action in a state may change the values of fluents in the state through its direct and indirect effects, thus giving rise to a new state. We assume that a law as (1) can be applied in all states, while when prefixed with **Init**, it only applies to the initial state.

Example 1. This example describes a mail delivery system, which checks if there is mail in the mailbox of employees and delivers mail to them. The actions in Σ are: $sense$ (verifying if there is mail in any of the mailboxes), $deliver(E)$ (delivering the mail to employee E), $wait$. The fluents are $mail(E)$ (there is mail in the mailbox of E). Π contains the following immediate effects and persistence laws:

$[deliver(E)]\neg mail(E)$
$[sense]mail(E) \leftarrow \ not\ [sense]\neg mail(E)$
$\bigcirc mail(E) \leftarrow mail(E), not \bigcirc \neg mail(E)$
$\bigcirc \neg mail(E) \leftarrow \neg mail(E), not \bigcirc mail(E)$

Their meaning is that: after delivering the mail to E, there is no mail for E any more; the action $sense$ may (non-monotonically) cause $mail(E)$ to become true. The last two rules define the persistence of fluent $mail$.

Observe that the persistence laws interact with the immediate effect laws above. The execution of $sense$ in a state in which there is no mail for some E ($\neg mail(E)$), may either lead to a state in which $mail(E)$ holds (by the second action law) or to a state in which $\neg mail(E)$ holds (by persistence of $\neg mail(E)$), while $mail(E)$ always persists. Thus, $sense$ is a nondeterministic action. The following precondition laws:

$[deliver(E)] \bot \leftarrow \neg mail(E)$
$[wait] \bot \leftarrow mail(E)$

specify that, if there is no mail for E, $deliver(E)$ is not executable, and, if there is mail for E, $wait$ is not executable.

We assume that there are only two employees, a and b, and that in the initial state there is mail for a and not for b, i.e. Π includes $\mathbf{Init}\ mail(a)$ and $\mathbf{Init}\ \neg mail(b)$.

The language is also well suited to describe causal dependencies among fluents [26] by the definition of *static* and *dynamic causal laws* similar to the ones in the action languages $\mathcal{K}$ [15] and $\mathcal{C}^+$ [27].

3.1 Temporal Answer Sets

In this section, we recall the notion of *temporal answer set* in [26], which extends the notion of *answer set* [18], and we prove properties of the transition system associated with a domain description. To this purpose, we let Π be the ground instantiation of the domain description, and Σ the set of all the ground instances of the action names in Π.

A temporal interpretation is defined as a pair (σ, S), where $\sigma \in \Sigma^\omega$ is a sequence of actions and S is a consistent set of ground literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ and l is a ground simple fluent literal, meaning that l holds in the state obtained by executing $a_1 \ldots a_k$ in (σ, S). S is *consistent* iff it is not the case

that both $[a_1;\ldots;a_k]l \in S$ and $[a_1;\ldots;a_k]\neg l \in S$, for some l, or $[a_1;\ldots;a_k]\bot \in S$. A temporal interpretation (σ, S) is *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.

The notion of satisfiability of a rule in a temporal interpretation (σ, S), as well as the notion of *reduct $\Pi^{(\sigma,S)}$ of (a domain description) Π relative to* (σ, S) can be defined as natural extensions of Gelfond and Lifschitz's ones [18]. With these notions, a temporal answer set of Π is defined as a temporal interpretation (σ, S) such that S is minimal (in the sense of set inclusion) among the S' such that (σ, S') is a partial interpretation satisfying the rules in the reduct $\Pi^{(\sigma,S)}$.

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

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

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

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

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

The satisfiability of rule bodies in a temporal interpretation is defined as usual. A rule $H \leftarrow Body$ is satisfied in a temporal interpretation (σ, S) if, for all action sequences $a_1 \ldots a_k$ (including the empty action sequence ε), $(\sigma, S), a_1 \ldots a_k \models Body$ implies $(\sigma, S), a_1 \ldots a_k \models H$. A rule $\mathbf{Init}\ H \leftarrow Body$ is satisfied in a partial temporal interpretation (σ, S) if, $(\sigma, S), \varepsilon \models Body$ implies $(\sigma, S), \varepsilon \models H$. A rule $[a_1;\ldots;a_h](H \leftarrow Body)$ (that we will introduce below for defining the reduct of Π) is satisfied in a temporal interpretation (σ, S) if, $(\sigma, S), a_1 \ldots a_k \models Body$ implies $(\sigma, S), a_1 \ldots a_k \models H$.

Definition 1. *[26] Let Π be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$. A temporal interpretation (σ, S) is a* temporal answer set of Π *if S is minimal (in the sense of set inclusion) among the S' such that (σ, S') is a partial interpretation satisfying the rules in Π.*

To define temporal answer sets of a program Π containing negation, given a temporal interpretation (σ, S) over $\sigma \in \Sigma^\omega$, we define the *reduct, $\Pi^{(\sigma,S)}$, of Π relative to* (σ, S) extending Gelfond and Lifschitz' transform [19] to compute a different reduct of Π for each prefix $a_1, \ldots, a_h$ of σ. Observe that $\Pi^{(\sigma,S)}$ is an infinite set of rules, but each reduct of Πrelative to a prefix $a_1, \ldots, a_h$ is finite.

Definition 2. *[26] The* reduct, $\Pi^{(\sigma,S)}_{a_1,\ldots,a_h}$, of Π relative to (σ, S) and to the prefix $a_1, \ldots, a_h$ of σ *, is the set of all the rules $[a_1;\ldots;a_h](H \leftarrow l_1, \ldots, l_m)$ such that $H \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ is in Π and $(\sigma, S), a_1, \ldots, a_h \not\models l_i$, for all $i = m+1, .., n$. $\Pi^{(\sigma,S)}_{\varepsilon}$ is defined similarly, but $[\varepsilon]$ in front of the rules is omitted. The* reduct $\Pi^{(\sigma,S)}$ of Π relative to (σ, S) *is the union of all $\Pi^{(\sigma,S)}_{\tau}$ for all prefixes τ of σ.*

Definition 3. *[26] A temporal interpretation (σ, S) is a* temporal answer set of Π *if (σ, S) is a temporal answer set of the reduct $\Pi^{(\sigma,S)}$.*

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

initial state is not complete, we consider all the possible ways to complete the initial state by introducing in Π, for each fluent name f, the rules: $\mathbf{Init}\ f \leftarrow not\ \neg f$ and $\mathbf{Init}\ \neg f \leftarrow not\ f$.

The case of total temporal answer sets is of special interest, as a total temporal answer set (σ, S) can be regarded as temporal model (σ, V_S), where, for each finite prefix $a_1 \dots a_k$ of σ, $V_S(a_1, \dots, a_k) = \{p : [a_1, \dots, a_k]p \in S\}$.

A total temporal interpretation (σ, S) provides, for each prefix $a_1 \dots a_k$, a complete state corresponding to that prefix. We denote by $w^{(\sigma,S)}_{a_1 \dots a_k}$ the state obtained by the execution of the actions $a_1 \dots a_k$ in the sequence, namely $w^{(\sigma,S)}_{a_1 \dots a_k} = \{l : [a_1; \dots; a_k]l \in S\}$.

Given a domain description Π over Σ with total answer sets, a *transition system* (W, I, T) can be associated with Π as follows: (a1) W is the set of all the possible consistent and complete states of the domain description; (a2) I is the set of all the states in W satisfying the static initial state laws in Π; (a3) $T \subseteq W \times \Sigma \times W$ 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: $w = w^{(\sigma,S)}_{[a_1;\dots;a_h]}$ and $w' = w^{(\sigma,S)}_{[a_1;\dots;a_h;a]}$, for some h. We can show that:

Proposition 1. *Each infinite path of the transition system $TS = (W, I, T)$ associated with a domain description Π, which starts from a state in I, corresponds to a temporal answer set of Π; and vice-versa.*

Proof. We prove the first part. Let $w_0, w_1, \dots$ be an infinite path in the transition system TS such that $w_0 \in I$ and $(w_{h-1}, a_h, w_h) \in T$, for all $h = 1, 2, \dots$. We define a total temporal interpretation (σ, S) of Π such that: $\sigma = a_1 a_2 \dots$ is the sequence of actions occurring in the path and S is defined as follows:

$$[a_1; \dots; a_h]l \in S \text{ if and only if } l \in w_h$$

where w_h is the h-th state in the path. We show that (σ, S) is a temporal answer set of Π by showing that S is minimal among those R such that the interpretation (σ, R) satisfies the rules in the reduct $\Pi^{(\sigma,S)}$.

Let us first prove that (σ, S) satisfies all rules $[a_1; \dots; a_h](H \leftarrow l_1, \dots, l_m)$ in $\Pi^{(\sigma,S)}$. Consider the transition (w_h, a, w_{h+1}) in the path (where $a = a_{h+1}$). By definition of TS there must be an answer set (σ', S') of Π such that $w_h = w^{(\sigma',S')}_{[b_1;\dots;b_k]}$ and $w_{h+1} = w^{(\sigma',S')}_{[b_1;\dots;b_k;a]}$, for some prefix $b_1, \dots, b_k$ of σ'. Hence, for all simple literals l:

$$l \in w_h \text{ iff } [b_1; \dots; b_k]l \in S'$$
$$l \in w_{h+1} \text{ iff } [b_1; \dots; b_k; a]l \in S'$$

Therefore, from the definition of S: (a) $[a_1; \dots; a_h]l \in S$ iff $[b_1; \dots; b_k]l \in S'$; (b) $[a_1; \dots; a_h; a]l \in S$ iff $[b_1; \dots; b_k; a]l \in S'$. From these, we can prove that, for all simple and temporal literals l: (c) $(\sigma, S), a_1, \dots, a_h \models l$ iff $(\sigma', S'), b_1, \dots, b_k \models l$. For simple literals, (c) is an immediate consequence of (a), while for temporal literals, it is a consequence of (b). As a consequence of (c) we have that:

$$[a_1; \dots; a_h](H \leftarrow l_1, \dots, l_m) \in \Pi^{(\sigma,S)} \text{ iff } [b_1; \dots; b_k](H \leftarrow l_1, \dots, l_m) \in \Pi^{(\sigma',S')}$$

To show that (σ, S) satisfies $[a_1; \dots; a_h](H \leftarrow l_1, \dots, l_m)$, assume that $(\sigma, S), a_1, \dots, a_h \models l_i$, for all $i = 1, \dots, m$. By (c), $(\sigma', S'), b_1, \dots, b_k \models l_i$, for all $i = 1, \dots, m$.

