Int J Softw Tools Technol Transfer
DOI 10.1007/s10009-014-0359-5

SPIN 2013

Efficient model-checking of weighted CTL with upper-bound constraints

Jonas Finnemann Jensen · Kim Guldstrand Larsen · Jiří Srba · Lars Kaerlund Oestergaard

© Springer-Verlag Berlin Heidelberg 2014

Abstract We present a symbolic extension of dependency graphs by Liu and Smolka in to model-check weighted Kripke structures against the computation tree logic with upper-bound weight constraints. Our extension introduces a new type of edges into dependency graphs and lifts the computation of fixed-points from boolean domain to non-negative integers to cope with the weights. We present both global and local algorithms for the fixed-point computation on symbolic dependency graphs and argue for the advantages of our approach compared to the direct encoding of the model-checking problem into dependency graphs. We implement all algorithms in a publicly available tool and evaluate them on several experiments. The principal conclusion is that our local algorithm is the most efficient one with an order of magnitude improvement for model checking problems with a high number of "witnesses".

Keywords Weighted CTL · Weighted automata · Model checking · On-the-fly methods

1 Introduction

Model-driven development is finding its way into industrial practise within the area of embedded systems. Here a key challenge is how to handle the growing complexity of systems, while meeting requirements on correctness, predictability, performance and not least time- and cost-to-market. In this respect model-driven development is seen as a valuable and promising approach, as it allows early design-space exploration and verification and may be used as the basis for systematic and unambiguous testing of a final product. However, for embedded systems, verification should not only address functional properties but also a number of non-functional properties related to timing and resource constraints.

Within the area of model-checking, a number of state-machine-based modeling formalisms have emerged, allowing for such quantitative aspects to be expressed. In particular, timed automata (TA) [2], and the extensions to weighted timed automata (WTA) [4,8] are popular and tool-supported formalisms that allow for such constraints to be modeled.

Interesting properties of TA and WTA may be expressed in natural weight-extended versions of classical temporal logics such as computation tree logic (CTL) for branching-time and LTL for linear-time. Just as TCTL and MTL provide extensions of CTL and LTL with time-constrained modalities, the logics weighted CTL (WCTL) and WMTL are extensions with weight-constrained modalities interpreted with respect to WTA. Unfortunately, the addition of weight now turns out to come with a price: whereas the model-checking problems for TA with respect to TCTL [1] and MTL [3] are decidable, it has been shown that model-checking WTA with respect to WCTL is undecidable [10].

In this paper, we reconsider this model-checking problem in the setting of *untimed* models, i.e. essentially weighted Kripke structures, and negation-free WCTL formulae with only upper bound constraints on weights. As main contributions, we show that in this setting the model-checking problem is in PTIME, and we provide an efficient symbolic, local (on-the-fly) model-checking algorithm.

J. F. Jensen · K. G. Larsen · J. Srba (✉) · L. K. Oestergaard
Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, 9220 Aalborg, Denmark
e-mail: srba@cs.aau.dk

J. F. Jensen
e-mail: jopsen@gmail.com

K. G. Larsen
e-mail: kgl@cs.aau.dk

L. K. Oestergaard
e-mail: larsko@gmail.com

Published online: 20 November 2014

Our results are based on a novel symbolic extension of the dependency graph framework of Liu and Smolka [20] where they encode boolean equation systems and offer global (classical iterative fixed-point computation by repeated functor application) and local (on-the-fly fixed-point computation by guided search) algorithms for computing minimal and maximal fixed points in linear time. Whereas a direct encoding of our model-checking problem into dependency graphs leads only to a pseudo-polynomial algorithm,[1] the novel symbolic dependency graphs allow for a polynomial encoding and a polynomial time fixed-point computation. Most importantly, the symbolic dependency graph encoding enables us to perform a symbolic local fixed-point evaluation. Experiments with the various approaches (direct versus symbolic encoding, global versus local algorithm) have been conducted on a large number of cases, demonstrating that the combined symbolic and local approach is the most efficient one. For model-checking problems with affirmative outcome, this combination is often one order of magnitude faster than the other approaches.

Related work The present paper is situated in the area of weighted automata [14,15] where weights come, in general, from the domain of abstract algebraic structures. Our work focuses only on the concrete domain of nonnegative integers that can be tested against given upper-bounds but on the other hand, we show that we can design efficient algorithms for solving model-checking problems in this case.

Laroussinie et al. [18] consider the problem of model-checking durational concurrent game structures with respect to timed ATL properties, offering a PTIME result in the case of non-punctual constraints in the formula. Restricting the game structures to a single player gives a setting similar to ours, as timed ATL is essentially WCTL. However, in contrast to [18], we do allow transitions with zero weight in the model, making a fixed-point computation necessary. As a result, the corresponding CTL model-checking (with no weight constraints) is a special instance of our approach, which is not the case for [18]. Most importantly, the work in [18] does not provide any local algorithm, which our experiments show is crucial for the performance. No implementation is provided in [18].

Buchholz and Kemper [11] propose a valued computation tree logic (CTL$) interpreted over a general set of weighted automata that includes CTL in the logic as a special case over the boolean semiring. For model-checking CTL$ formulae they describe a matrix-based algorithm. Their logic is more expressive than the one proposed here, since they support negation and all the comparison operators. In addition, they permit nested CTL formulae and can operate on max/plus semirings in $O(\min(\log(t) \cdot mm, t \cdot nz))$ time, where t is the number of vector matrix products, mm is the complexity of multiplying two matrices of order n and nz is the number of non-zero elements in special matrix used for checking "until" formulae up to some bound t. However, they do not provide any on-the-fly technique for verification.

