Achieving Completeness in Bounded Model Checking of Action Theories in ASP *

Laura Giordano
DiSIT
Università del Piemonte
Orientale, Italy

Alberto Martelli
Dipartimento di Informatica
Università di Torino
Italy

Daniele Theseider Dupré
DiSIT
Università del Piemonte
Orientale, Italy

Abstract

Temporal logics can be used in reasoning about actions for specifying constraints on domain descriptions and temporal properties to be verified. In this paper, we exploit bounded model checking (BMC) techniques in the verification of *dynamic linear time temporal logic* (DLTL) properties of an action theory, which is formulated in a temporal extension of *answer set programming* (ASP). To achieve completeness, we propose an approach to BMC which exploits the Büchi automaton construction while searching for a counterexample. We provide an encoding in ASP of the temporal action domain and of bounded model checking of DLTL formulas.

Introduction

Temporal logics are well suited for reasoning about actions, as they allow for the specification of temporal constraints in a domain description as well as for the verification of temporal properties of the domain. In planning, CTL and LTL have been used in the specification of temporally extended goals (e.g., in (Pistore and Traverso 2001; Baier, Bacchus, and McIlraith 2009)), search control knowledge (Bacchus and Kabanza 2000), and fairness constraints (De Giacomo, Patrizi, and Sardiña 2010). Claßen and Lakemeyer (2008) introduced a second order extension of CTL*, $\mathcal{ESG}$, to 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 exploit Bounded Model Checking (BMC) techniques in the verification of properties of an action theory formulated in a temporal extension of *answer set programming* (ASP). BMC, as defined in (Biere et al. 2003), provides, in general, a partial decision procedure for the verification of a property, since it searches for a counterexample of the property as a path of length k, with increasing values of k; upper bounds for k are determined for some classes of properties, namely unnested properties. Clarke et al. (2004) address the problem of completeness with a *semantic* translation scheme, based on Büchi automata.

Helianko and Niemelä (2003) 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 (Giordano, Martelli, and Theseider Dupré 2012) this encoding is extended to deal with action theories, including complex actions expressed in Dynamic Linear Time Temporal Logic (DLTL) (Henriksen and Thiagarajan 1999). DLTL extends LTL with an until operator $\mathcal{U}^\pi$ indexed by a program π which, as in PDL, is a regular expression of atomic actions. Next state modalities indexed by single actions can be defined from $\mathcal{U}^\pi$.

The above papers do not address the problem of achieving completeness; to this end, in this paper we propose an alternative encoding of BMC of DLTL formulas in ASP, where the search for a counterexample exploits the Büchi automaton construction (Gerth et al. 1995) as well as the transition system. Unlike (Clarke et al. 2004), 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 terminating in a loop back to a previous state l, in which the states are all distinct from each other. A counterexample is searched up to a value for k such that in the product automaton there is no path of length k whose states are all distinct from each other.

Verification is performed on a transition system provided by a domain description in a temporal action theory, whose semantics is defined in terms of *temporal answer sets* (Giordano, Martelli, and Theseider Dupré 2012). Temporal properties are 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. The correctness and completeness of this encoding is based on the results in (Henriksen and Thiagarajan 1999; Giordano and Martelli 2006). The encoding in ASP uses a number of ground atoms which is linear in the size of the formula and quadratic in k. Thanks to the completeness result, we provide a decision procedure for the verification of satisfiability and validity properties of an action domain.

Temporal action language

In this paper, a *domain description* Π is a set of laws describing the effects of actions and their executability pre-

*This work has been partially supported by Regione Piemonte, Project ICT4Law.

conditions, provided in a first-order extension of the temporally extended logic programming language in (Giordano, Martelli, and Theseider Dupré 2012), where temporal fluent literals are used, e.g., $[a]l$ means that after executing a, the literal l is true; $\bigcirc l$ means that l is true in the next state. The following example gives the flavor of the language.