Moreover, $[b_1;\ldots;b_k](H \leftarrow l_1,\ldots,l_m)$ is in $\Pi^{(\sigma',S')}$. As (σ',S') is an answer set of Π, then (σ',S') satisfies $\Pi^{(\sigma',S')}$ and, hence, $(\sigma',S'), b_1,\ldots,b_k \models H$. Again by (c), $(\sigma,S), a_1,\ldots,a_h \models H$. Therefore, (σ,S) satisfies $[a_1;\ldots;a_h](H \leftarrow l_1,\ldots,l_m)$.

We still need to prove that S is minimal among those R such that (σ,R) satisfies $\Pi^{(\sigma,S)}$. Suppose, by absurdum, it is not and there is an R such that (σ,R) satisfies $\Pi^{(\sigma,S)}$ and $R \subset S$. We want to show that this leads to a contradiction.

As $R \subset S$, there must be a smallest h such that $[a_1;\ldots;a_{h-1};a]l \in S$ and $[a_1;\ldots;a_{h-1};a]l \notin R$. Let us consider the states $w_{h-1} = w^{(\sigma,S)}_{[a_1;\ldots;a_{h-1}]}$ and $w_h = w^{(\sigma,S)}_{[a_1;\ldots;a_{h-1};a]}$ on the path and the transition $(w_{h-1},a,w_h) \in T$ (for $a = a_h$). By construction of TS, there must be an answer set (σ',S') of Π going through this transition, i.e. a (σ',S') such that, for some prefix $b_1,\ldots,b_k$ of σ', $w_{h-1} = w^{(\sigma',S')}_{[b_1;\ldots;b_{k-1}]}$ and $w_h = w^{(\sigma',S')}_{[b_1;\ldots;b_{k-1};a]}$. It is easy to see that, for all simple and temporal literals l:

$$(\sigma,S), a_1,\ldots,a_{h-1} \models l \text{ iff } (\sigma',S'), b_1,\ldots,b_{k-1} \models l$$

and hence, that:

$$[a_1;..;a_{h-1}](h \leftarrow l_1,..,l_m) \in \Pi^{(\sigma,S)} \text{ iff } [b_1;..;b_{k-1}](h \leftarrow l_1,..,l_m) \in \Pi^{(\sigma',S')} \quad (2)$$

$$[a_1;..;a_{h-1};a](l_0 \leftarrow l_1,..,l_m) \in \Pi^{(\sigma,S)} \text{ iff } [b_1;..;b_{k-1};a](l_0 \leftarrow l_1,..,l_m) \in \Pi^{(\sigma',S')} \quad (3)$$

where l_0 is a simple literal (as well as all the literals in (3)).

We show that, starting from (σ,R), we can build an interpretation (σ',R') such that (σ',R') satisfies the rules in $\Pi^{(\sigma',S')}$ and $R' \subset S'$, thus contradicting the minimality of the answer set (σ',S') of Π. We define the interpretation R' as follows:

$$[b_1,\ldots,b_{k-1};a]l \in R' \text{ iff } [a_1,\ldots,a_{h-1};a]l \in R \quad (4)$$

$$[b_1,\ldots,b_j]l \in R' \text{ iff } [b_1,\ldots,b_j]l \in S', \text{for all prefixes } b_1\ldots b_j \text{ of } \sigma' \text{ with } j \neq k \quad (5)$$

(observe that $b_k = a$). It is clear that $R' \subset S'$. The proof that the interpretation (σ',R') satisfies the rules $[b_1;\ldots;b_j](H \leftarrow l_1,\ldots,l_m)$ in $\Pi^{(\sigma',S')}$ can be done by cases, by considering the different cases for j ($j < k-1$, $j = k-1$, $j = k$ and $j > k$) and for H (simple or temporal literal).

Let us consider the case $j = k$. Consider the rule $[b_1;\ldots;b_{k-1};a](l_0 \leftarrow l_1,\ldots,l_m)$ in $\Pi^{(\sigma',S')}$ with l_0 simple literal. If $(\sigma',R'), b_1,\ldots,b_{k-1},a \models l_j$ (for $j = 1,\ldots,m$, the l_j's are also simple), then $[b_1;\ldots;b_{k-1};a]l_j \in R'$ and, by (4), we have $[a_1;\ldots;a_{h-1};a]l_j \in R$. We have assumed that (σ,R) satisfies all the rules in $\Pi^{(\sigma,S)}$, an in particular, by (3), it satisfies $[a_1;\ldots;a_{h-1};a](l_0 \leftarrow l_1,\ldots,l_m)$. Hence, $[a_1;\ldots;a_{h-1};a]l_0 \in R$ and, by (4), $[b_1;\ldots;b_{k-1};a]l_0 \in R'$. Therefore, (σ',R') satisfies the above rule in $\Pi^{(\sigma',S')}$.

Consider the rule $[b_1;\ldots;b_{k-1};a](H \leftarrow l_1,\ldots,l_m)$ in $\Pi^{(\sigma',S')}$, with $H = \bigcirc l$. Suppose $(\sigma',R'), b_1,\ldots,b_{k-1},a \models l_j$ (for $j = 1,\ldots,m$). We can show that (σ',S'), $b_1,\ldots,b_{k-1},a \models l_j$ (for $j = 1,\ldots,m$). By condition (iii) on the rules, l_j can either be a simple fluent literal or a temporal literal of the form $\bigcirc l'$. Consider the case l_j is a simple literal. Then $[b_1;\ldots;b_{k-1};a]l_j \in R'$ and, by (4), $[a_1;\ldots;a_{h-1};a]l_j \in R$. By definition of R, $R \subset S$, hence $[a_1;\ldots;a_{h-1};a]l_j \in S$ and, as by construction $w_h = w^{(\sigma',S')}_{[b_1;\ldots;b_{k-1};a]}$, $[b_1;\ldots;b_{k-1};a]l_j \in S'$. In the case $l_j = \bigcirc l'$, then $(\sigma',R'), b_1,\ldots,b_{k-1}$, $a, b_{k+1} \models l'$, i.e. $[b_1;\ldots;b_{k-1};a;b_{k+1}]l' \in R'$ and, by (5), $[b_1;\ldots;b_{k-1};a;b_{k+1}]l' \in S'$. Thus, $(\sigma',S'), b_1,\ldots,b_{k-1},a \models \bigcirc l'$.

As $(\sigma', S'), b_1, \ldots, b_{k-1}, a \models l_j$ (for $j = 1, \ldots, m$) and (σ', S') satisfies the rules in $\Pi^{(\sigma', S')}$, we have $(\sigma', S'), b_1, \ldots, b_{k-1}, a \models H$. As $H = \bigcirc l$, $[b_1; \ldots; b_{k-1}; a; b_{k+1}]l \in S'$ and, by (5), $[b_1; \ldots; b_{k-1}; a; b_{k+1}]l \in R'$. Hence, $(\sigma', R'), b_1, \ldots, b_{k-1}, a \models H$, and (σ', R') satisfies the rule. We proceed similarly in case $H = [b]l$.

Let us consider the case $j = k-1$. Consider the rule $[b_1; \ldots; b_{k-1}](H \leftarrow l_1, \ldots, l_m)$ in $\Pi^{(\sigma', S')}$, with $H = [a]l$. Let $(\sigma', R'), b_1, \ldots, b_{k-1} \models l_j$ (for $j = 1, \ldots, m$). By (2), we have that $[a_1; \ldots; a_{h-1}](H \leftarrow l_1, \ldots, l_m)$ is in $\Pi^{(\sigma, S)}$. We can show (see below) that $(\sigma, R), a_1, \ldots, a_{h-1} \models l_j$ (for $j = 1, \ldots, m$). Given that (σ, R) satisfies all the rules in $\Pi^{(\sigma, S)}$, it follows that $(\sigma, R), a_1, \ldots, a_{h-1} \models [a]l$, and $[a_1; \ldots; a_{h-1}; a]l \in R$. Thus, by (4), $[b_1; \ldots; b_{k-1}; a]l \in R'$, and therefore $(\sigma', R'), b_1, \ldots, b_{k-1} \models [a]l$.

Let us show that $(\sigma, R), a_1, \ldots, a_{h-1} \models l_j$ (for $j = 1, \ldots, m$). If l_j is simple, then by definition of R' (5), $(\sigma', S'), b_1, \ldots, b_{k-1} \models l_j$ (for $j = 1, \ldots, m$) and, also, $(\sigma, S), a_1, \ldots, a_{h-1} \models l_j$ (for $j = 1, \ldots, m$). As w_h is the first state on which R differs form S, $(\sigma, R), a_1, \ldots, a_{h-1} \models l_j$ (for $j = 1, \ldots, m$). If $l_j = [a]l$, from $(\sigma', R'), b_1, \ldots, b_{k-1} \models [a]l$ we get $(\sigma', R'), b_1, \ldots, b_{k-1}, a \models l$ and, by (4): (σ, R), $a_1, \ldots, a_{h-1}, a \models l$ and $(\sigma, R), a_1, \ldots, a_{h-1} \models [a]l$. We proceed similarly in case $H = \bigcirc l$.

The case for $j = k - 1$ with H simple literal, and the cases for $j < k - 1$ and $j > k$ are straightforward.

The fact that (σ', R') satisfies the rules in $\Pi^{(\sigma', S')}$ contradicts the minimality of S' and the assumption that (σ', S') is an answer set. Hence assuming that S is not minimal among the R such that (σ, R) satisfies $\Pi^{(\sigma, S)}$ leads to a contradiction. This concludes the first part of the proof.

