# Constraint-Based Algorithm for Computing Temporal Invariants

Jussi Rintanen

Department of Information and Computer Science
Aalto University, Helsinki, Finland*

**Abstract.** Automatically identified invariants are an important part of reductions of state-space reachability problems to SAT and related formalisms as a method of pruning the search space. No general algorithms for computing temporal invariants have been proposed before. Earlier algorithms restrict to unconditional actions and at-most-one invariants. We propose a powerful inductive algorithm for computing invariants for timed systems, showing that a wide range of timed modeling languages can be handled uniformly. The algorithm reduces the computation of timed invariants to a sequence of temporal logic consistency tests.

## 1 Introduction

Invariants are facts that hold in all reachable states of a transition system. In search methods other than explicit state space search, including symbolic search with SAT [9] and backward chaining search, the search space includes (partial) states that are not reachable from the initial states. For these search methods invariants help pruning the search space. Additionally, as an approximate upper-bound for the set of all reachable states, invariants can help in analyzing properties of the state space, with applications in planning, verification, diagnosis, and other forms of reasoning about transition systems.

The leading methods for computing invariants for untimed/asynchronous systems can be viewed as approximations of exact symbolic methods for computing the set of all reachable states, such as those based on binary decision diagrams [2]. These methods inductively compute a sequence of sets of states reachable with a given number of actions. Upon reaching a fixpoint, the computation terminates. Most works on invariants have adopted the inductive construction [10,6,7,11,12] which is well understood in the context of untimed/asynchronous systems.

The conceptual difficulties about reasoning with partial *temporal* states, as well as the concurrency of actions, have hampered attempts to apply the inductive construction in the timed setting. The main challenges in the timed setting are, first, identifying the form of induction suitable for timed systems with several concurrent and temporally overlapping actions, and, second, developing sufficiently powerful and efficient temporal reasoning methods for handling complex timed transition system models. We present solutions to both of these problems.

* Also affiliated with Griffith University, Brisbane, Australia, and the Helsinki Institute of Information Technology, Finland. This work was funded by the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170).

E. Fermé and J. Leite (Eds.): JELIA 2014, LNAI 8761, pp. 665–673, 2014.

The structure of the paper is as follows. After formal preliminaries in Section 2, the new algorithm is presented in Section 3. A core component of the algorithm, temporal consistency tests, is presented in Section 4. Experiments with the algorithm are summarized in Section 5 before concluding the paper in Section 6.

## 2 Problem Definition

Formulas $x$ and $\neg x$ for $x \in X$ are *literals*. The *complement* $\overline{l}$ of a literal $l$ is defined by $\overline{x} = \neg x$ and $\overline{\neg x} = x$. We define actions as pairs consisting of a precondition (a propositional formula) and an effect which indicates how and when state variables change. Effects are conditional on values of state variables at the time instant the action is taken.

**Definition 1 (Actions).** *Let $X$ be a finite set of (Boolean) state variables. An* action *over $X$ is a pair $(p, e)$ where $p$ is a propositional formula over $X$ and $e$ is a set of rules $\phi \rhd l@t$, where $\phi$ is a propositional formula over $X$, $l$ is a literal over $X$, and $t > 0$ is a rational number. Effect $l$ takes place after time $t$ has passed provided that $\phi$ was true.*

**Definition 2 (Transition systems).** *A transition system is a 3-tuple $\langle X, I, A\rangle$ where $X$ is a finite set of (Boolean) state variables, $I : X \to \{0, 1\}$ is the initial state (a total function from state variables to 0 and 1), and $A$ is a set of actions.*

Above we have left out one important component of timed systems, dependencies between actions that prevent some combinations of actions being taken. Actions may use the same (implicitly represented) resources, and hence cannot temporally overlap. Several alternative definitions of this kind of exclusions between actions are possible [13,8,4], and here we only assume that exclusions can be factored to binary relations between actions: if a given action $a_1$ is taken at time $t$, then another action $a_2$ cannot be taken during the interval $[t + t_1, t + t_2]$ for some $t_1$ and $t_2$ such that $0 \leq t_1 \leq t_2$.