Another related work by Bouyer et al. [9] shows that model-checking with respect to WCTL is PSPACE-complete for one-clock WTA and for TCTL (the only cost variable is the time elapsed).

Several approaches to on-the-fly/local algorithms for model-checking the modal mu-calculus have been proposed. Andersen [5] describes a local algorithm for model-checking the modal mu-calculus for alternation depth one running in $O(n \cdot \log(n))$ where n is the product of the size of the assertion and the labeled transition system. Liu and Smolka [20] improve on the complexity of this approach with a local algorithm running in $O(n)$ (where n is the size of the input graph) for evaluating alternation-free fixed points. This is also the algorithm that we apply for WCTL model-checking and the one we extend for symbolic dependency graphs. Cassez et. al. [12] present another symbolic extension of the algorithm by Liu and Smolka; a zone-based forward, local algorithm for solving timed reachability games. Later Liu et al. [19] also introduce a local algorithm for the evaluation of alternating fixed points with the complexity $O(n + (\frac{n+ad}{ad})^{ad})$, where ad is the alternation depth of the graph. We do not consider the evaluation of alternating fixed points in the weighted setting as this is left for future work.

Outline. Weighted Kripke structures (WKS) and WCTL are presented in Sect. 2. Section 3 then introduces dependency graphs. Model-checking WCTL with this framework is discussed in Sect. 4. In Sect. 5 we propose symbolic dependency graphs and demonstrate how they can be used for WCTL model-checking in Sect. 6. Experimental results are presented in Sects. 7 and 8 concludes the paper.

2 Basic definitions

We shall first introduce the notion of a WKS and WCTL. Let $\mathbb{N}_0$ be the set of nonnegative integers. A *WKS* is a quadruple $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, where S is a finite set of states, $\mathcal{AP}$ is a finite set of atomic propositions, $L\colon S \rightarrow \mathcal{P}(\mathcal{AP})$ is a mapping from states to sets of atomic propositions, and $\rightarrow \subseteq S \times \mathbb{N}_0 \times S$ is the transition relation.

Instead of $(s, w, s') \in \rightarrow$, meaning that from the state s under the weight w we can move to the state s', we often write $s \xrightarrow{w} s'$. A WKS is *nonblocking* if for every $s \in S$ there is an s' such that $s \xrightarrow{w} s'$ for some weight w. From now on we consider only nonblocking WKS.[2]

[1] Exponential in the encoding of the weights in the model and the formula.

[2] A blocking WKS can be turned into a nonblocking one by introducing a new state with no atomic propositions, zero-weight

$s \models \mathbf{true}$	
$s \models a$	if $a \in L(s)$
$s \models \varphi_1 \wedge \varphi_2$	if $s \models \varphi_1$ and $s \models \varphi_2$
$s \models \varphi_1 \vee \varphi_2$	if $s \models \varphi_1$ or $s \models \varphi_2$
$s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$	if there exists a run σ starting from s and a position $p \geq 0$ such that $\sigma(p) \models \varphi_2, W_\sigma(p) \leq k$ and $\sigma(p') \models \varphi_1$ for all $p' < p$
$s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$	if for any run σ starting from s, there is a position $p \geq 0$ such that $\sigma(p) \models \varphi_2, W_\sigma(p) \leq k$ and $\sigma(p') \models \varphi_1$ for all $p' < p$
$s \models EX_{\leq k}\ \varphi$	if there is a state s' such that $s \xrightarrow{w} s', s' \models \varphi$ and $w \leq k$
$s \models AX_{\leq k}\ \varphi$	if for all states s' such that $s \xrightarrow{w} s'$ where $w \leq k$ holds that $s' \models \varphi$

Fig. 1 Semantics of WCTL

A *run* in an WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ is an infinite computation

$$\sigma = s_0 \xrightarrow{w_0} s_1 \xrightarrow{w_1} s_2 \xrightarrow{w_2} s_3 \ldots$$

where $s_i \in S$ and $(s_i, w_i, s_{i+1}) \in \rightarrow$ for all $i \geq 0$. Given a *position* $p \in \mathbb{N}_0$ in the run σ, let $\sigma(p) = s_p$. The *accumulated weight* of σ at position $p \in \mathbb{N}_0$ is then defined as $W_\sigma(p) = \Sigma_{i=0}^{p-1} w_i$ where by definition $W_\sigma(0) = 0$.

We can now define negation-free WCTL with weight upper-bounds. The set of WCTL formulae over the set of atomic propositions $\mathcal{AP}$ is given by the abstract syntax

$$\varphi ::= \mathbf{true} \mid \mathbf{false} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid E\ \varphi_1\ U_{\leq k}\ \varphi_2 \mid A\ \varphi_1\ U_{\leq k}\ \varphi_2 \mid EX_{\leq k}\ \varphi \mid AX_{\leq k}\ \varphi$$

where $k \in \mathbb{N}_0 \cup \{\infty\}$ and $a \in \mathcal{AP}$. We assume that the element ∞ is larger than any other natural number and that $\infty + k = \infty - k = \infty$ for all $k \in \mathbb{N}_0$. In Fig. 1 we now inductively define the satisfaction triple $s \models \varphi$, meaning that a state s satisfies a formula φ.