The second part of the proof (the vice-versa) follows trivially from the definition of transition system associated with Π. □

The following proposition can be proved in a similar way:

Proposition 2. *The next states of a state w in a transition system TS only depend on w and not on previous states.*

In the following section, we make use of a next state function $nextTSstate$ that, given a state w and an action a, determines all the states reachable in the transition system from w by a (if any). Observe, that restrictions (i) - (iii) on rules (1) , are essential to ensure that possible successors of a state w only depend on w. For instance, with rules of the form $l_2 \leftarrow \bigcirc \bigcirc l_1$ or $\bigcirc \bigcirc l_2 \leftarrow l_1$ Proposition 2 would not hold.

3.2 Verification of Enriched Domain Descriptions

As a total temporal answer set of a domain description can be interpreted as a DLTL model, it is easy to combine domain descriptions with DLTL formulas. This can be done by adding to the domain description Π a set of DLTL formulas $\mathcal{C}$ used as constraints on the executions of the domain description. We denote by $(\Pi, \mathcal{C})$ the enriched domain description, and we define the *extensions of* $(\Pi, \mathcal{C})$ to be the temporal answer sets (σ, S) of Π satisfying the constraints $\mathcal{C}$. For example, taking *begin* as a new action name,

$\langle begin \rangle \top$

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

impose that the agent continuously executes a loop where it senses mail and delivers it. DLTL formulas can be used to encode properties to be verified on the enriched domain description; for example, $\Box(mail(a) \supset \Diamond\neg mail(a))$, i.e., if there is mail for a, the agent will eventually deliver it. This does not hold: a run is possible where there is always mail for both a and b, but the mail is repeatedly delivered to b and never to a.

Given an enriched domain description $(\Pi, \mathcal{C})$, some problems, e.g. planning, can be formulated as *satisfiability* of a formula φ, and others, such as the one in the example above, as validity of a formula φ. Usually, the validity of a property φ formulated as a DLTL formula is reduced to the *unsatisfiability* of $\neg\varphi$. In this case, if a model satisfying $\neg\varphi$ is found, it represents a counterexample to the validity of φ.

4 Bounded Model Checking with Büchi Automata

Satisfiability and validity problems can be solved by *model checking*. The standard approach to model checking for LTL is based on Büchi automata. A *Büchi automaton* over an alphabet Σ is a tuple $\mathcal{B} = (Q, \rightarrow, Q_{in}, F)$ where:

- Q is a finite nonempty set of states;
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation;
- $Q_{in} \subseteq Q$ is the set of initial states;
- $F \subseteq Q$ is a set of accepting states.

Let $\sigma \in \Sigma^\omega$. Then a *run* of $\mathcal{B}$ over σ is a map $\rho : prf(\sigma) \rightarrow Q$ such that $\rho(\varepsilon) \in Q_{in}$ and $\rho(\tau) \xrightarrow{a} \rho(\tau a)$ for each $\tau a \in prf(\sigma)$ with $a \in \Sigma$. The run ρ is *accepting* iff $inf(\rho) \cap F \neq \emptyset$, where $inf(\rho) \subseteq Q$ is given by: $q \in inf(\rho)$ iff $\rho(\tau) = q$ for infinitely many $\tau \in prf(\sigma)$. Finally $\mathcal{L}(\mathcal{B})$, the language of ω-words accepted by $\mathcal{B}$, is: $\mathcal{L}(\mathcal{B}) = \{\sigma | \exists$ an accepting run of $\mathcal{B}$ over $\sigma\}$.

The satisfiability problem for a LTL formula α can be solved by constructing a Büchi automaton $\mathcal{B}_\alpha$ such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if α is satisfiable [21]. Given a system modeled by a transition system TS, corresponding to a Büchi automaton $\mathcal{B}_{TS}$, *model checking* verifies that α holds for the system, building the *product automaton* $\mathcal{B}_{TS} \times \mathcal{B}_{\neg\alpha}$ and checking for emptiness of the accepted language.

Biere et al. [7] showed that model checking is sometimes more efficient if, instead of building the product automaton, a path of the transition system satisfying $\neg\alpha$ is searched for. This technique is called *bounded model checking* (BMC), since it looks for infinite paths which can be represented as a finite path of length k with a back loop from state k to a previous state l in the path (a *(k,l)-loop*); the search proceeds iteratively, increasing k until a model satisfying α (a counterexample) is found — if one exists.

A BMC problem can be efficiently reduced to a propositional satisfiability problem or to an ASP problem [30]. If no model exists and the transition system contains a loop, the iterative procedure in general does not stop, i.e., it is a partial decision procedure for validity. Techniques for achieving completeness are described e.g. in [7] for some kinds of LTL formulas.

In this paper, we propose an approach to model checking which combines the advantages of BMC, in particular the possibility of formulating it easily and efficiently as an ASP problem, with the advantages of reasoning on the product Büchi automaton described above, mainly its completeness.

In the following we show how to adapt the procedure for building a Büchi automaton corresponding to a given DLTL formula [23] to the "on-the-fly" construction of the *product* Büchi automaton, and we show how this construction can be used to build a *(k,l)-loop* corresponding to a run of this automaton.

In this construction we assume that, as in [23], $until$ formulas are indexed with (non deterministic) finite automata rather than regular expressions, i.e., we have $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ instead of $\alpha \mathcal{U}^{\pi} \beta$, where $\mathcal{L}(\mathcal{A}(q)) = [[\pi]]$. We denote with $\mathcal{A}(q)$ a finite automaton $\mathcal{A}$ with initial state q and transition relation δ. For instance, we introduce an automaton $\mathcal{A}(q_0)$ equivalent to the regular program $sense; (deliver(a) + deliver(b) + wait); begin$, as follows: $\mathcal{A}$ has states $\{q_0, q_1, q_2, q_3\}$, initial state q_0, final state q_3 and transition function $\{q_1\} = \delta(q_0, sense)$, $\{q_2\} = \delta(q_0, deliver(a)) = \delta(q_0, deliver(b)) = \delta(q_0, wait)$, $\{q_3\} = \delta(q_2, begin)$.

The following equivalences hold for the until operator [31]:

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

The construction of the nodes makes use of tableau rules which handle DLTL *signed formulas*, i.e. formulas prefixed with the symbol **T** or **F**. These rules are used for *expanding* a set of formulas[1] with the following notation and meaning:

- $\phi \Rightarrow \psi_1, \psi_2$, if ϕ belongs to the set of formulas, then add ψ_1 and ψ_2 to the set;
- $\phi \Rightarrow \psi_1 | \psi_2$, if ϕ belongs to the set of formulas, then make two copies of the set and add ψ_1 to one of them and ψ_2 to the other one.

The rules are the following:

Tor:	$\mathbf{T}(\alpha \vee \beta) \Rightarrow \mathbf{T}\alpha \mid \mathbf{T}\beta$
For:	$\mathbf{F}(\alpha \vee \beta) \Rightarrow \mathbf{F}\alpha, \mathbf{F}\beta$
Tneg:	$\mathbf{T}\neg\alpha \Rightarrow \mathbf{F}\alpha$
Fneg:	$\mathbf{F}\neg\alpha \Rightarrow \mathbf{T}\alpha$
TuntilFS:	$\mathbf{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \Rightarrow \mathbf{T}(\beta \vee (\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta))$ (q final state)
TuntilNFS:	$\mathbf{T}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \Rightarrow \mathbf{T}(\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta)$ (q non-final state)
FuntilFS:	$\mathbf{F}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \Rightarrow \mathbf{F}(\beta \vee (\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta))$ (q final state)
FuntilNFS:	$\mathbf{F}\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \Rightarrow \mathbf{F}(\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta)$ (q non-final state)

We use a function *tableau* which takes as input a set of formulas s, adds to it $\mathbf{T} \bigvee_{a \in \Sigma} \langle a \rangle \top$, and returns a (possibly empty) set of sets of formulas, obtained by repeatedly applying the above rules (by possibly creating new sets) until all non-elementary formulas in all sets have been expanded. We call *elementary formulas* the formulas of the form $\mathbf{T}\phi$ or $\mathbf{F}\phi$ where ϕ is either $\top$, or $\bot$, or a proposition or $\langle a \rangle \alpha$. Formula $\mathbf{T} \bigvee_{a \in \Sigma} \langle a \rangle \top$ makes explicit that in DLTL each state must be followed by a next state.

[1] In this section "formula" means "signed DLTL formula".

If the expansion of a set of formulas produces an inconsistent set, then this set is deleted. A set of formulas s is *inconsistent* in the following cases: (i) $\mathbf{T}\bot \in s$; (ii) $\mathbf{F}\top \in s$; (iii) $\mathbf{T}\alpha \in s$ and $\mathbf{F}\alpha \in s$; (iv) $\mathbf{T}\langle a\rangle\alpha \in s$ and $\mathbf{T}\langle b\rangle\beta \in s$ with $a \neq b$, because in a linear time logic two different actions cannot be executed in the same state.

Given a DLTL formula α, the procedure in [23] builds a Büchi automaton $\mathcal{B}_\alpha$ by associating with each state of the automaton a triple $(\mathcal{F}, x, f)$, where $\mathcal{F}$ is an expanded set of formulas, $x \in \{0, 1\}$ and $f \in \{\downarrow, \checkmark\}$ are used to track fulfillment of until formulas, as we will describe below.

Here instead we assume to be given a transition system TS for a domain description Π, and a DLTL formula α describing constraints and the negation of the property to be proved. We want to extend the construction in [23] to obtain a product automaton whose states combine the states of $\mathcal{B}_\alpha$ with those of the transition system TS. Thus each state s of the product automaton will be a tuple $(\mathcal{F}, w, x, f)$, where $\mathcal{F}, x, f$ are as above, while w is a state of the transition system whose literals are represented as signed formulas (namely, p is represented with $\mathbf{T}p$ and $\neg p$ is represented with $\mathbf{F}p$), with the constraint that $\mathcal{F} \cup w$ is consistent.

The initial states have the form $(\mathcal{F}_0, w_0, 0, \checkmark)$, where $\mathcal{F}_0$ is a set of formulas obtained by applying function $tableau$ to α, and w_0 is an initial state of TS, such that $\mathcal{F}_0 \cup w_0$ is consistent.