*Plans* are finite sets $P \subseteq A \times \mathbb{Q}+$ that schedule actions so that action exclusions are respected: if $a_1 \in A$ is exclusive of $a_2 \in A$ being taken at $[t_1, t_2]$, and $(a_1, t) \in P$, then $(a_2, t') \in P$ for no $t'$ such that $t + t_1 \leq t' \leq t + t_2$.

**Definition 3 (Plans and executions).** *Given a transition system $\langle X, I, A\rangle$, an* execution *for a* plan *$P$ is a mapping $v : \mathbb{Q} \times X \to \{0, 1\}$ from time points and state variables to 0 and 1 such that*

1. *$v(0, x) = I(x)$ for all $x \in X$,*
2. *$v(t, p) = 1$ if $(a, t) \in P$ and $a = (p, e)$, where $v(t, p)$ denotes the obvious generalization of the values $v(t, x), x \in X$ to arbitrary Boolean formulas $p$,*
3. *if $(a, t) \in P$ and $\phi \rhd l@t' \in a$, then $v(t + t', l) = 1$,*
4. *state variables not changed by actions retain their values: for any $t_l$ and $t_u$ such that $t_l < t_u$, if*
   - *$v(t_l, x) = 1$, and*
   - *there is no $(a, t') \in P$ such that $\phi \rhd \neg x@t'' \in e$ (where $a = (p, e)$) and $t_l \leq t' + t'' \leq t_u$*

   *then $v(t_i, x) = 1$ for all $t_i$ such that $t_l < t_i \leq t_u$. (Analogously for $v(t_l, x) = 0$.)*

For a given transition system $T = \langle X, I, A\rangle$, propositional formulas $\phi$ such that $v(t, \phi) = 1$ for every execution $v$ of $T$ and every $t \in \mathbb{Q}$ are *invariants*. Later we use a generalization of this notion to temporal logic formulas.

### 2.1 Temporal Logic Representations

We use a linear temporal logic for reasoning about actions and invariants, with both actions and invariants represented as formulas in the logic.

**Definition 4.** *Let $\Sigma = \{x_1, \ldots, x_n\}$ be a set of atomic propositions. Then our temporal language consists of exactly those formulas that are obtained with the following inductive definition.*

1. *$x$ is a formula for every $x \in \Sigma$.*
2. *Formulas with $\neg$, $\vee$ and $\wedge$ are defined in the usual way.*
3. *$[t_0, t_1]\phi$ is a formula if $\phi$ is a formula and $t_0$ and $t_1$ are rational numbers such that $t_0 \leq t_1$. This is a metric temporal modal operator saying that $\phi$ holds at all time points $t$ such that $t_{now} + t_0 \leq t \leq t_{now} + t_1$ where $t_{now}$ is the time point in which the formula is evaluated. The formula $[t]\phi$ is defined as an abbreviation for $[t, t]\phi$.*
4. *$\phi_1 \mathcal{U} \phi_2$ is a formula if $\phi_1$ and $\phi_2$ are formulas. This is the temporal operator* until*, which says that if $\phi_1$ is true in all time points until $\phi_2$ is true.*

Boolean connectives $\rightarrow$ and $\leftrightarrow$ are defined by $\phi \rightarrow \psi \equiv_{def} \neg\phi \vee \psi$ and $\phi \leftrightarrow \psi \equiv_{def} (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

### 2.2 Semantics for Temporal Formulas

Temporal formulas are evaluated with respect to linear temporal models $v : \mathbb{Q} \times \Sigma \rightarrow \{0, 1\}$ that assign a truth value to every rational time point and atomic proposition. Consequently, we can identify executions with linear temporal models. Formulas have a standard semantics, with the truth of formulas at time point $t$ denoted by $v \models_t \phi$.