Example 1 A mail delivery agent may check, with the action $sense$, if there is mail in the mailbox of employees, and may deliver mail to them. The immediate effects, persistency and precondition laws are:

$[deliv(E)]\neg mail(E)$
$[sense]mail(E) \leftarrow \ not\ [sense]\neg mail(E)$
$\bigcirc mail(E) \leftarrow mail(E), not \bigcirc \neg mail(E)$
$\bigcirc \neg mail(E) \leftarrow \neg mail(E), not \bigcirc mail(E)$
$[deliv(E)] \perp \leftarrow \neg mail(E)$
$[wait] \perp \leftarrow mail(E)$

i.e.: after delivering the mail to E, there is no mail for E any more; $sense$ may (non-monotonically) cause $mail(E)$ to become true; the fluent $mail$ persists; if there is no mail for E, $deliv(E)$ is not executable, while, if there is mail for E, $wait$ is not executable. As a result of the interaction of persistence and the action law, $sense$ is nondeterministic: executing it when there is no mail for E, $mail(E)$ may either persist false, or become true due to the action law.

We assume that there are only two employees, a and b, and, in the initial state, there is mail for a and not for b, i.e. Π includes **Init** $mail(a)$ and **Init** $\neg mail(b)$, where the **Init** prefix specifies that a law only applies to the initial state.

DLTL formulas may be added as constraints on the possible executions, or they may encode properties to be verified on the enriched domain description. For example,

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

impose that the agent continuously executes a loop where it senses mail and delivers the mail.

We may want to check that, if there is mail for a, the agent will eventually deliver it, i.e.: $\Box(mail(a) \supset \Diamond\neg mail(a))$. This does not hold, as there is a possible scenario in which there is always mail for a and for b, but the mail is repeatedly delivered to b and never to a.

The semantics of a domain description is given, following (Giordano, Martelli, and Theseider Dupré 2012), via *temporal answer sets*, which extend the notion of *answer set* to capture the linear structure of temporal models. Although the 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. A domain description with total answer sets defines a *transition system*.

Verification of domain descriptions

Given an enriched domain description $(\Pi, \mathcal{C})$ (where $\mathcal{C}$ is a set of DLTL formulas, the constraints), its extensions are the temporal answer sets of Π satisfying the constraints $\mathcal{C}$.

Verification may be formulated as *satisfiability* of a formula φ, or, as in the example above, as *validity* of a formula φ, usually reduced to the *unsatisfiability* of $\neg\varphi$, in which case, a model satisfying $\neg\varphi$, if any, represents a counterexample to the validity of φ.

Satisfiability and validity problems can be solved by means of *model checking* techniques. The standard approach to model checking for LTL is based on Büchi automata. The satisfiability problem for a LTL formula α can be solved by constructing a Büchi automaton $\mathcal{B}_\alpha$ (Gerth et al. 1995) such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if α is satisfiable.

Given a system modeled by a transition system TS, which corresponds to a Büchi automaton $\mathcal{B}_{TS}$, *model checking* verifies that the property α holds for the system, by constructing the *product automaton* of $\mathcal{B}_{TS}$ and $\mathcal{B}_{\neg\alpha}$, and by checking for emptiness of the accepted language.

Biere et al. (2003) showed that model checking can be 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 the length k until a model satisfying α is found — if one exists.

A BMC problem can be efficiently reduced to a propositional satisfiability problem or to an ASP problem (Heljanko and Niemelä 2003). 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 in (Biere et al. 2003) for some kinds of LTL formulas.

BMC with Büchi automata

We show now how to adapt the procedure for building a Büchi automaton corresponding to a given DLTL formula (Giordano and Martelli 2006) to the construction of a *(k,l)-loop* corresponding to a run of the product Büchi automaton.

We assume that, as in (Giordano and Martelli 2006), $until$ formulas are indexed with finite automata rather than regular expressions. Thus, we have $\alpha\mathcal{U}^{\mathcal{A}(q)}\beta$ instead of $\alpha\mathcal{U}^{\pi}\beta$, where $\mathcal{A}(q)$ a finite automaton with initial state q and $\mathcal{L}(\mathcal{A}(q)) = [[\pi]]$.

The construction of states uses tableau rules which handle DLTL *signed formulas*, i.e. formulas prefixed with **T** or **F**.

A rule $\phi \Rightarrow \psi_1, \psi_2$ means that if ϕ is in a state, ψ_1 and ψ_2 are added; a rule $\phi \Rightarrow \psi_1 | \psi_2$ means that if ϕ is in a state, the state is duplicated, ψ_1 is added to a copy and ψ_2 to the other one. The rules for disjunction and the **T** rules for $until$ are provided below; other rules are similar, and further rules can be given for derived connectives and modal operators.