Example 1 Consider the WKS in Fig. 2 with seven states annotated with the propositions *mow* and *dump*. It represents a grass field with different routes a lawn mower can take from the starting position s_0 to the state s_6 where the grass can be dumped. The weights on the different transitions represent the number of units of grass that is accumulated in the grass container when selecting a particular route. The lawn mower breaks if we try to store more than six units of grass in its container. The question is whether the grass is always dumped before the lawn mower breaks, irrelevant of the selected route. This is expressed by the formula $A\ mow\ U_{\leq 6}\ dump$ that is satisfied in the initial state s_0

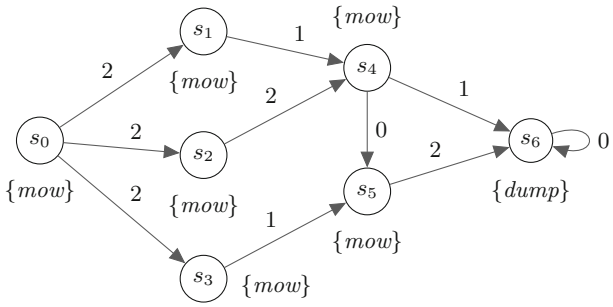


Fig. 2 A lawn mower example, $s_0 \models A\ mow\ U_{\leq 6}\ dump$

as even the most expensive path from s_0 to s_6 (through the states s_2, s_4, and s_5) accumulates only six units of grass.

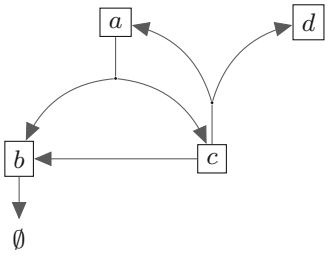
3 Dependency graphs

In this section we present the dependency graph framework and a local algorithm for minimal fixed-point computation as originally introduced by Liu and Smolka [20]. This framework can be applied to model-checking of the alternation-free modal mu-calculus, including its sublogics like CTL. Later, in Sect. 4, we demonstrate how to extend the framework from CTL to WCTL and provide an encoding of the WCTL model-checking problem into dependency graphs.

Definition 1 [*Dependency graph (DG)*] A *dependency graph* is a pair $G = (V, E)$ where V is a finite set of *configurations*, and $E \subseteq V \times \mathcal{P}(V)$ is a finite set of *hyper-edges*.

Let $G = (V, E)$ be a dependency graph. For a hyper-edge $e = (v, T)$, we call v the source configuration and T the target (configuration) set of e. For a configuration v, the set of its edge-successors is given by $succ(v) = \{(v, T) \in E\}$. The size of G is $|G| = |V| + |E|$ where $|V|$ and $|E|$ is the size of these components in a binary representation (note that the size of a hyper-edge depends on the number of nodes it connects to).

footnote2 continued
self-loop and with zero-weight transitions from all blocking states into this newly introduced state.



	a	b	c	d
A_0	0	0	0	0
$F(A_0)$	0	1	0	0
$F^2(A_0)$	0	1	1	0
$F^3(A_0)$	1	1	1	0
$F^4(A_0)$	1	1	1	0

Fig. 3 Four iterations of the global algorithm

An *assignment* $A\colon V \to \{0, 1\}$ is a function that assigns boolean values to configurations of G. A *prefixed-point assignment* of G is an assignment A where, for every configuration $v \in V$, holds that if $(v, T) \in E$ and $A(u) = 1$ for all $u \in T$ then also $A(v) = 1$.

By taking the standard component-wise ordering $\sqsubseteq$ on assignments, where $A \sqsubseteq A'$ if and only if $A(v) \leq A'(v)$ for all $v \in V$ (assuming that $0 < 1$), we get by the Knaster [16] and Tarski [23] fixed-point theorem that there exists a unique minimum prefixed-point assignment, denoted by $A_{\min}$. The minimum prefixed-point assignment $A_{\min}$ of G can be computed by repeated applications of the monotonic function F from assignments to assignments, starting from A_0 where $A_0(v) = 0$ for all $v \in V$, and where

$$F(A)(v) = \bigvee_{(v,T)\in E} \left(\bigwedge_{u\in T} A(u) \right)$$

for all $v \in V$. Here we implicitly assume that a disjunction over an empty set equals to 0 and a conjunction over an empty set equals to 1. We are guaranteed to reach a fixed point after a finite number of applications of F due to the finiteness of the complete lattice of assignments ordered by $\sqsubseteq$. Hence there exists an $m \in \mathbb{N}_0$ such that $F^m(A_0) = F^{m+1}(A_0)$, in which case we have $F^m(A_0) = A_{\min}$. We will refer to this iterative algorithm as the *global* one.

Example 2 Figure 3 shows a dependency graph where configurations are illustrated as labeled squares and hyper-edges are drawn as a span of lines to every configuration in the respective target set. The corresponding function F is given as a boolean equation system. These equations represent naturally F such that in a given assignment, we evaluate their right-hand sides and the new assignment then updates the valuation of the left-hand side configurations. We also depict four iterations of the global algorithm (sufficient to compute the minimum prefixed-point assignment).