**Definition 5.** *The truth of a temporal formula $\phi$ at time point $t$ in a given model $v$ is recursively defined as follows.*

1. *$v \models_t b$ iff $v(t, b) = 1$, for atomic propositions $b \in \Sigma$.*
2. *Truth with truth-functional $\neg$, $\vee$ and $\wedge$ is as usual.*
3. *$v \models_t \phi \mathcal{U} \psi$ iff $v \models_{t'} \phi$ for all $t' \geq t$ such that $v \not\models_{t''} \psi$ for all $t''$ such that $t \leq t'' \leq t'$.*
4. *$v \models_t [t_1, t_2]\phi$ iff $v \models_{t'} \phi$ for all $t'$ such that $t_1 \leq t' \leq t_2$.*

We also use half-open and open intervals with the operators $]t_1, t_2]$, $[t_1, t_2[$, and $]t_1, t_2[$, which are defined analogously. The operator $\Box$ is identified with $]-\infty, \infty[$.

### 2.3 Representation of Actions

We translate action descriptions into this temporal language. The atomic propositions are $a_1, \ldots, a_n$ where $1, \ldots, n$ is some indexing of the $n = |A|$ actions in a transition system $\langle X, I, A \rangle$, and $x_1, \ldots, x_m$ for the $m = |X|$ state variables in $X$. Actions $(p, e)$ with index $i$ are formalized as follows.

$$a_i \rightarrow p \tag{1}$$

$$(\phi \wedge a_i) \rightarrow [t]l \text{ for all } (\phi \rhd l@t) \in e \tag{2}$$

Depending on the planning language used, there are *action exclusion* constraints preventing an action from being taken if some other action has been taken recently. Main forms of such constraints can be translated into formulas

$$a_i \rightarrow [t_0, t_1] \neg a_j \tag{3}$$

where $t_0$ and $t_1$ are rational numbers such that $t_0 \leq t_1$. The working of our invariant algorithm is independent of how these constraints are derived. Representative definitions of action exclusion can be found in literature [13,8,4].

*Frame axioms* indicate when a fact remains unchanged. For every $x \in X$ we have

$$x \rightarrow (x \mathcal{U} c) \tag{4}$$

where $c$ is the disjunction of all formulas $[-t](a_i \wedge \phi)$ such that $(\phi \rhd \neg x@t) \in e$ for the action $(p, e)$ with index $i$. There is an analogous axiom $\neg x \rightarrow (\neg x \mathcal{U} c^{\neg})$ indicating the conditions $c^{\neg}$ for change from true to false.

We denote the set of all formulas above by $\alpha_{X,A}$.

## 3 The Algorithm

Standard invariant algorithms [10,5,3,6,7,11,12] are not applicable in the timed setting where multiple actions can be taken concurrently. For classical planning, the basic induction step in invariant computation is determining, for a given action, which true facts remain true after the action has been taken. With timed models, this basic step must cover the possibility of other actions being taken concurrently. For example, two actions both with precondition $a$ and respectively effects $\neg a, b$ and $\neg a, c$ individually cannot falsify candidate invariant $\neg(b \wedge c)$, but taken simultaneously they will.

The inductive algorithm for deriving invariants for timed systems is given in Figure 1, with the subprocedure *weaken* explained later.

```
1:  PROCEDURE temporalinvariants(X, I, A);
2:  C := {x ∈ X | I ⊨ x} ∪ {¬x | x ∈ X, I ⊭ x};
3:  REPEAT
4:     C_old := C;
5:     FOR EACH a ∈ A and c ∈ C such that φ ▷ l@t is an effect of a and l̄ occurs in c DO
6:        S_{a,c} := the formula given in Lemma 1;
7:        IF S_{a,c} is consistent THEN
8:              C := C\{c};
9:              C := C ∪ weaken_{S_{a,c}}(c);
10: UNTIL C = C_old;
11: RETURN C;
```

**Fig. 1.** Algorithm for computing timed invariants

The induction follows the idea of constructing a schedule of temporal actions step by step. We consider the construction of such schedules in a specific form. Instead of

an inductive step that allows adding an arbitrary action in an arbitrary location of a schedule, we only add actions in the end of the schedule so that no other action is taken later, and no actions with a smaller index (according to an arbitrary ordering $<$) can be taken at the same last time point.

Assuming that all schedules of $i-1$ actions satisfy a certain set $C$ of candidate invariants, we consider schedules of $i$ actions to test which of the candidate invariants are still satisfied. The passage from $i-1$ to $i$ actions corresponds to adding an action $a \in A$ to a schedule with $i-1$ actions.