Tor: $\mathbf{T}(\alpha \vee \beta) \Rightarrow \mathbf{T}\alpha | \mathbf{T}\beta$
For: $\mathbf{F}(\alpha \vee \beta) \Rightarrow \mathbf{F}\alpha, \mathbf{F}\beta$
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 is a final state)
TuntilNFS: $\mathbf{T}\alpha\mathcal{U}^{\mathcal{A}(q)}\beta \Rightarrow \mathbf{T}(\alpha \wedge \bigvee_{a\in\Sigma}\langle a\rangle$

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

Figure 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 $x' = x; f' = \downarrow\}$

Figure 2: Function $next\_states$

$\bigvee_{q' \in \delta(q,a)} \alpha\mathcal{U}^{\mathcal{A}(q')}\beta)(q$ is not a final state)

In the construction we use a function *tableau* which takes as input a set of signed formulas and returns a set of consistent sets of signed formulas, obtained by repeatedly applying the above rules until all formulas in all sets have been expanded. Note that this construction may create new sets, because of splitting rules such as Tor.

We describe now how to build a path of the product automaton, which is constructed by the BMC procedure while searching for a counterexample. Each state s of the path is a tuple $s = (\mathcal{F}, w, x, f)$, where $\mathcal{F}$ is an expanded set of formulas, w is a state of the transition system whose literals are represented as signed formulas, $x \in \{0, 1\}$ and $f \in \{\downarrow, \checkmark\}$ are used to track fulfillment of until formulas.

Given a domain description Π with the associated transition system TS, and a DLTL formula α describing constraints and properties to be proved, the initial states will 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.

Transitions of the product automaton are defined by function $next\_states(s, a)$, defined in Figure 2, which returns the set of successor states of s after a. This function makes use of 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 $next\mathcal{F}(\mathcal{F}, a)$, which returns a set of formulas obtained by propagating the formulas in $\mathcal{F}$ through action a. Function $next\mathcal{F}$ is defined in Figure 1. 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*.

The fields x and f are used to characterize accepting states of the product automaton, and to check that all until formulas are fulfilled in a finite number of steps. In order to do this, all true until formulas are extended with a label 0 or 1, i.e. they have the form $\mathbf{T}\alpha\mathcal{U}_l^{\mathcal{A}(q)}\beta$ where $l \in \{0, 1\}$. Roughly speaking, when an until formula is created, it is given the label $1 - x$, while, if it is propagated from a previous state, it keep the previous label. When there are no more until formulas with label x, then x switches to $1 - x$ and f

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

Figure 3: Function BMC

function $max()$
$i := 0$
do
$i := i + 1$
$path :=$ **choose in** $\{s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots s_i$ **such that**
$s_j \neq s_m$ for $0 \leq j < m \leq i\}$
while $path \neq null$
return $i - 1$

Figure 4: Function max

is set to $\checkmark$. A state with $f = \checkmark$ is an *accepting state* of the product automaton, and a run ρ containing infinite accepting states is an *accepting run*.

It is an obvious consequence of the construction that:

Proposition 1 *(i) Any accepting run of the product automaton corresponds to an infinite path of the transition system (i.e., a temporal answer set of* Π*) satisfying the initial DLTL formula* α*; (ii) every infinite path of the transition system which is a model of* α *corresponds to an accepting run of the product automaton.*

The proof of this proposition, omitted for lack of space, exploits Theorems 4 and 5 in (Giordano and Martelli 2006).

Our approach to BMC relies on the well known result that the language accepted by a Büchi automaton is nonempty iff there is a reachable accepting state with a cycle back to itself. The construction of the *(k,l)-loop* is described by function BMC in Figure 3. The construct **choose in** S returns any of the elements of set S or $null$ if $S = \emptyset$. With $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots s_i$ we represent a finite path of the product automaton, where s_0 is an initial state and $s_i \in next\_states(s_{i-1}, a_{i-1})$. Given an integer k, we look for 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 until max_k is reached.