Algorithm 1: Liu–Smolka local algorithm

Input: Dependency graph $G = (V, E)$ and a configuration $v_0 \in V$.
Output: Minimum prefixed-point assignment $A_{\min}(v_0)$ for v_0.

```
1  Let A(v) = ⊥ for all v ∈ V
2  A(v0) = 0; D(v0) = ∅
3  W = succ(v0)
4  while W ≠ ∅ do
5  |  let e = (v, T) ∈ W
6  |  W = W \ {e}
7  |  if A(u) = 1 for all u ∈ T then
8  |  |  A(v) = 1; W = W ∪ D(v)
9  |
10 |  else if there is u ∈ T such that A(u) = 0 then
11 |  |  D(u) = D(u) ∪ {e}
12 |  else if there is u ∈ T such that A(u) = ⊥ then
13 |  |  A(u) = 0; D(u) = {e}; W = W ∪ succ(u)
14 return A(v0)
```

In model-checking, we are often only interested in the minimum prefixed point assignment $A_{\min}(v)$ for a specific configuration $v \in V$. For this purpose, Liu and Smolka [20] suggest a local algorithm presented with minor modifications[3] in Algorithm 1. The algorithm maintains three data-structures throughout its execution: an assignment A, a dependency set D for every configuration and a set of hyper-edges W. The dependency set $D(v)$ for a configuration v maintains a list of hyper-edges that were processed under the assumption that $A(v) = 0$. Whenever the value of $A(v)$ changes to 1, the hyper-edges from $D(v)$ must be reprocessed to propagate this change to the respective sources of the hyper-edges.

Theorem 1 (Correctness of local algorithm [20]) *Given a dependency graph $G = (V, E)$ and a configuration $v_0 \in V$, Algorithm 1 computes the minimum prefixed point assignment $A_{\min}(v_0)$ for the configuration v_0.*

As argued in [20], both the local and global model-checking algorithms run in linear time.

4 WCTL and dependency graphs

In this section, we suggest a reduction from the model-checking problem of WCTL (on WKS) to the computation of the minimum prefixed-point assignment on a dependency graph.

Given a WKS $\mathcal{K}$, a state s of $\mathcal{K}$, and a WCTL formula φ, we construct a dependency graph where every configuration is a pair of a state and a formula. Starting from the initial pair

[3] At line 12 we added the assignment $D(u) = \{e\}$; the original algorithm sets the dependency set to empty here, leading to an incorrect propagation.