To define the transitions of the product automaton we use the functions $nextTSstates(w, a)$, which returns the set of the states of the transition system TS reached with a transition a from state w, and $next\mathcal{F}(\mathcal{F}, a)$, defined in Figure 1, which returns a set of set of formulas obtained by propagating the formulas in $\mathcal{F}$ through action a. This function first checks whether it is possible to execute action a from $\mathcal{F}$, then propagates elementary temporal formulas through a and expands them with *tableau*. Function $next\_states(s, a)$, in Figure 2, returns the set of successor states of s after a.

function $next\mathcal{F}(\mathcal{F}, a)$
if $\mathcal{F}$ does not contain a formula $\mathbf{T}\langle a\rangle\top$ **then return** $\emptyset$
else return $tableau(\{\mathbf{T}\alpha | \mathbf{T}\langle a\rangle\alpha \in \mathcal{F}\} \cup \{\mathbf{F}\alpha | \mathbf{F}\langle a\rangle\alpha \in \mathcal{F}\})$

Fig. 1. Function $next\mathcal{F}$

function $next\_states((\mathcal{F}, w, x, f), a)$
return $\{(\mathcal{F}', w', x', f')$ **such that**
$\mathcal{F}' \in next\mathcal{F}(\mathcal{F}, a)$, $w' \in nextTSstates(w, a)$, $\mathcal{F}' \cup w'$ is consistent,
if there exist no $\mathbf{T}\langle a\rangle\alpha\mathcal{U}_x^{\mathcal{A}(q)}\beta \in \mathcal{F}$ **then** $x' = 1 - x; f' = \checkmark$
else $x' = x; f' = \downarrow\}$

Fig. 2. Function $next\_states$

The fields x and f are used to characterize accepting states of the product automaton, and are used to check that all until formulas are fulfilled in a finite number of steps.

If a state s_i of an accepting run ρ contains the until formula $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$, then there must be a state $s_j, i \leq j$ in ρ satisfying the conditions given by the semantics of until. We say that s_j *fulfills* the until formula. If s_i does not fulfill the until formula, then it is possible to show that, according to the axioms of until, s_i contains a formula $\mathbf{T}\langle a_i\rangle\alpha\mathcal{U}^{\mathcal{A}(q')}\beta$, where $q' \in \delta(q, a_i)$ and, according to function $next\mathcal{F}(\mathcal{F}_i, a_i)$, s_{i+1}

contains a formula $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q')}\beta$. We say that this until formula is *derived* from formula $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ in state s_i. If a state contains an until formula which is not derived from a predecessor state, we say that the formula is *new*. New until formulas are obtained during the expansion of *tableau*.

In order to check fulfillment of until formulas, we must be able to track them along the states of the run. This is done using the field x and by extending accordingly signed formulas so that all true until formulas have a label 0 or 1, i.e. they have the form $\mathbf{T}\alpha\mathcal{U}_l^{\mathcal{A}(q)}\beta$ where $l \in \{0,1\}$. For each state $(\mathcal{F}, w, x, f)$, the label of an until formula in $\mathcal{F}$ is assigned as follows: if it is a derived until formula, then its label is the same as that of the until formula in the predecessor state it derives from, otherwise, if the formula is new, it is given the label $1 - x$.

Let us assume that in a state s_i we have $x = 0$. Then all new until formulas of s_i have label 1, and all until formulas with label 0 must be derived from previous states. If s_i belongs to an accepting run, all until formulas will be fulfilled in a finite number of steps. The value 0 of x is propagated to the next states until a state s_j does not contain any more until formulas with label 0. Then x is switched to 1, and we proceed in the same way. Whenever x changes its value, we set $f = \checkmark$. A state with $f = \checkmark$ is an *accepting state* of the product automaton, and a run ρ containing infinitely many accepting states is an *accepting run*. It is possible to prove that:

Proposition 3. *(a) Any accepting run of the product automaton corresponds to an infinite path of the transition system satisfying the initial DLTL formula α; (b) every infinite path of the transition system which is a model of α corresponds to an accepting run of the product automaton.*

The proof is based on the following theorems proved in [23], dealing with the Büchi automaton $\mathcal{B}_\alpha$ constructed for the formula α.

Theorem 3 in [23]. *Let $M = (\sigma, V)$ and $M, \varepsilon \models \alpha$. Then $\sigma \in \mathcal{L}(\mathcal{B}_\alpha)$.*

Given a temporal model M of α, Theorem 3 states that there exists an accepting run ρ of $\mathcal{B}_\alpha$ over σ, where $\mathcal{B}_\alpha$ is the Büchi automaton associated with α. In particular, ρ is a sequence of nodes of the form $\rho(\tau) = (\mathcal{F}_\tau, x_\tau, f_\tau)$, analogous to states of the product automaton, but without the second component w. By construction of ρ, (and by Lemma 2 in [23]), for each prefix τ of σ, the formulas in $\mathcal{F}_\tau$ are satisfied in M at τ.

Theorem 4 in [23]. *Let $\sigma \in \mathcal{L}(\mathcal{B}_\alpha)$. There is a model $M = (\sigma, V)$ such that $M, \varepsilon \models \alpha$.*

In fact, given an accepting run of $\mathcal{B}_\alpha$ over σ, a temporal model $M = (\sigma, V)$ of α can be defined in such a way that: if $\mathbf{T}p \in \mathcal{F}_\tau$, then $p \in V(\tau)$ and, if $\mathbf{F}p \in \mathcal{F}_\tau$, then $p \notin V(\tau)$. For those propositions p such that neither $\mathbf{T}p \in \mathcal{F}_\tau$, nor $\mathbf{F}p \in \mathcal{F}_\tau$, V can assign an arbitrary value to p in τ.

Proof of Proposition 3. (a) By construction, any accepting run ρ of the product automaton is both a path of the transition system and an accepting run of $\mathcal{B}_\alpha$. According to Theorem 4 in [23], for each accepting run ρ of $\mathcal{B}_\alpha$ over σ there is a model (σ, V) of α. As, for any prefix τ of σ, the propositions (fluents) which do not appear in $\mathcal{F}$ can be assigned an arbitrary value in $V(\tau)$, we choose for them the valuation given by the w component of the state (note that, by construction, $\mathcal{F}$ and w are consistent and w is complete). Thus the infinite path ρ of the transition system is a model of α.

(b) Let ρ over σ be the infinite path of TS, which defines a model $M = (\sigma, V)$ of α where $V(\tau)$ is the set of propositions true in the state w_τ obtained after τ in ρ. Then, by Theorem 3 in [23], $\mathcal{B}_\alpha$ has an accepting run ρ' over σ, such that for each prefix τ of σ, the formulas in $\mathcal{F}_\tau$ are satisfied in M at τ. So, in particular, they are consistent with w_τ and, thus, we can merge ρ and ρ' by obtaining a run of the product automaton. □

function $BMC()$
$k := 0$
do $paths := \{s_0 \stackrel{a_0}{\rightarrow} s_1 \stackrel{a_1}{\rightarrow} \ldots s_{k+1}$ **such that** $s_j \neq s_m$ for $0 \leq j < m \leq k\}$
if $paths = \emptyset$ **then return** $failure$
$path :=$ **choose in** $\{s_0 \stackrel{a_0}{\rightarrow} s_1 \stackrel{a_1}{\rightarrow} \ldots s_{k+1}$ **such that**
$s_0 \stackrel{a_0}{\rightarrow} s_1 \stackrel{a_1}{\rightarrow} \ldots s_{k+1} \in paths$, $s_l = s_{k+1}$ for some $l \leq k$,
s_{acc} is an accepting state for some $l \leq acc \leq k\}$
$k := k + 1$
while $path = null$
return $path$

Fig. 3. Function BMC

The construction of the *(k,l)-loop* is described by the function BMC in Figure 3. The construct **choose in** S returns any of the elements of set S or $null$ if $S = \emptyset$. With $s_0 \stackrel{a_0}{\rightarrow} s_1 \stackrel{a_1}{\rightarrow} \ldots s_i$ we represent a finite path of the product automaton, where s_0 is an initial state and $s_i \in next\_states(s_{i-1}, a_{i-1})$.

BMC executes a loop by incrementing k at each step. It first checks if there is a simple path, i.e., a path of the product automaton without repeated states, of length k. If there is no such path, *BMC* terminates with failure, because there cannot be a longer simple path. Otherwise it looks for a *(k,l)-loop*, that is a path of length $k + 1$ such that $s_{k+1} = s_l$ for some previous state s_l in the path. Furthermore the loop must contain an accepting state. If such a loop is found, it finitely represents an accepting run. Otherwise, k is increased. This algorithm always terminates because the maximal length of the simple paths of the Büchi automaton is finite, and thus *BMC* returns either a *(k,l)-loop* if there is one, or *failure* if it reaches the maximal length without finding a loop. In case of failure, the following property, analogous to Theorem 1 in [10] (as regards the first completeness threshold there), guarantees that the product automaton has no accepting run and hence, by Proposition 3, that there is no path in the transition system satisfying the formula.

Property. Given a Büchi automaton $\mathcal{B}$, $\mathcal{L}(\mathcal{B})$ is nonempty iff there is a simple path $s_0, \ldots, s_k$ with a back arc (s_k, s_l) such that path $s_l, \ldots, s_k$ contains an accepting state.

The proof is analogous to part (b) of the proof of Theorem 1 in [10]. In fact, from a counterexample having the form of a non-simple path $s_0, \ldots, s_k$ with a back arc (s_k, s_l), one with a simple path can be obtained with the (possibly repeated) application of the same transformations in that proof. Observe that in the standard approach for BMC in [7] the path of length k is a path of the transition system and the search cannot be restricted to simple paths without missing solutions. [7] defines techniques for achieving completeness for unnested properties.

The set of tableau rules can be easily extended to deal with other boolean connectives and derived modal operators. In the following, we use tableau rules for $\Box$ and $\Diamond$,

using the equivalences $\Box\beta \equiv (\beta \wedge \bigcirc\Box\beta))$ and $\Diamond\beta \equiv (\beta \vee \bigcirc\Diamond\beta))$. Observe that, as false box formulae correspond to negated until formulas, we need to label them with x.