Observe that we can consider only *simple* paths, that is, paths without repeated states. This property allows to define a terminating algorithm, thus achieving *completeness*, by passing the length of the longest simple path as parameter to BMC.

The length of the longest simple path can be found iteratively, searching for a simple path of length i (without loop), and incrementing i at each step (See Figure 4). Since the number of different states is finite, this procedure terminates.

Encoding bounded model checking in ASP

We now outline a translation into standard ASP of the above procedure for building a path of the product Büchi automaton (for more details see (Giordano, Martelli, and Theseider Dupré 2011)). 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 to the number of states. States are represented in ASP as integers from 0 to K, where K is given by the predicate `laststate(State)`. The predicate `occurs(Action,State)` describes transitions. Occurrence of exactly one action per state can be encoded as:

```
-occurs(A,S):- occurs(A1,S),action(A),
   action(A1),A!=A1,state(S).
occurs(A,S):- not -occurs(A,S),action(A),
   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)`, **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 two states $S1$ and $S2$ are equal.

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

```
:- state(S1), state(S2), S1!=S2, eq(S1,S2),
   laststate(K), 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 S.

```
loop(L):- state(L), laststate(K), L<=K,
   SuccK=K+1, eq(L,SuccK).
accept:- loop(L), state(S), laststate(K),
    L<=S, S<=K, acc(S).
:- not accept.
```

Given a domain description Π and a set of DLTL formulas, representing constraints or negated properties, 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, e.g., the effect rule and precondition for *deliv* in the example become:

```
-holds(mail(E),NS):- occurs(deliv(E),S),
   fluent(mail(E)), NS=S+1.
:- occurs(deliv(E),S),-holds(mail(E),S).
```

DLTL formulas are represented as ASP terms. For instance, in the encoding, each labeled until formula is represented as `until(A,Q,Alpha,Beta,Label)`, where the automaton $\mathcal{A}$ is described by the predicates `trans(A,Q1,Act,Q2)` defining transitions, and `final(A,Q)` defining final states.

The translation of the tableau rules can be formulated by means of ASP rules such as:

```
tt(or(F2,and(F1,
    diamond(Act,until(Aut,Q1,F1,F2,L)))),S):-
  tt(until(Aut,Q,F1,F2,L),S), state(S),
  label(L),final(Aut,Q),occurs(Act,S),
  choose(until(Aut,Q,F1,F2,L),S,Act,Q1).
tt(and(F1,
    diamond(Act,until(Aut,Q1,F1,F2,L))),S):-
  tt(until(Aut,Q,F1,F2,L),S), state(S),
  label(L),not final(Aut,Q),occurs(Act,S),
  choose(until(Aut,Q,F1,F2,L),S,Act,Q1).
```

The predicate `choose` non deterministically chooses a transition `Q1` among the ones possible for action `Act` in the automaton `Aut`, and uses that choice in the expansion of the until formula. The term `diamond(Act,alpha)` encodes $\langle act \rangle \alpha$.

Note that, to express splitting of sets of formulas, as in the case of disjunction, we can exploit disjunction in the head of clauses, provided by some ASP languages such as DLV, or choice constructs available in other languages.

Furthermore, 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 the (grounding of the) encoding in ASP is linear in the size of the formula ϕ to be verified and in the number f of ground fluents while quadratic in the size of k.

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

Proposition 2 *Let Π be a domain description whose temporal answer sets are total, let $tr(\Pi)$ be the ASP encoding of Π (for a given k), and let ϕ be a DLTL formula.*

If there is a temporal answer set of Π that satisfies the formula ϕ, then there exists an answer set of the ASP program $tr(\Pi) \cup tt(tr(\phi), 0)$ (where $tr(\phi)$ is the ASP term representing ϕ); and vice versa.

For achieving completeness, the search for the longest simple path can be done by removing from the above ASP encoding the rules for defining loops and the rules for defining the Büchi acceptance condition.

The translation has been run in iClingo (Gebser et al. 2008). For the dining philosophers problems in (Heljanko and Niemelä 2003), the scalability of the approach in this paper is similar to the one for the method — without Büchi automaton — in (Giordano, Martelli, and Theseider Dupré 2012) and the one in (Heljanko and Niemelä 2003), when looking for a counterexample. E.g., a counterexample for DP(12) is found in 183 seconds, wrt 274 seconds for a Clingo implementation of the method in (Giordano, Martelli, and Theseider Dupré 2012) — see also Appendix C in that paper.