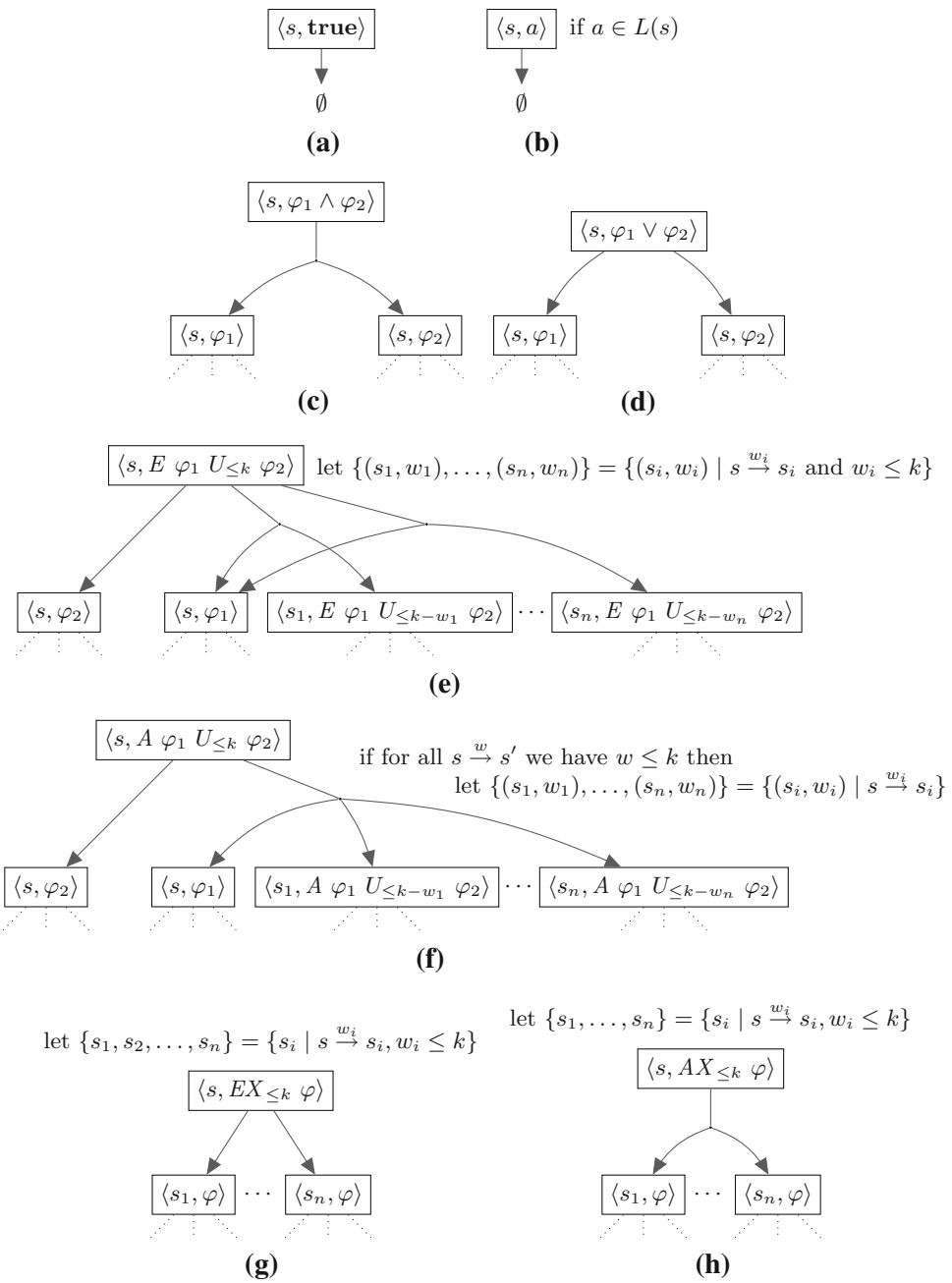


Fig. 4 Dependency graph encoding of state-formula pairs. **a** True. **b** Proposition. **c** Conjunction. **d** Disjunction. **e** Existential until. **f** Universal until. **g** Existential next. **h** Universal next

$\langle s, \varphi \rangle$, the dependency graph is constructed according to the rules given in Fig. 4. Note that in Fig. 4f the hyper-edge with several targets is present only if all outgoing transitions from s have weight at most k.

An example of the constructed dependency graph for our lawn mower example from Fig. 2 with the initial state s_0 and the formula A *mow* $U_{\leq 6}$ *dump* is given in Fig. 5.

The next theorem relates the validity of a WCTL formula in a given state with the minimum prefixed-point computation in the corresponding dependency graph.

Theorem 2 (Correctness of the encoding) *Let* $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ *be a WKS,* $s \in S$ *a state, and* φ *a WCTL formula. Let* G *be the constructed dependency graph rooted with* $\langle s, \varphi \rangle$. *Then* $A_{\min}(\langle s, \varphi \rangle) = 1$ *if and only if* $s \models \varphi$.

Proof By structural induction on φ.

(I) Let $\varphi = \textbf{true}$. Clearly $s \models \textbf{true}$. At the same time $A_{\min}(\langle s, \textbf{true} \rangle) = 1$ as in Fig. 4a we have a hyper-edge from the configuration $\langle s, \textbf{true} \rangle$ to the empty target set, implying that $A(v) = 1$ for any prefixed-point assignment A of G.

(II) Let $\varphi = a$. We prove that $A_{\min}(\langle s, a \rangle) = 1$ if and only if $s \models a$. If $a \in L(s)$, meaning that $s \models a$, then by