The base case of the induction is the execution with 0 actions with $I$ as the initial state at some unspecified time point and at all preceding and succeeding time points.

For the inductive cases, we can over-approximate executions with $i-1$ actions that satisfy candidate invariants $C_{old}$ with the following formula.

$$\begin{aligned} S = \; & \{\Box\phi \mid \phi \in \alpha_{X,A}\} \cup \\ & \{[-\infty, 0[\phi \mid \phi \in C_{old}\} \end{aligned}$$

This formula says that all changes during the execution correspond to some actions (as formalized by $\alpha_{X,A}$) and that all candidate invariants hold until 0.

For a given action $a$ and candidate invariant $c$ that could be falsified by an effect $\phi \rhd l@t$ we extend this set further. Now $a$ will be taken at time point 0 as the last action at (the arbitrarily chosen) time point 0, no actions are taken after 0, and no action with index smaller than $a$'s is taken at 0. Hence we have

$$\begin{aligned} S_{a,c} = \; & S \cup \{[0]a, [0]\phi, [t]\neg c\} \\ & \cup \{]0, \infty[\neg a' \mid a' \in A\} \\ & \cup \{[0]\neg a' \mid a' \in A, d(a') < d(a)\} \end{aligned}$$

for executions with some $i-1$ actions extended with the $i$th action $a$.

The important properties of $S_{a,c}$ are stated in the next lemma. Section 4 provides an efficient incomplete procedure for the consistency tests. Although we have fixed 0 to be the time point where $a$ is taken in $S_{a,c}$, this choice does not lose generality and the result holds for an arbitrary time points.

**Lemma 1.** *Let $\alpha_{X,A}$ be the translation of actions $A$ into temporal logic as given earlier, $a \in A$ an action with effect $\phi \rhd \overline{l}@t$, $C_{old}$ a set of candidate invariants, and $c$ a candidate invariant with occurrence of the literal $l$. Let*

$$\begin{aligned} S_{a,c} = \; & \{\Box\phi \mid \phi \in \alpha_{X,A}\} \\ & \cup \{[-\infty, 0[\phi \mid \phi \in C_{old}\} \\ & \cup \{[0]a, [0]\phi, [t]\neg c\} \\ & \cup \{]0, \infty[\neg a' \mid a' \in A\} \\ & \cup \{[0]\neg a' \mid a' \in A, d(a') < d(a)\}. \end{aligned}$$

*If $S_{a,c}$ is inconsistent, then there is no execution with actions from $A$ such that 1. action $a$ is taken at some time point $t'$, 2. formulas in $C_{old}$ are true until $t'$ (excluding $t'$), 3. no action is taken after $t'$, 4. no lower index action is taken at $t'$, and 5. action $a$ makes one of the literals in $c$ is false at $t' + t$.*

After an attempt to prove that a candidate invariant remains true has failed, it will be replaced by logically weaker candidate invariants. The new candidate invariants either add a new disjunct, or replace a disjunct $[t_0, t_1]l$ by a weaker one $[t'_0, t'_1]l$ such that $t_0 \leq t'_0 \leq t'_1 \leq t_1$ and either $t_0 < t'_0$ or $t'_1 < t_1$.

Our algorithm is general and is not limited to any particular form of (candidate) invariants. For performance reasons, our implementation (Section 5) limits to (candidate) invariants of forms $l$ and $l_1 \vee [-t, 0]l_2$, $t \geq 0$.

We define weaken$_{S_{a,c}}(\phi)$ as the set of all maximal weakenings of $\phi$ as read from the partial assignment that satisfied $S_{a,c}$ (see Section 4), with maximality defined in terms of inclusion of intervals $[t, t']l$.

**Lemma 2.** *weaken$_{S_{a,c}}(\phi) \not\models \phi$, and $\phi \models \phi'$ for all $\phi' \in$ weaken$_{S_{a,c}}(\phi)$.*

In the main loop of the algorithm, line 5 tests whether the effects of action $a$ mention a state variable occurring in a candidate invariant $c$. If not, $c$ cannot be possibly falsified by $a$. Otherwise, a more thorough test is performed in the form of the consistency test on line 7. If $S_{a,c}$ is inconsistent then $a$ cannot possibly falsify $c$. If $S_{a,c}$ is consistent, then it may be possible that $c$ is falsified by $a$, and $c$ has to be weakened or eliminated. This consistency test for the linear temporal logic is approximated as described in Section 4.