Example 2. Consider the domain description given in Example 1 with the constraints and the property given in Section 3.2. We describe some steps of the (non deterministic) construction of a *(k,l)-loop* for $k = 7$. For the initial state s_0 we have $w_0 = \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}, x_0 = 0, f_0 = \checkmark$. $\mathcal{F}_0$ contains the following formulas:

$\mathcal{F}_0.1:\ \mathbf{T}\langle begin\rangle\top$
$\mathcal{F}_0.2:\ \mathbf{T}\Box[begin]\langle\mathcal{A}(q_0)\rangle\top$
$\mathcal{F}_0.3:\ \mathbf{F}\Box_1(mail(a) \supset \Diamond\neg mail(a))$
$\mathcal{F}_0.4:\ \mathbf{T}[begin]\langle\mathcal{A}(q_0)\rangle\top -$ from $\mathcal{F}_0.2$
$\mathcal{F}_0.5:\ \mathbf{T} \bigcirc \Box[begin]\langle\mathcal{A}(q_0)\rangle\top$ from $\mathcal{F}_0.2$
$\mathcal{F}_0.6:\ \mathbf{F} \bigcirc \Box_1(mail(a) \supset \Diamond\neg mail(a))$ from $\mathcal{F}_0.3$

The first two formulas are the two constraints, where $\mathcal{A}(q_0)$ is the automaton equivalent to the regular program $sense; (deliver(a) + deliver(b) + wait); begin$ introduced above. The third formula is the property with the **F** label. Note that the $\Box$ operator has label 1 since $x_0 = 0$. All other formulas are obtained by applying the $tableau$ rules[2].

Since $\mathcal{F}_0$ contains the formula $\mathbf{T}\langle begin\rangle\top$, we can only execute action $begin$ in s_0. In s_1 we have $w_1 = \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}$, from the domain description, and $x_1 = 1$, $f_0 = \checkmark$. x_1 changes its value from the previous state, because there are no formulas in s_0 with label 0. $\mathcal{F}_1$ is obtained by propagating the "next" formulas in $\mathcal{F}_0$ and by applying *tableau* to them:

$\mathcal{F}_1.1:\ \mathbf{T}\langle\mathcal{A}(q_0)\rangle_0\top$ from $\mathcal{F}_0.4$
$\mathcal{F}_1.2:\ \mathbf{T}\Box[begin]\langle\mathcal{A}(q_0)\rangle\top$ from $\mathcal{F}_0.5$
$\mathcal{F}_1.3:\ \mathbf{F}\Box_1(mail(a) \supset \Diamond\neg mail(a))$ from $\mathcal{F}_0.6$
$\mathcal{F}_1.4:\ \mathbf{T}\langle sense\rangle\langle\mathcal{A}(q_1)\rangle_0\top$ from $\mathcal{F}_1.1$
$\mathcal{F}_1.5:\ \mathbf{T}[begin]\langle\mathcal{A}(q_0)\rangle\top$ from $\mathcal{F}_1.2$
$\mathcal{F}_1.6:\ \mathbf{T} \bigcirc \Box[begin]\langle\mathcal{A}(q_0)\rangle\top$ from $\mathcal{F}_1.2$
$\mathcal{F}_1.7:\ \mathbf{F}(mail(a) \supset \Diamond\neg mail(a))$ from $\mathcal{F}_1.3$
$\mathcal{F}_1.8:\ \mathbf{F}\neg mail(a))$ from $\mathcal{F}_1.7$
$\mathcal{F}_1.9:\ \mathbf{F}\Diamond\neg mail(a))$ from $\mathcal{F}_1.7$
$\mathcal{F}_1.10:\ \mathbf{F} \bigcirc \Diamond\neg mail(a))$ from $\mathcal{F}_1.9$

Because of $\mathcal{F}_1.4$ the next action will be $sense$. This action is non deterministic, and we choose $w_2 = \{\mathbf{T}mail(a), \mathbf{T}mail(b)\}$. By continuing with the construction, we can get the following path (we omit the value of the $\mathcal{F}_i$'s in the states):

$(\mathcal{F}_0, \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}, 0, \checkmark) \overset{begin}{\rightarrow} (\mathcal{F}_1, \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}, 1, \checkmark) \overset{sense}{\rightarrow}$
$(\mathcal{F}_2, \{\mathbf{T}mail(a), \mathbf{T}mail(b)\}, 0, \checkmark) \overset{deliver(b)}{\rightarrow} (\mathcal{F}_3, \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}, 0, \downarrow) \overset{begin}{\rightarrow}$
$(\mathcal{F}_4, \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}, 0, \downarrow) \overset{sense}{\rightarrow} (\mathcal{F}_5, \{\mathbf{T}mail(a), \mathbf{T}mail(b)\}, 1, \checkmark) \overset{deliver(b)}{\rightarrow}$
$(\mathcal{F}_6, \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}, 1, \downarrow) \overset{begin}{\rightarrow} (\mathcal{F}_7, \{\mathbf{T}mail(a), \mathbf{F}mail(b)\}, 1, \downarrow) \overset{sense}{\rightarrow}$
$(\mathcal{F}_8, \{\mathbf{T}mail(a), \mathbf{T}mail(b)\}, 0, \checkmark)$

Since $\mathcal{F}_8 = \mathcal{F}_2$, the two states n_8 and n_2 are equal. Thus we have an arc back from s_7 to s_2, and the path from s_2 to s_7 contains an accepting state. The path represents a counterexample to the property we wanted to prove.

[2] We consider only the most significant formulas.

Let us modify the domain description by adding a fluent $pr(E)$ which associates a priority to the mailboxes. We can add the following rules:

$[deliver(E)]\neg pr(E)$

$[deliver(E)]pr(E') \leftarrow E \neq E', mail(E')$

$[deliver(E)] \perp \leftarrow \neg pr(E), pr(E'), E \neq E'$

Init $\neg pr(a)$ and **Init** $\neg pr(b)$.

By executing function BMC, we obtain $failure$ after 15 steps. Therefore the property $\Box(mail(a) \supset \Diamond\neg mail(a))$ holds in the modified domain description.

5 An ASP Encoding of BMC

We now provide a translation into ASP of the above procedure for building a path of the product Büchi automaton. We use predicates like `fluent, action, state` to express the type of atoms. As we are interested in infinite runs represented as *(k,l)-loops*, we assume a bound k (a constant) to the number of states. States are represented in ASP as integers from 0 to k. The predicate `occurs(Action,State)` describes transitions.

To encode disjunction we use "choice constructs" provided by clingo [16] and other ASP solvers. Thus we encode a disjunction $a \vee b \vee c$ with `1{a,b,c}`, meaning: choose a subset of $\{a, b, c\}$ of cardinality at least one.

Occurrence of exactly one action per state is encoded as follows, where "`:action(A)`" constrains A to be an action, and `1{}1` constrains cardinality to 1:

```
1{occurs(A,S) : action(A)}1 :- state(S).
```

As we have seen, states are associated with a set of fluent literals, a set of signed formulas, and the values of x and f. Fluent literals are represented with the predicate `holds(Fluent,State)` or `-holds(Fluent,State)`, **T** or **F** formulas with `tt(Formula,State)` or `ff(Formula, State)`, x with the predicate `x(Val,State)` and f with the predicate `acc(State)`, which is true if `State` is an accepting state.

States on the path must be all different, and thus we need to define a predicate `eq(S1,S2)` to check whether the two states $S1$ and $S2$ are equal:

```
eq(S1,S2):- state(S1), state(S2),not diff(S1,S2).
diff(S1,S2):- state(S1),state(S2),tt(F,S1),not tt(F,S2).
diff(S1,S2):- state(S1),state(S2),holds(F,S1),not holds(F,S2).
```

and similarly for all the other components of a state.

The following constraint requires all states up to k to be different:

```
:-state(S1),state(S2),S1!=S2,eq(S1,S2),S1<=k,S2<=k.
```

Furthermore we need constraints stating that there is a transition from state k to a previous state L, and that there is a state S, $L \leq S \leq k$, such that `acc(S)` holds, i.e. S is an accepting state. To do this we compute the successor of state k, and check that it is equal to L[3].

```
1{loop(L) : state(L) : L<=k}1.
:- state(L), loop(L), not eq(L,k+1).
accept:- loop(L), L<=S, S<=k, acc(S).
:- not accept.
```

[3] Since states are all different, there will be at most one state equal to the successor of k.

Given a domain description Π and $\varphi_1, \ldots \varphi_n$, representing DLTL constraints or the negated property, we want to compute the temporal answer sets of the domain description Π satisfying the temporal formulas, if any. The rules in Π can be easily translated to ASP, similarly to [18]. In the following we provide the translation of our running example, see [26] for details.

```
action(sense). action(wait).
action(deliver(a)). action(deliver(b)).
fluent(mail(a)). fluent(mail(b)).
```

action effects:

```
holds(mail(E),NS):- occurs(sense,S), fluent(mail(E)),NS=S+1,
   not -holds(mail(E),NS).
-holds(mail(E),NS):-occurs(deliver(E),S),fluent(mail(E)),NS=S+1.
```

persistence:

```
holds(F,NS):- holds(F,S), fluent(F),NS=S+1,not -holds(F,NS).
-holds(F,NS):- -holds(F,S),fluent(F),NS=S+1,not holds(F,NS).
```

preconditions:

```
:- occurs(deliver(E),S),-holds(mail(E),S).
:- occurs(wait,S), holds(mail(E),S).
```

initial state:

```
-holds(mail(a),0). -holds(mail(b),0).
```

DLTL formulas are represented as ASP terms. In the encoding, each formula $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ is represented as `until(A,q,alpha,beta)`, where the automaton $\mathcal{A}$ is described by the predicates `trans(A,Q1,Act,Q2)` defining transitions, and `final(A,Q)` defining final states. Predicate `x(L,S)` gives the value $L = 0, 1$ of x in state S. We introduce the terms `until(A,q,alpha,beta,L)` and `diamond(Act,alpha)` for encoding labeled until formulas and $\langle a \rangle \alpha$ formulas. The expansion of signed formulas can be formulated by means of ASP rules corresponding to the tableau rules given in the previous section.