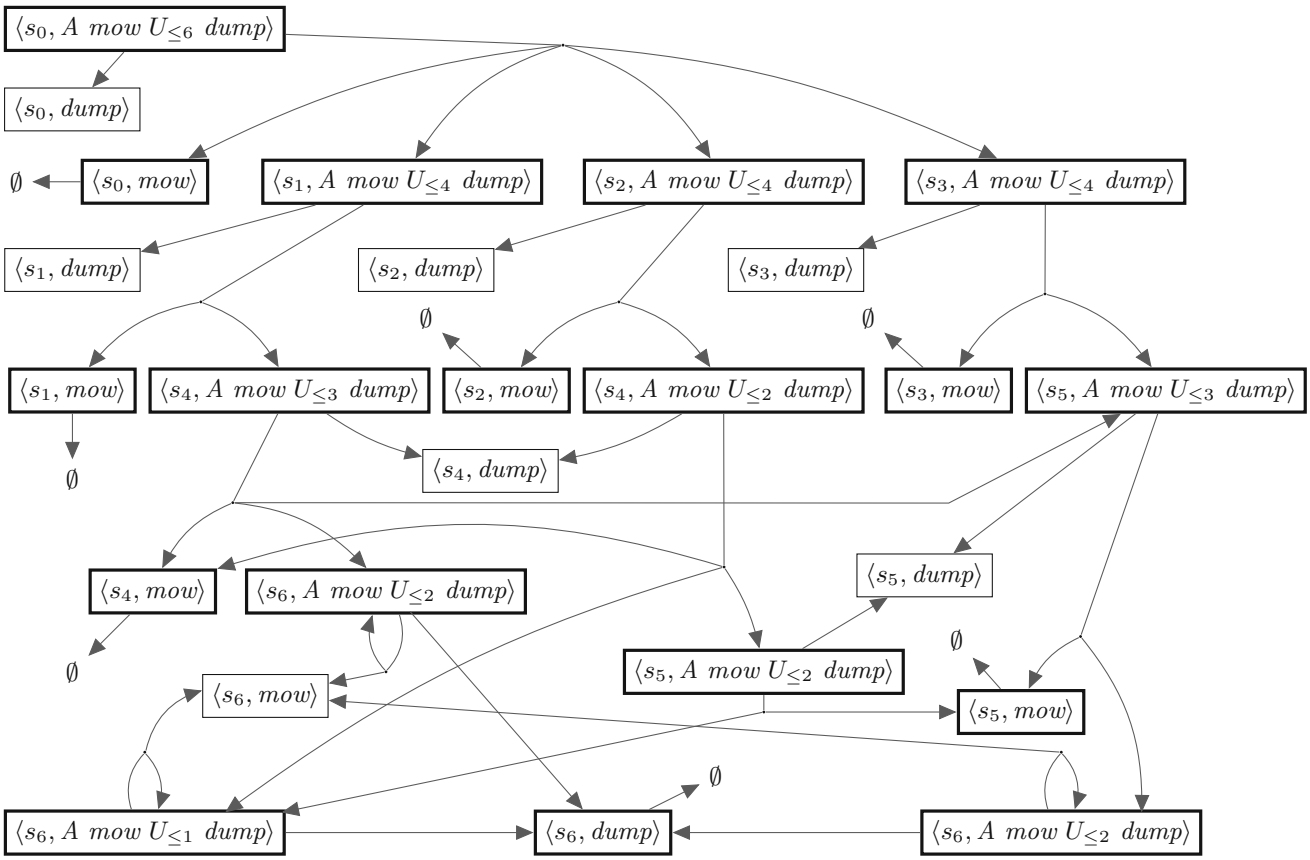


Fig. 5 Dependency graph for the lawn mower example (thick configurations have the value 1 in $A_{\min}$)

Fig. 4b there is a hyper-edge from the configuration $\langle s, a\rangle$ to the empty target set. As in (I) this means that $A_{\min}(\langle s, a\rangle) = 1$. If $a \notin L(s)$, meaning that $s \not\models a$, then by the side-condition in Fig. 4b we can conclude that there is no hyper-edge from the configuration $\langle s, a\rangle$ and hence $A_{\min}(\langle s, a\rangle) = 0$.

(III) Let $\varphi = \varphi_1 \wedge \varphi_2$. We show that $A_{\min}(\langle s, \varphi_1 \wedge \varphi_2\rangle) = 1$ if and only if $s \models \varphi_1 \wedge \varphi_2$. By Fig. 4c, a configuration $\langle s, \varphi_1 \wedge \varphi_2\rangle$ has a single hyper-edge with the target set $\{\langle s, \varphi_1\rangle, \langle s, \varphi_2\rangle\}$. Hence it is easy to see that $A_{\min}(\langle s, \varphi_1 \wedge \varphi_2\rangle) = 1$ if and only if $A_{\min}(\langle s, \varphi_1\rangle) = 1$ and $A_{\min}(\langle s, \varphi_2\rangle) = 1$, which is by induction hypothesis equivalent to $s \models \varphi_1$ and $s \models \varphi_2$, implying that $s \models \varphi_1 \wedge \varphi_2$ and vice versa.

(IV) Let $\varphi = \varphi_1 \vee \varphi_2$. We show that $A_{\min}(\langle s, \varphi_1 \vee \varphi_2\rangle) = 1$ if and only if $s \models \varphi_1 \vee \varphi_2$. By Fig. 4d, a configuration $\langle s, \varphi_1 \vee \varphi_2\rangle$ has two hyper-edges with the target sets $\{\langle s, \varphi_1\rangle\}$ and $\{\langle s, \varphi_2\rangle\}$. Hence $A_{\min}(\langle s, \varphi_1 \vee \varphi_2\rangle) = 1$ if and only if $A_{\min}(\langle s, \varphi_1\rangle) = 1$ or $A_{\min}(\langle s, \varphi_2\rangle) = 1$, which is by induction hypothesis equivalent to $s \models \varphi_1$ or $s \models \varphi_2$, implying that $s \models \varphi_1 \vee \varphi_2$ and vice versa.

(V) Let $\varphi = E\ \varphi_1\ U_{\leq k}\ \varphi_2$. We show that we have $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ if and only if $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$.

$\Leftarrow$: Let us first argue that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ implies $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$. Let $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$, meaning that there is a run σ starting from s and a position $p \geq 0$ such that

$$\sigma(p) \models \varphi_2 \tag{1}$$
$$\sigma(p') \models \varphi_1 \quad \text{for all } p' < p, \text{ and} \tag{2}$$
$$W_\sigma(p) \leq k. \tag{3}$$

We assume that the position p is the smallest such number satisfying the three conditions above. The claim is now proved by another (nested) mathematical induction on p. If $p = 0$, meaning that $s \models \varphi_2$, we know by induction hypothesis (from the structural induction) that $A_{\min}(\langle s, \varphi_2\rangle) = 1$ and this implies that also $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ by the left-most edge in Fig. 4e. If $p > 0$ then $s = \sigma(0) \xrightarrow{w} \sigma(1)$, where $w \leq k$, such that $s \models \varphi_1$ and $\sigma(1) \models E\ \varphi_1\ U_{\leq k-w}\ \varphi_2$. By structural induction hypothesis we know that $A_{\min}(\langle s, \varphi_1\rangle) = 1$ and by the mathematical induction hypothesis we get $A_{\min}(\langle \sigma(1), E\ \varphi_1\ U_{\leq k-w}\ \varphi_2\rangle) = 1$. As the rule in Fig. 4e contains a hyper-edge with the targets $\langle s, \varphi_1\rangle$ and $\langle \sigma(1), E\ \varphi_1\ U_{\leq k-w}\ \varphi_2\rangle$, necessarily also $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$.