Only candidate invariants that pass all tests and cannot therefore be falsified by any action in any reachable state will remain in the set $C$ until the algorithm reaches a fixpoint. Therefore all such formulas are invariants. However, due to the approximate consistency test and syntactic restrictions on the form of the invariants, not all invariants are always found.

**Theorem 1.** *If the algorithm temporalinvariants($X, I, A$) returns $C$, then all formulas in $C$ are true everywhere in every execution of $\langle X, I, A\rangle$.*

*Proof.* The proof is by induction on $i$, the number of iterations of the outermost repeat-until loop on lines 3-10. The proof is based on identifying the number of iterations with the number of action occurrences in an execution. Let $C_i$ be the value of the variable $C$ in the beginning of each iteration of the loop (with iterations numbered as 0,1,...).

Induction hypothesis: $v \models_t c$ for every $t$ and every $c \in C_i$ and every execution $v$ of $\langle X, I, A\rangle$ with $i$ actions.

Base case $i = 0$: By construction, $C_0$ consists of formulas true in the initial state, and hence in all states that precede or follow, as no actions are taken anywhere.

Inductive case $i \geq 1$: For any $c \in C_i$ it must be that $S_{a,c}$ is inconsistent for every $a \in A$ (otherwise $c$ would have been eliminated between lines 7 and 9.)

Hence by Lemma 1 there is no execution with $i$ actions in which that "last" action would make $c$ false (with $C_{old}$ representing all executions with $i - 1$ actions.) Assume there is an execution $v$ with $i$ actions in which some other than the "last" action would make $c$ false. We could remove $a$ from this execution to obtain an execution with $i - 1$ actions that still falsifies $c$, and hence by induction hypothesis we would have $c \notin C_{i-1}$. Since $C_i$ is logically weaker than $C_{i-1}$ (due to formulas being replaced by strictly weaker ones (Lemma 2)), we could not have $c \in C_i$. Hence there is no execution with $i$ actions with $c$ false at some time point.

## 4 Approximate Consistency Tests

We now present an efficient and sound but incomplete approximation of temporal logic consistency based on constraint networks, as required in the consistency tests of $S_{a,c}$ in the preceding section. A constraint network is constructed for a set of temporal logic formulas. Every subformula is represented as a node in the constraint network and associated with two sets of intervals, one for *true* and another for *false*. Each rule infers intervals for a node given its neighbors. The neighbors are the immediate subformulas, the parent formula, and sibling formulas. The rules are of the form

$$\frac{(g_1 : i_1)\phi_1, \ldots, (g_n : i_n)\phi_n}{(g : i)\phi}$$

where $\phi_1, \ldots, \phi_n$ and $\phi$ are formulas respectively with tags $(g_j : i_j), j \in \{1, \ldots, n\}$ and $(g : i)$, where each $g_j$ is either $\top$ or $\bot$ to express truth or falsity, and $i_j$ is a set (union) of intervals where $\phi_j$ has the specified truth value. A rule is applied by first selecting a node for a formula of the form of the consequent $\phi$. Then the tags $(g_j : i_j)$ for $\phi_1, \ldots, \phi_n$ (in some rules only a single interval) are retrieved. Finally the tag $(g : i)$ for the consequent is computed and the new intervals $i$ added to the old intervals of $\phi$.