Disjunction:

```
1{tt(F1,S),tt(F2,S)}:- tt(or(F1,F2),S).
ff(F1,S):- ff(or(F1,F2),S).
ff(F2,S):- ff(or(F1,F2),S).
```

Negation:

```
ff(F,S):- tt(neg(F),S).
tt(F,S):- ff(neg(F),S).
```

Until:

```
tt(until(Aut,Q,F1,F2,1-Lab),S):-
   tt(until(Aut,Q,F1,F2),S),x(Lab,S), state(S).
tt(or(F2,and(F1,cont_until(Aut,Q,F1,F2,Lab))),S):-
   tt(until(Aut,Q,F1,F2,Lab),S), state(S), final(Aut,Q).
tt(and(F1,cont_until(Aut,Q,F1,F2,Lab)),S):-
   tt(until(Aut,Q,F1,F2,Lab),S), state(S), not final(Aut,Q).
ff(or(F2,and(F1,cont_until(Aut,Q,F1,F2))),S):-
   ff(until(Aut,Q,F1,F2),S), state(S), final(Aut,Q).
ff(and(F1,cont_until(Aut,Q,F1,F2)),S):-
   ff(until(Aut,Q,F1,F2),S), state(S), not final(Aut,Q).
```

where `cont_until(Aut,Q,F1,F2)` represents the nested disjunction contained in the equivalences for until formulas given in Section 4. Thus it can be expanded as:

```
1{tt(diamond(A,until(Aut,Q1,F1,F2,Lab)),S) : trans(Aut,Q,A,Q1)}:-
   tt(cont_until(Aut,Q,F1,F2,Lab),S).
ff(diamond(A,until(Aut,QN,F1,F2)),S):-
   ff(cont_until(Aut,Q,F1,F2),S), trans(Aut,Q,A,QN).
```

Diamond

```
occurs(Act,S):- tt(diamond(Act,F),S).
tt(F,NS):- tt(diamond(Act,F),S), NS=S+1.
ff(F,NS):- ff(diamond(Act,F),S),occurs(Act,S), NS=S+1.
```

Inconsistency of signed formulas is formulated with the following constraints:

```
:- ff(true,S), state(S).
:- tt(F,S), ff(F,S), state(S).
:- tt(F,S), not holds(F,S).
:- ff(F,S), not -holds(F,S).
```

As a difference with the tableau construction, rather than introducing the translation of formula $\mathbf{T}\bigvee_{a\in\Sigma}\langle a\rangle\top$ in the initial state, we include the rule

```
tt(diamond(A,true),S):- occurs(A,S).
```

as we know that at least one action (and at most one) occurs in a state.

Predicates x and acc are defined as follows:

```
cont(S):-state(S),x(Lab,S),tt(diamond(_,until(_,_,__,Lab)),S).
x(Lab,SN):- x(Lab,S),SN=S+1, cont(S).
-acc(SN):- x(Lab,S),SN=S+1, cont(S).
x(1-Lab,SN):- x(Lab,S),SN=S+1, not cont(S).
acc(SN):- x(Lab,S),SN=S+1, not cont(S).
x(0,0). acc(0).
```

Finally, we must add a fact `tt(`$tr(\varphi_i)$`,0)` for each DLTL formula φ_i to be satisfied in the model, where $tr(\varphi_i)$ is the ASP term representing φ_i. It is easy to see that:

Proposition 4. *The size of grounding of the ASP encoding is* $O((|f|+|\phi|^3)\times k^2)$.

Observe that the number of ground instances of predicate `holds` is $O(|f|\times k)$ and that the number of ground instances of predicates `eq` and `diff` is $O(k^2)$. The derived until of a formula $\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ are at most $|\Sigma|\times|\mathcal{A}(q)|$. Hence, the number of the subformulas of the initial formula ϕ is overapproximated by $|\phi|^3$ and the number of the ground instances of predicates `tt`, `ff` is $O(|\phi|^3\times k)$. Thus, the number of ground rules in the encoding is $O((|f|+|\phi|^3)\times k^2)$ for DLTL (but $O((|f|+|\phi|)\times k^2)$) for LTL.

We can prove that there is a one to one correspondence between the temporal answer sets of a domain description satisfying a given temporal formula and the answer sets of the ASP program encoding the domain and the formula.

Theorem 1. *Let Π be a domain description whose temporal answer sets are total and let ϕ be a DLTL formula. If there is a temporal answer set of Π that satisfies the formula ϕ, then there exists, for some k, an answer set of the ASP program $tr(\Pi)\cup tt(tr(\phi),0)$ (where $tr(\Pi)$ is the ASP encoding of Π and $tr(\phi)$ is the ASP term representing ϕ) ; and vice-versa.*

Proof sketch. For the first part, let (σ, S) be a temporal answer set of Π satisfying ϕ. By Proposition 1, it corresponds to a run in the transition system satisfying ϕ. By Proposition 3, there is an accepting run of the product automaton of $\mathcal{B}_{TS}$ and $\mathcal{B}_{\phi}$. Then, in the product automaton there must be an accepting run which is a *(k,l)-loop* and is computed by the function BMC. We can show that from any *(k,l)- loop* computed by function BMC, we can construct an answer set of the ASP program $tr(\Pi) \cup tt(tr(\phi), 0)$. Indeed, any *(k,l)-loop* computed by function BMC corresponds, by Proposition 3, to a run in the transition system satisfying ϕ and, hence (by Proposition 1) to a temporal answer set (σ', S') of Π satisfying ϕ, where the sequence σ' can be finitely represented as a *(k,l)- loop*. By a proof similar to the one of Theorem 1 in [26] we can construct a corresponding answer set of the ASP program $tr(\Pi)$. It is possible to see that this answer set can be extended to an answer set of $tr(\Pi) \cup tt(tr(\phi), 0)$ proving that the ASP encoding of the tableau rules mimics the tableau construction. For the other direction, we can show that, given an answer set of the ASP program $tr(\Pi) \cup tt(tr(\phi), 0)$, we can construct a *(k,l)-loop* which is non-deterministically computed by function BMC (and is an accepting run of the product automaton). By Proposition 3, it corresponds to an infinite path of the transition system satisfying the formula ϕ and, by Proposition 1 it corresponds to a temporal answer set of Π satisfying ϕ. □

According to the construction of the product automaton described in this paper, each state in the path depends only on the previous state. This suggests that the performance of the ASP solver might be improved using an incremental ASP solver. In particular we used iClingo [17]. The rules in an iClingo program are divided into three parts: $\#base$ contains all static rules, $\#cumulative\ k$ contains all rules which depend on k and which will be accumulated over a whole incremental computation, $\#volatile\ k$ whose rules hold only in step k and are dismissed in the next step. In our case, the cumulative part contains all the rules which define the structure of a state, the next action and equality of states. The volatile part contains the rule which define a loop and the presence of an accepting state within the loop (the rule for predicates `loop` and `accept`). For achieving completeness, we must first search for the longest simple path and then use it as an upper bound in the search for a counterexample. The search for the longest simple path can be done removing from the iClingo encoding the volatile part and stopping the computation as soon as an unsatisfiable step max is found (there is no simple path of length max).

Problem	Tableau-BMC	BMC	Problem	Tableau-BMC	BMC
DP(6)	0.11	0.18	MAIL(5)	0.13	0.03
DP(8)	0.78	0.59	MAIL(10)	3.52	10.23
DP(10)	9.89	3.68	MAIL(15)	28.15	timeout
DP(12)	146.51	33.01	MAIL(20)	183.11	timeout

Table 1. Results, compared with the method in [26]

Table 1 provides some results to compare this approach with the method ("BMC") which does not use Büchi automata in [26], in particular, with a Clingo implementation of that method where the solver is invoked with increasing values of k until a model is found. "BMC" is implemented in Clingo, because in the standard formulation of BMC,

the truth value of a temporal formula in a state depends on the next state, and therefore the ASP rules defining the formulas cannot be put in the cumulative part of iClingo. The table provides running times in seconds, or "timeout" for more than 1 hour, on a machine with Intel Xeon E5520 processors (2.26Ghz) and 32 GB RAM, for finding a model for two classes of problems: DP(n), the dining philosophers problems in [30] and appendix C of [26], and MAIL(n), on the domain used as a running example in this paper, with n mailboxes, where a model is searched for the formula:
$\Diamond(\bigwedge_{i=1}^{n} mail(i) \wedge \Diamond(\langle deliver(1)\rangle \top \wedge \Diamond(\langle deliver(2)\rangle \top \wedge \ldots \wedge \Diamond\langle deliver(n)\rangle \top) \ldots))$

For DP(n), the scalability of the two approaches is similar. For MAIL(n), the method in this paper is superior; in particular, the weakness of "BMC" is in detecting that there is no model for values of k smaller than the one for which there is a model.

The search for the longest path is practically feasible only for problems where the action domain is sufficiently constrained. In a variation of the MAIL(n) problems, with the original property to be verified ($\Box(mail(1) \supset \Diamond\neg mail(1))$), which holds due to causal rules which impose to serve the full mailboxes in round robin (so that the next mailbox to serve is uniquely defined, in a more general way wrt the case of 2 mailboxes in Section 4), such a search is feasible up to $n = 6$ (162.31 seconds).

6 Reasoning about epistemic knowledge and incomplete states

As we have seen in the previous sections, our approach to action theories verification is defined for domain descriptions which have total temporal answer sets. This is due to the fact that total temporal answer sets correspond to temporal models (in a two valued temporal logic), and we can verify the satisfiability of temporal formulas over such models. In many situations, however, we may be interested in reasoning about incomplete states, i.e., states in which some fluent is known to be true, some fluent is known to be false, while other fluents are unknown. Answer sets allow a natural representation of incomplete states by partial interpretations. For a proposition p, three cases can be distinguished: either p belongs to the answer set or $\neg p$ belongs to the answer set or neither p nor $\neg p$ belong to the answer set.

However, shifting to partial temporal answer sets would produce a mismatch between the three-valued temporal action logic and the two-valued temporal semantics which is used for verification. To avoid this mismatch, we introduce an epistemic operator in the language to represent epistemic knowledge (and, in particular, epistemic fluent literals) and its dynamic, allowing for actions with knowledge producing and knowledge reducing effects. This approach has the advantage that the ASP encoding of bounded model checking developed in the paper can be exploited as it is, to reason about epistemic states.