$\Rightarrow$: We shall argue that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ implies the fact that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$. Let $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$. By Fig. 4e this means that either (i) $A_{\min}(\langle s, \varphi_2\rangle) = 1$, or that (ii) $A_{\min}(\langle s, \varphi_1\rangle) = 1$ and at the same time also $A_{\min}(\langle s_i, E\ \varphi_1\ U_{\leq k-w_i}\ \varphi_2\rangle) = 1$ for some s_i where $k - w_i \geq 0$. In case (i), $A_{\min}(\langle s, \varphi_2\rangle) = 1$ implies by the induction hypothesis that $s \models \varphi_2$, meaning that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ according to the WCTL semantics. In case (ii), by induction hypothesis we know that $s \models \varphi_1$ and we can repeat the same argument as before for the configuration $\langle s_i, E\ \varphi_1\ U_{\leq k-w_i}\ \varphi_2\rangle$ where $s \xrightarrow{w_i} s'$. If at some point the formula φ_2 gets satisfied in a state s' due to $A_{\min}(\langle s', \varphi_2\rangle) = 1$, then clearly there is a run leading from s to s', satisfying φ_1 in all intermediate states until s' is reached and the accumulated weight does not exceed k. This implies that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$. Observe now, that this is always the case, as otherwise if all the successors of $\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle$ in the dependency graph that get the value 1 in $A_{\min}$ never have a hyper-edge to a configuration of the form $\langle s', \varphi_2\rangle$ where $A_{\min}(\langle s', \varphi_2\rangle) = 1$, we necessarily get only cyclic dependencies among the hyper-edges, meaning that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 0$. This contradicts our assumption.

(VI) Let $\varphi = A\ \varphi_1\ U_{\leq k}\ \varphi_2$. We show that we have $A_{\min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ if and only if $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$.
$\Leftarrow$: We first argue that $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$ implies $A_{\min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$. Let $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$, meaning that for any run σ starting in s there is a position $p \geq 0$ satisfying the conditions (1), (2) and (3) listed in the proof of the existential until. Let p be the maximum position where φ_2 holds over all such runs. Clearly p is finite, as otherwise there must be a loop in the given WKS reachable from s such that φ_2 never holds on the loop, contradicting our assumption $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$. The claim is now proved by a nested mathematical induction on p.
If $p = 0$ then $s = \sigma(0) \models \varphi_2$ and by the structural induction hypothesis $A_{\min}(\langle s, \varphi_2\rangle) = 1$. The left-most edge in Fig. 4f guarantees that also $A_{\min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$. If $p > 0$ then for every run σ starting from s, the first transition $s = \sigma(0) \xrightarrow{w} \sigma(1)$ satisfies that $w \leq k$ and so there is present the right-most hyper-edge as depicted in Fig. 4f. Clearly $s \models \varphi_1$ and $\sigma(1) \models A\ \varphi_1\ U_{\leq k-w}\ \varphi_2$ in strictly less than p steps, so we can use the mathematical induction hypothesis to argue that all the target configurations of the hyper-edge get the value 1 in the minimum prefixed-point assignment. This implies the fact that $A_{\min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$.
$\Rightarrow$: We show that $A_{\min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ implies the fact that $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$. The arguments follow the same reasoning as in the analogous case for the existential until.

(VII) Let $\varphi = EX_{\leq k}\ \varphi$. It is easy to argument that $A_{\min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$ if and only if $s \models EX_{\leq k}\ \varphi$. By the WCTL semantics $s \models EX_{\leq k}\ \varphi$ iff there is a transition $s \xrightarrow{w} s'$ such that $w \leq k$ and $s' \models \varphi$. By induction hypothesis $s' \models \varphi$ iff $A_{\min}(\langle s', \varphi\rangle) = 1$. This is by the rule in Fig. 4g a necessary and sufficient condition for $A_{\min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$.

(VIII) Let $\varphi = AX_{\leq k}\ \varphi$. The argument is here analogous to the existential case. Note that the hyper-edge in Fig. 4h consists only of the successors reachable by a transition with weight at most k. This corresponds exactly with the definition of semantics of $AX_{\leq k}\ \varphi$ given in Fig. 1. □

To profit from the local algorithm by Liu and Smolka [20] presented in the previous section, we construct the dependency graph for a given state and a WCTL formula in an on-the-fly manner, generating successor configurations only when requested by the algorithm. This on-the-fly exploration often gives us a more efficient model-checking algorithm compared to the global approach. This is documented by the experiments summarized in Sect. 7.

However, the main drawback of the presented approach is that we may need to construct exponentially large dependency graphs. This is demonstrated in Fig. 6 where a single-state WKS on the left gives rise to a large dependency graph on the right where its size depends on the bound in the formula. In the figure the node $\langle s, E\ a\ U_{\leq k}\ b\rangle$ is repeated for all k, $0 \leq k \leq 1{,}000$, giving us an exponentially large encoding compared to the size of the constants in the formula (stored in binary). Hence the so far described method gives us only a pseudo-polynomial algorithm for model-checking WCTL.

5 Symbolic dependency graphs

We have seen in the previous section that the use of dependency graphs for WCTL model-checking suffers from an exponential explosion due to the unfolding of the until operators. We can, however, observe that the validity of $s \models E\ a\ U_{\leq k}\ b$ implies $s \models E\ a\ U_{\leq k+1}\ b$. In what follows we suggest a novel extension of dependency graphs, called *symbolic dependency graphs*, that use the implication above to reduce the size of the constructed graphs. Then in Sect. 6 we use symbolic dependency graphs for efficient (polynomial time) model-checking of WCTL.

Definition 2 [*Symbolic dependency graph (SDG)*] A symbolic dependency graph is a triple $G = (V, H, C)$, where V is a finite set of configurations, $H \subseteq V \times \mathcal{P}(\mathbb{N}_0 \times V)$ is a finite set of weighted hyper-edges, and $C \subseteq V \times (\mathbb{N}_0 \cup \{\infty\}) \times V$ is a finite set of cover-edges.