The rules for truth-functional connectives are obvious. Next we list some of the more interesting propagation rules. The rules for the interval operator are the following.

$$\frac{(\top : [t_0, t_0'])\phi}{(\top : [t_0 - t, t_0' - t'])[t, t']\phi} \quad (5) \qquad \frac{(\top : [t_0, t_0'])[t, t']\phi}{(\top : [t_0 + t, t_0' + t'])\phi} \quad (7)$$

$$\frac{(\bot : [t_0, t_0'])\phi}{(\bot : [t_0 - t', t_0' - t])[t, t']\phi} \quad (6)$$

In the first rule, the interval $[t_0 - t, t_0' - t']$ is empty if $t_0' - t' < t_0 - t$. In the first two rules, intervals with infinite end-points are handled specially, as subtraction of $\infty$ or $-\infty$ from itself is not well-defined. We define finite additions and subtractions to the infinities by $\infty + r = \infty$ and $-\infty + r = -\infty$. In rule (5), if both the interval starting point $t_0$ of $\phi$ and the operator starting point $t$ are $-\infty$, then the starting point for $[t, t']\phi$ is $-\infty$ as well. Similarly for end points $\infty$.

The rules for the *until* operator are the following.

$$\frac{(\bot : i_0)\phi_0, (\bot : i_1)\phi_1}{(\bot : i_0 \cap i_1)\phi_0 \mathcal{U} \phi_1} \quad (8)$$

$$\frac{(\top : [t, t'])\phi_0 \mathcal{U} \phi_1, (\bot : [t_1, t_1'])\phi_1}{(\top : [\max(t, t_1), t_1'])\phi_0} \text{ if } \begin{matrix}[t, t_9'] \cap \\ [t_1, t_1'] \neq \emptyset\end{matrix} \quad (9)$$

In all of the rules above we have used *closed* intervals only. The rules can be adapted to open and half-open intervals as well as to interval operators with such intervals.

Initially, all nodes are labelled with $(\top : \emptyset)$ and $(\bot : \emptyset)$ with $\top$ denoting *true* and $\bot$ *false*, and with the empty set of intervals $\emptyset$ indicating that no truth or falsity is known for any time interval. We detect a contradiction when $i_1 \cap i_2 \neq \emptyset$ for $(\top : i_1)\phi$ and $(\bot : i_2)\phi$ and some $\phi$, that is, $\phi$ has to be both true and false in at least one time point.

The constraint propagation procedure does not terminate for all formula sets. However, for all sets $S_{a,c}$ we have tried the procedure quickly terminates.

## 5 Experiments

We have experimented with the algorithm and problem instances featured in the temporal planning tracks of the 2008 and 2011 planning competitions (IPC), modeled in timed PDDL. Table 1 summarizes runtimes and other statistics.

**Table 1.** Statistics for a number of IPC domains. We give the runtimes (in seconds) for the easiest and hardest instance in each domain, the number of instances for which the computation terminated in under 10 minutes, the numbers of invariants found, and the numbers of actions.

| | runtime | | | invariants | | actions | |
|---|---|---|---|---|---|---|---|
| problem | min. | max. | $< 600$ s | min. | max. | min. | max. |
| *crewplanning* | 1.33 | 83.34 | 30/30 | 225 | 2925 | 27 | 1393 |
| *elevators* | 7.85 | $> 14400$ | 25/30 | 740 | $\geq 23934$ | 1672 | 141384 |
| *elevators/numeric* | 2.79 | 7311.33 | 22/30 | 704 | $\geq 31620$ | 448 | 10734 |
| *openstacks/adl* | 1.45 | 283.16 | 30/30 | 86 | 1508 | 45 | 2074 |
| *openstacks/numeric* | 0.64 | 5.13 | 30/30 | 104 | 960 | 15 | 102 |
| *openstacks/numeric/adl* | 0.68 | 7.16 | 30/30 | 66 | 638 | 15 | 102 |
| *openstacks/strips* | 1.52 | 384.80 | 30/30 | 124 | 1830 | 45 | 2074 |
| *parcprinter* | 2.57 | $> 14400$ | 24/30 | 656 | $\geq 17530$ | 61 | 3979 |
| *pegsol* | 0.62 | 3.61 | 30/30 | 40 | 986 | 76 | 76 |
| *sokoban* | 5.40 | $> 14400$ | 19/30 | 1682 | $\geq 40274$ | 280 | 28240 |
| *transport/numeric* | 1.08 | $> 14400$ | 16/30 | 124 | $\geq 57240$ | 66 | 22869 |
| *floortile* | 2.10 | 19.33 | 20/20 | 264 | 2610 | 148 | 606 |
| *matchcellar* | 0.43 | 1.03 | 10/10 | 2 | 10 | 3 | 210 |
| *parking* | 3.94 | 392.60 | 20/20 | 653 | 6991 | 726 | 7406 |
| *storage* | 27.96 | $> 14400$ | 10/20 | 2430 | $\geq 18684$ | 3400 | 130480 |
| *tms* | 1.79 | $> 14400$ | 5/20 | 774 | $\geq 5664$ | 133 | 12141 |
| *turnandopen* | 4.36 | 805.85 | 18/20 | 704 | 10354 | 464 | 12216 |