The approach we propose is related to Demolombe and Pozos Parra's [14] approach in the situation calculus, to Baral and Son's 0-approximation [5], to the treatment of epistemic fluents in the language Dylog [4], as well as to the treatment of incomplete states in the planning literature [33, 32].

To represent an incomplete state in a concise way, we introduce *epistemic fluent literals* of the form $\mathcal{K}l$ and $\neg\mathcal{K}l$, where $\mathcal{K}$ is an epistemic modality and l a simple fluent literal. $\mathcal{K}l$ means that l is known to be true. We assume $\mathcal{K}$ to be a normal modality,

with a serial accessibility relation (i.e., modal axioms K and D hold for $\mathcal{K}$). Positive and negative introspection axioms do not hold for $\mathcal{K}$. Indeed, following the solution proposed in [4,22], we restrict epistemic modalities to occur only in front of literals. In particular, there are no nested epistemic modalities nor can they be applied to a boolean combination of literals (in particular, $\mathcal{K}(l_1 \vee l_2)$ is not allowed in epistemic states). With these restrictions, positive and negative introspection axioms are clearly useless and the addition of the epistemic operator only doubles the number of fluents in the domain description with respect to the non-epistemic case.

An *epistemic state* (or, simply, a state) is defined as a set of ground epistemic literals. It is said to be *consistent* if, for all simple fluent literals l, it is neither the case that both $\mathcal{K}l$ and $\neg\mathcal{K}l$ belong to the state, nor that both $\mathcal{K}l$ and $\mathcal{K}\neg l$ belong to the state, nor that $\bot$ belongs to the state. We say that an epistemic state is (epistemically) *complete* if, for each ground fluent literal l, either $\mathcal{K}l$ or $\neg\mathcal{K}l$ belongs to the state. A consistent epistemic state provides a *three-valued* interpretation of fluents in which a fluent p is *true* when $\mathcal{K}p$ holds, is *false* when $\mathcal{K}\neg p$ holds, and *undefined* otherwise.

An *epistemic domain description* is a set of epistemic rules of the form:

$$t_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n \tag{6}$$

where the t_i's are either epistemic fluent literals or *temporal-epistemic fluent literals*, i.e., literals of the form $[a]t$ and $\bigcirc t$, with t an epistemic fluent literal. Rule (6) is subject to restrictions (i)-(iii) as rule (1) in Section 3 (where "simple literals" are replaced by "epistemic literals").

The notion of temporal answer set is suitably extended to define temporal epistemic answer sets, where each action sequence $a_1, \ldots, a_k$ is associated with an epistemic state $w^{(\sigma,S)}_{K:\, a_1 \ldots a_k} = \{t : [a_1; \ldots; a_k]t \in S \text{ and } t \text{ is an epistemic literal}\}$. The execution of the actions makes the epistemic state evolve, producing a *revision* of knowledge. Observe that rules of the form (6) are well suited for modeling the epistemic effect of actions, including knowledge producing actions and knowledge losing actions. Consider, for instance, the rules

$$[drop]\mathcal{K}broken \leftarrow \mathcal{K}fragile$$
$$[drop]\neg\mathcal{K}\neg broken \leftarrow \neg\mathcal{K}\neg fragile$$
$$[drop]\neg\mathcal{K}\neg broken \leftarrow \mathcal{K}weakly\_fragile$$
$$[sense\_b]\mathcal{K}broken \leftarrow not\ [sense\_b]\mathcal{K}\neg broken$$
$$[sense\_b]\mathcal{K}\neg broken \leftarrow not\ [sense\_b]\mathcal{K}broken$$

meaning that: if the object is known to be fragile, then it is known to be broken after dropping it; if the object may be fragile, then it may be broken after dropping it; if the object is known to be weakly fragile, then it may be broken after dropping it; the action of sensing causes to know whether the object is broken or not. Notice that, in the second and third rules, the action $drop$ may cause a loss of information (when executed in a state in which $\mathcal{K}\neg broken$ holds), while the action $sense\_b$ has the effect of acquiring information. This shows that the epistemic language allows for a richer representation of actions that, at the object level (with two valued interpretations) is not possible.

To guarantee that $\mathcal{K}f$ and $\mathcal{K}\neg f$ cannot both hold in an epistemic state (as required by seriality), the two causal laws $\neg\mathcal{K}f \leftarrow \mathcal{K}\neg f$ and $\neg\mathcal{K}\neg f \leftarrow \mathcal{K}f$ are introduced,

for each fluent f. Persistence is applied to epistemic literals $\mathcal{K}l$ and $\neg\mathcal{K}l$ (for each l) and makes an epistemic state evolve into a new one. For instance, $\bigcirc\mathcal{K}\ broken \leftarrow \mathcal{K}\ broken \wedge not\ \bigcirc\neg\mathcal{K}\ broken$ and $\bigcirc\neg\mathcal{K}\ broken \leftarrow \neg\mathcal{K}\ broken \wedge not\ \bigcirc\mathcal{K}\ broken$ model the persistence of the epistemic literals $\mathcal{K}\ broken$ and $\neg\mathcal{K}broken$; similar persistence laws apply to the epistemic fluent $\mathcal{K}\neg broken$.

To verify temporal properties of epistemic domain descriptions we extend DLTL formulas as well, to include epistemic fluent literals. Essentially, the epistemic atoms $\mathcal{K}l$ play the role of atomic propositions in DLTL formulas. In particular, we assume that in an *epistemic DLTL formula* all simple fluent literals must occur within the scope of the epistemic operator $\mathcal{K}$ and that epistemic operators $\mathcal{K}$ can only occur in front of simple fluent literals. Although we could define a decidable epistemic extension of DLTL, extending its language with a normal and serial modality $\mathcal{K}$ and providing a suitable Kripke semantics for the epistemic and temporal modalities, the strong restrictions we pose on the occurrence of epistemic modalities in the formulas make this extension useless for the definition and verification of our action theory: we can simply regard the epistemic formulas $\mathcal{K}l$ as being new fluent names, with the seriality requirement encoded by the laws introduced above[4].

The verification of a temporal formula containing epistemic fluents over a temporal epistemic answer set requires, as in the non-epistemic case, that the temporal answer set is total, that is, it is a sequence of consistent and complete epistemic states. In particular, rules of the form $\neg\mathcal{K}l \leftarrow not\ \mathcal{K}l$ are introduced for each epistemic atom $\mathcal{K}l$ to get complete epistemic initial states: if $\mathcal{K}l$ does not hold in the initial state, $\neg\mathcal{K}l$ is assumed to hold. Observe that, as for the non-epistemic case, when the above rules for completing the initial state are present in the domain description and all the epistemic fluents are persistent, one can guarantee that the temporal epistemic answer sets of the domain description are total.

Each total temporal epistemic answer set (σ, S) can be seen as corresponding to a branch in a temporal epistemic model which satisfies the formulas in S (a similar correspondence is stated for temporal deontic models in Proposition 2 of [25]). The evaluation of temporal epistemic properties of the domain description can then be done over the temporal epistemic models corresponding to the answer sets of the domain description. In practice, however, given the restriction on the occurrence of the epistemic operator in the language, the verification of a temporal epistemic formula over an epistemic domain description can be reduced to an object-level verification once all the epistemic literals $\mathcal{K}p$ and $\mathcal{K}\neg p$ are regarded as object-level literals.

As an example of epistemic domain description, let us consider the "bomb in the toilet" problem [15]:

Example 3. We have been alarmed that there is a bomb (exactly one) in a lavatory. There are suspicious packages which could contain the bomb. There is one toilet bowl, and it is possible to dunk a package into it. If the dunked package contains the bomb,

[4] A deontic extension of DLTL for reasoning about obligations has been introduced in [25] to model obligations with deadlines. While in [25] temporal formulas are allowed to occur inside the scope of deontic modalities, here we do not allow temporal formulas to occur within the scope of epistemic operators: the consistency of an epistemic state can be easily be enforced through constraints.

the bomb is disarmed. Here, we consider the variant with uncertain clogging (example BTUC(p) in [15]): the toilet may be clogged or not after having dunked a package in it. The toilet can be unclogged by flushing it.

We consider the fluent names $armed(P)$ and $clogged$ and the action names $dunk(P)$ and $flush$. The domain description $\Pi^{\mathcal{K}}$ contains the following laws:

$[dunk(P)]\neg\mathcal{K}clogged$ $\quad$ $[dunk(P)]\mathcal{K}\neg armed(P)$
$[dunk(P)]\neg\mathcal{K}\neg clogged$ $\quad$ $[dunk(P)]\bot \leftarrow \neg\mathcal{K}\neg clogged$
$[flush]\mathcal{K}\neg clogged$

A package P is known to be non-armed after dunking it, while $clogged$ becomes unknown; P cannot be dunked if the toilet is not known to be unclogged. After flushing, the toilet is known to be unclogged. Persistence laws are introduced for all epistemic fluents $\mathcal{K}clogged, \mathcal{K}\neg clogged, \mathcal{K}armed(P), \mathcal{K}\neg armed(P)$ and their negations, as described above. The initial state is specified as $\{\ \mathcal{K}\neg clogged\}$, and the rules for completing the initial state add to it the epistemic literals $\neg\mathcal{K}\ armed(p)$ and $\neg\mathcal{K}\neg armed(p)$, for all packages p.

Given the specification above, we may want to check, for example, the validity of the following formulas: (F1) $\Box(\mathcal{K}\neg armed(1) \rightarrow \Box\mathcal{K}\neg armed(1))$, i.e., when in a state package 1 is known to be disarmed, it is also known to be disarmed for all later states; (F2) $\neg\Diamond(\mathcal{K}clogged \wedge \langle dunk(1)\rangle\top)$, i.e., there is no reachable state in which the toilet is clogged and package 1 is dunked. Table 2 reports, in column $\Pi^{\mathcal{K}}$, the running times in seconds for verifying F1 and F2 for different values of n, the number of packages in $\Pi^{\mathcal{K}}$. Both the time to compute the upper bound (longest path length) and the time for verification, given the bound, are reported.