The search for the longest simple path is substantially more costly and practically feasible only for problems where the action domain is sufficiently constrained.

Conclusions

We have presented 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 the domain description allow for state trajectory constraints to be captured. The approach provides a uniform ASP metodology for specifying domain descriptions and for verifying them, which can be used for several reasoning tasks, including business process verification (D'Aprile et al. 2010) and planning with temporal constraints (Bacchus and Kabanza 2000).

Unlike (Heljanko and Niemelä 2003; Giordano, Martelli, and Theseider Dupré 2012), this paper provides a decision procedure for BMC, but it does not assume, as in (Clarke et al. 2004), that a Büchi automaton is computed in advance.

The action language in this paper is related to the logic programming based planning language $\mathcal{K}$ (Eiter et al. 2004), which is well suited for planning under incomplete knowledge. Unlike $\mathcal{K}$, the action language in this paper does not allow for concurrent actions, but it provides temporal constraints. $\texttt{DLV}^{\mathcal{K}}$ (Eiter et al. 2003), which implements $\mathcal{K}$ in the disjunctive logic programming system DLV, does not allow to express and verify temporal properties.

$\mathcal{C}$ and $\mathcal{C}^+$ (Giunchiglia and Lifschitz 1998; Giunchiglia et al. 2004) also deal with actions with indirect and nondeterministic effects and with concurrent actions. Their semantics is based on a nonmonotonic causal logic. Several kinds of reasoning (prediction, postdiction or planning) can be performed, however the languages do not exploit standard temporal logic constructs to reason about actions.

$\mathcal{ESG}$ (Claßen and Lakemeyer 2008) 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. Because of first order quantification, this fragment is in general undecidable.

In (Baader, Liu, and ul Mehdi 2010) 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, obtained from LTL by allowing $\mathcal{ALCO}$-assertions in place of propositions. The behaviors of the program on which verification is performed are given by a Büchi automaton. As a difference, in our approach the action domain is given as a temporal ASP action theory. Concerning the verification language, DLTL does not allow for first order constructs as $\mathcal{ALCO}$-LTL, while it allows for the specification of regular expressions.

References

Baader, F.; Liu, H.; and ul Mehdi, A. 2010. Verifying properties of infinite sequences of description logic actions. In *ECAI*, 53–58.

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

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artif. Intell.* 173(5-6):593–618.

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

Clarke, E.; Kroening, D.; Ouaknine, J.; and Strichman, O. 2004. Completeness and complexity of bounded model checking. In *VMCAI*, 85–96.

Claßen, J., and Lakemeyer, G. 2008. A logic for nonterminating Golog programs. In *Proc. KR 2008*, 589–599.

D'Aprile, D.; Giordano, L.; Gliozzi, V.; Martelli, A.; Pozzato, G. L.; and Theseider Dupré, D. 2010. Verifying Business Process Compliance by Reasoning about Actions. In *CLIMA XI*, volume 6245 of *LNAI*.

De Giacomo, G.; Patrizi, F.; and Sardiña, S. 2010. Generalized planning with loops under strong fairness constraints. In *Proc. KR 2010*.

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 Trans. Comput. Log.* 5(2):206–263.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2008. Engineering an incremental ASP solver. In *Proc. ICLP08*, volume 5366 of *LNCS*, 190–205.

Gerth, R.; Peled, D.; M.Y.Vardi; and Wolper, P. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. 15th Work. Protocol Specification, Testing and Verification*.

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

Giordano, L.; Martelli, A.; and Theseider Dupré, D. 2011. Achieving completeness in bounded model checking of action theories in ASP . Technical report, TR-INF-2011-12-04-UNIPMN, Dip. Informatica, Univ. Piemonte Orientale.

Giordano, L.; Martelli, A.; and Theseider Dupré, D. 2012. Reasoning about actions with temporal answer sets. *Theory and Practice of Logic Programming*.

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

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

Heljanko, K., and Niemelä, I. 2003. Bounded LTL model checking with stable models. *TPLP* 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.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *Proc. IJCAI 2001*, 479–486.