## 6 Conclusion

We have, for a first time, presented a general algorithm for computing a large class of invariants for timed systems, with the analysis of such systems and speeding up reasoning with them as the main applications. Earlier works on timed invariants in planning have limited to narrow classes of invariants or narrow classes of timed models [13,1]. Our framework is applicable to a wide range of timed models and forms of invariants.

Similarly to the strongest earlier algorithms for untimed or asynchronous systems, our algorithm is based on a fixpoint iteration which starts from a set of candidate invariants characterizing the initial state, and weakens this set to cover all reachable states of the system. Iteration $N$ of the algorithm corresponds to reachability by schedules of $N$ timed actions, with the fixpoint corresponding to schedules with any number of actions.

Due to the generality of our algorithm, its scalability for large action sets and high number of state variables is not as good as with simpler algorithms. Future work will focus on finding interesting performance vs. generality trade-offs.

## References


1. Bernardini, S., Smith, D.E.: Automatic synthesis of temporal invariants. In: Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation, SARA 2011, Parador de Cardona, Cardona, Catalonia, Spain, July 17-18. AAAI Press (2011)
2. Burch, J.R., Clarke, E.M., Long, D.E., MacMillan, K.L., Dill, D.L.: Symbolic model checking for sequential circuit verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13(4), 401–424 (1994)
3. Edelkamp, S., Helmert, M.: Exhibiting knowledge in planning problems to minimize state encoding length. In: Biundo, S., Fox, M. (eds.) ECP 1999. LNCS (LNAI), vol. 1809, pp. 135–147. Springer, Heidelberg (2000)
4. Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research 20, 61–124 (2003)
5. Gerevini, A., Schubert, L.: Inferring state constraints for domain-independent planning. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998) and the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI 1998), pp. 905–912. AAAI Press (1998)
6. Gerevini, A., Schubert, L.K.: Discovering state constraints in DISCOPLAN: Some new results. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000) and the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI 2000), pp. 761–767. AAAI Press (2000)
7. Haslum, P., Geffner, H.: Admissible heuristics for optimal planning. In: Chien, S., Kambhampati, S., Knoblock, C.A. (eds.) Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, pp. 140–149. AAAI Press (2000)
8. Haslum, P., Geffner, H.: Heuristic planning with time and resources. In: Cesta, A. (ed.) Recent Advances in AI Planning, Sixth European Conference on Planning (ECP 2014), pp. 107–112. AAAI Press (2014)
9. Kautz, H., Selman, B.: Pushing the envelope: planning, propositional logic, and stochastic search. In: Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference, pp. 1194–1201. AAAI Press (1996)
10. Rintanen, J.: A planning algorithm not based on directional search. In: Cohn, A.G., Schubert, L.K., Shapiro, S.C. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR 1998), pp. 617–624. Morgan Kaufmann (1998)
11. Rintanen, J.: An iterative algorithm for synthesizing invariants. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000) and the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-2000), pp. 806–811. AAAI Press (2000)
12. Rintanen, J.: Regression for classical and nondeterministic planning. In: Ghallab, M., Spyropoulos, C.D., Fakotakis, N. (eds.) ECAI 2008: Proceedings of the 18th European Conference on Artificial Intelligence, pp. 568–571. IOS Press (2008)
13. Smith, D.E., Weld, D.S.: Temporal planning with mutual exclusion reasoning. In: Dean, T. (ed.) Proceedings of the 16th International Joint Conference on Artificial Intelligence, pp. 326–337. Morgan Kaufmann Publishers (1999)