Problem	Π		$\Pi^{\mathcal{K}}$		$\Pi_1^{\mathcal{K}}$	
	t bound	t verif	t bound	t verif	t bound	t verif
F1, n=3	16.75	12.19	2.30	2.40	0.19	0.21
F1, n=4	262.17	103.85	15.58	6.25	0.72	0.55
F1, n=5	timeout	-	366.19	30.38	4.44	1.65
F2, n=3	1.05	0.28	0.29	0.29	0.05	0.05
F2, n=4	10.50	10.82	1.91	0.81	0.12	0.13
F2, n=5	134.53	134.27	13.48	3.59	0.39	0.28
F2, n=6	timeout	-	192.80	5.78	5.43	0.57

Table 2.

We also consider an enriched domain description $\Pi_1^{\mathcal{K}}$ with the following additional preconditions that avoid useless actions: $[dunk(P)]\bot \leftarrow \mathcal{K}\ \neg armed(P)$, and $[flush]\bot \leftarrow \mathcal{K}\ \neg clogged$. Table 2 also reports the execution times for the verification of F1 and F2 in $\Pi_1^{\mathcal{K}}$, showing significant savings.

In our experiments we also considered an object-level specification Π of the first domain description $\Pi^{\mathcal{K}}$, in which action $dunk(P)$ non-deterministically causes $clogged$ or $\neg clogged$ and all the possible completions of the initial state are considered (each package may be either armed or not in the initial state). The formulas to be verified are obtained by removing epistemic operators from F1 and F2. We can see that the epistemic version is substantially faster ("timeout" means > 1000).

The main difference between our formalization and the one in [15] is that the language $\mathcal{K}$ does not introduce explicit epistemic constructs: it simply represents the fact that "p is known" by p, the fact that "$\neg p$ is known" by $\neg p$, and the fact that "p is unknown" by the absence of both p and $\neg p$ in the state. Hence, as observed in [15], $\mathcal{K}$ has no specific constructs to express directly the effect of some fluent being unknown. To represent the fact that $clogged$ is unknown after executing $dunk(P)$, $clogged$ and $\neg clogged$ are considered to be inertial only for actions different from $dunk(P)$. Of course, the solution in [15] is more concise as it avoids the introduction of negated epistemic literals, and the approach with epistemic fluents could of course be exploited in $\mathcal{K}$ as well. We do not adopt the solution proposed in [15] because we require states to be complete for verification.

A *knowledge-based* approach has also been used to define the PKS planner, allowing to plan under conditions of incomplete knowledge and sensing [33]. PKS generalizes the STRIPS approach, by representing a state as a set of databases that model the agent's knowledge.

7 Conclusions

The paper presents a bounded model checking approach for the verification of properties of temporal action theories in ASP. The temporal action theory is formulated in a temporal extension of ASP, where DLTL constraints in domain descriptions allow for state trajectory constraints as advocated in PDDL3 [20]. The approach provides a uniform ASP methodology for specifying and verifying domain descriptions. It combines the flexibility of knowledge representation in ASP to encode action domains (and, in particular, to model defeasible actions and causal rules) with temporal verification capabilities providing, unlike [26], a decision procedure for verification and an incremental approach to BMC in ASP. In principle, the approach is applicable to other action languages, provided it is possible to define a transition system, in which a state is only determined from the previous one by a next state function.

Helianko and Niemelä [30] developed a compact encoding of bounded model checking of LTL formulas as the problem of finding stable models of logic programs. In [26] this encoding is extended to address the verification of action domains including DLTL constraints. In this paper, we follow a different approach to BMC which exploits the Büchi automaton construction to achieve completeness.

[10] first proposed the use of the Büchi automaton in BMC. Our encoding in ASP is defined without assuming that the Büchi automaton is computed in advance. The states of the automaton are computed on-the-fly, building the path of the product automaton. This requires equality among states to be checked during the construction of a *(k,l)-loop*, which makes the size of translation quadratic in k. [8] develops efficient encodings of BMC for LTL extended with past operators. In particular, it develops an incremental encoding based on SAT techniques which can be extended with a termination check, thus achieving completeness in the proof of properties. Our incremental encoding exploits the state of the art ASP solver iClingo [17].

The action language in this paper is related to the logic programming based planning language $\mathcal{K}$ [15] and to the languages $\mathcal{C}$ and $\mathcal{C}^+$ [28,27]. Unlike $\mathcal{K}$, $\mathcal{C}$ and $\mathcal{C}^+$, our

action language does not allow for concurrent actions, but it provides general temporal constraints. $\mathcal{K}$, $\mathcal{C}$ and $\mathcal{C}^+$ can perform several kinds of reasoning, such as prediction, postdiction and planning. However, they do not exploit standard temporal logic constructs to reason about actions. A detailed comparison of the language introduced in Section 3 and the languages $\mathcal{K}$, $\mathcal{C}$ and $\mathcal{C}^+$ can be found in [26].

The presence of temporal constraints in our action language is related to the work on temporally extended goals in [12,6], which, however, is concerned with expressing preferences among goals and exceptions in goal specification.

$\mathcal{ESG}$ [11] is a second order extension of CTL* for reasoning about nonterminating Golog programs. The paper presents a method for verification of a first order CTL fragment of $\mathcal{ESG}$, using model checking and regression-based reasoning. Due to first order quantification, this fragment is in general undecidable. DLTL [31] is a decidable LTL fragment of $\mathcal{ESG}$ for which standard LTL model checking techniques can be adopted [23]. Satisfiability in DLTL is known to be PSPACE-complete, as for LTL [31].

In [1] the verification problem for action logic programs with nonterminating behavior is addressed using an action formalism based on a temporalized description logic, $\mathcal{ALCO}$-LTL. DLTL does not allow for first order constructs as $\mathcal{ALCO}$-LTL, while it allows for the specification of regular expressions.

In [9] Cabalar introduces normal forms for Temporal Equilibrium Logic (TEL), an extension of the Answer Set semantics to arbirary theories in the syntax of Linear Temporal Logic. The rules in Π, in our action theories, appear to be in normal form and it would be interesting to investigate the possibility of mapping the LTL fragment of our action theories into TEL.

Acknowledgments: We thank the anonymous referees for their helpful comments. This work has been partially supported by Regione Piemonte, Project ICT4LAW.

References

1. F. Baader, H. Liu, and A. ul Mehdi. Verifying properties of infinite sequences of description logic actions. In *ECAI*, pages 53–58, 2010.
2. F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.
3. J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6):593–618, 2009.
4. M. Baldoni, A. Martelli, V. Patti, and L. Giordano. Programming rational agents in a modal action logic. *Ann. Math. Artif. Intell.*, 41(2-4), 2004.
5. C. Baral and T. C. Son. Formalizing Sensing Actions - A transition function based approach. *Artificial Intelligence*, 125(1-2):19–21, 2001.
6. C. Baral and J. Zhao. Non-monotonic temporal logics for goal specification. In *IJCAI 2007*, pages 236–242, 2007.
7. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
8. A. Biere, K. Heljanko i, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded ltl model checking. *Logical Methods in Computer Science*, 2 (5:5):1–64, 2006.
9. P. Cabalar. A normal form for linear temporal equilibrium logic. In *JELIA, LNCS 6341*, pages 64–76, 2010.

10. E.M. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and complexity of bounded model checking. In *VMCAI*, pages 85–96, 2004.
11. J. Claßen and G. Lakemeyer. A logic for non-terminating Golog programs. In *Proc. KR 2008*, pages 589–599, 2008.
12. U. Dal Lago, M. Pistore, and P: Traverso. Planning with a language for extended goals. In *Proc. AAAI02*, 2002.
13. G. De Giacomo, F. Patrizi, and S. Sardiña. Generalized planning with loops under strong fairness constraints. In *Proc. KR 2010*, 2010.
14. R. Demolombe and M. Pozos Parra. A simple and tractable extension of situation calculus to epistemic logic. In *ISMIS*, pages 515–524, 2000.
15. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM TOCL*, 5(2):206–263, 2004.
16. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Comm.*, 24(2):105–124, 2011.
17. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In *Proc. ICLP08*, volume 5366 of *LNCS*, pages 190–205, 2008.
18. M. Gelfond. *Handbook of Knowledge Representation, ch. 7, Answer Sets*. Elsevier, 2007.
19. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proc. of the 5th Int. Conf. and Symposium*, 1988.
20. A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. *Technical Report, Department of Electronics and Automation, University of Brescia, Italy*, 2005.
21. R. Gerth, D. Peled, M.Y.Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *15th Work. Protocol Specification, Testing and Verification*, 1995.
22. L. Giordano and A. Martelli. Reasoning about web services in a temporal action logic. In *Reasoning, Action and Interaction in AI Theories and Systems, LNAI 4155*. Springer, 2006.
23. L. Giordano and A. Martelli. Tableau-based automata construction for Dynamic Linear time Temporal Logic. *Annals of Mathematics and AI*, 46(3):289–315, 2006.
24. L. Giordano, A. Martelli, and C. Schwind. Specifying and verifying interaction protocols in a temporal action logic. *Journal of Applied Logic*, 5:214–234, 2007.
25. L. Giordano, A. Martelli, and D. Theseider Dupré. Temporal deontic action logic for the verification of compliance to norms in asp. In *ICAIL 2013, 14th Int. Conf. on AI and Law*.
26. L. Giordano, A. Martelli, and D. Theseider Dupré. Reasoning about actions with temporal answer sets. *TPLP*, 13(2), 2013.
27. E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, , and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1-2):49–104, 2004.
28. E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *AAAI/IAAI*, pages 623–630, 1998.
29. F. Giunchiglia and P. Traverso. Planning as model checking. In *Proc. The 5th European Conf. on Planning (ECP'99)*, pages 1–20, 1999.
30. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *TPLP*, 3(4-5):519–550, 2003.
31. J.G. Henriksen and P.S. Thiagarajan. Dynamic linear time temporal logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.
32. H. Palacios and H. Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *AAAI*, pages 900–905, 2006.
33. R. P. A. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *AIPS*, pages 212–222, 2002.
34. M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *Proc. IJCAI 2001*, pages 479–486, 2001.
35. S. Sohrabi and S. A. McIlraith. Optimizing web service composition while enforcing regulations. In *ISWC 2009, Chantilly, USA, LNCS 5823*, pages 601–617, 2009.