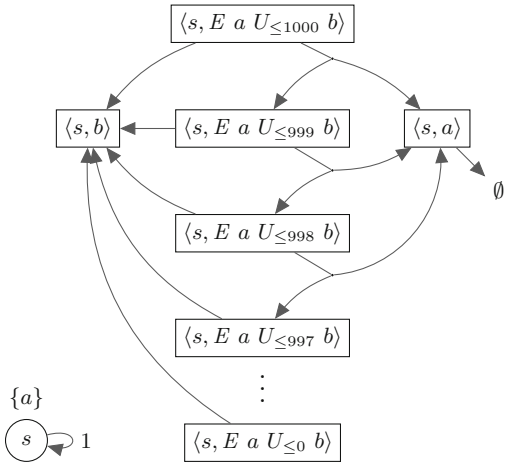
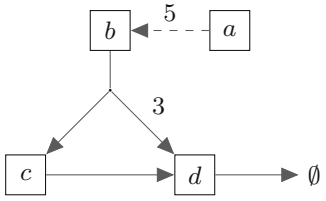


Fig. 6 A WKS and its dependency graph for $E\ a\ U_{\leq 1,000}\ b$



(a) A symbolic dependency graph

i	a	b	c	d
A_0	∞	∞	∞	∞
$F(A_0)$	∞	∞	∞	0
$F^2(A_0)$	∞	∞	0	0
$F^3(A_0)$	∞	3	0	0
$F^4(A_0)$	0	3	0	0
$F^5(A_0)$	0	3	0	0

(b) Minimum pre fixed-point computation

Fig. 7 Minimum prefixed-point assignment of an SDG. **a** A symbolic dependency graph. **b** Minimum prefixed-point computation

The difference from dependency graphs explained earlier is that for each hyper-edge of an SDG a weight is added to all of its target configurations, and a new type of edge called a cover-edge is introduced. Let $G = (V, H, C)$ be a symbolic dependency graph. The size of G is $|G| = |V| + |H| + |C|$ where $|V|$, $|H|$ and $|C|$ is the size of these components in a binary representation (as before the size of a hyper-edge depends on the number of nodes it connects to). For a hyper-edge $e = (v, T) \in H$ where v is the source configuration and T is the target set of e, we say that $(w, u) \in T$ is a hyper-edge branch with weight w pointing to the target configuration u. The edge-successor set $succ(v) = \{(v, T) \in H\} \cup \{(v, k, u) \in C\}$ is the set of hyper-edges and cover-edges with v as the source configuration.

Figure 7a shows an example of an SDG. The main difference is that symbolic dependency graphs operate over the complete lattice $\mathbb{N}_0 \cup \{\infty\}$, contrary to standard dependency graphs that use only boolean values. Hyper-edges are denoted by solid lines and hyper-edge branches have weight 0 unless they are annotated with another weight. Cover-edges are drawn as dashed lines annotated with a cover-condition.

Intuitively, in each iteration of the fixed-point computation, for each configuration in the graph we consider the minimum over all outgoing hyper-edges where for each hyper-edge we take the maximum over all values of the target configurations plus the corresponding weights. Contrary to this, a cover-edge propagates the value 0 to its source configuration, provided that the value in its target configuration is smaller than or equal to the cover-condition.

We shall now formally describe a global algorithm for the computation of the minimum prefixed-point. An assignment $A\colon V \to \mathbb{N}_0 \cup \{\infty\}$ in an SDG $G = (V, H, C)$ is a mapping from configurations to values. We denote the set of all assignments by *Assign* and define a partial order $\sqsubseteq$ over the assignments such that $A \sqsubseteq A'$ if and only if $A(v) \geq A'(v)$ for all $v \in V$. Note that the partial order is the opposite of the normal ordering on integers, hence, $A_0(v) = \infty$ for all $v \in V$ is the smallest element in the lattice.

A *prefixed-point assignment* is an assignment $A \in Assign$ such that $F(A) \sqsubseteq A$ where $F\colon Assign \to Assign$ is defined as

$$F(A)(v) = \begin{cases} 0 & \text{if there is } (v, k, u) \in C \text{ such that} \\ & A(u) \leq k < \infty, \text{ or } A(u) < k = \infty \\ \min\limits_{(v,T)\in H} \big(\max\{A(u) + w \mid (w, u) \in T\}\big) & \text{otherwise.} \end{cases} \tag{4}$$

We assume that $\max \emptyset = 0$ and $\min \emptyset = \infty$. The function F is clearly monotonic on the complete lattice of all assignments ordered by $\sqsubseteq$. It follows by the Knaster-Tarski fixed-point theorem that there exists a unique minimum prefixed-point assignment of G, denoted $A_{\min}$.

As the lattice contains no infinite decreasing sequences of weights (nonnegative integers), the minimum prefixed-point assignment $A_{\min}$ of G can be computed by a finite number of applications of the function F on the smallest assignment A_0, where all configurations have the initial value ∞. So

there exists an $m \in \mathbb{N}_0$ such that $F^m(A_0) = F^{m+1}(A_0)$, implying that $F^m(A_0) = A_{\min}$ is the minimum prefixed-point assignment of G. Figure 7b shows a computation of the minimum prefixed-point assignment on the given symbolic dependency graph.

The next theorem demonstrates that fixed-point computation via the global algorithm (repeated applications of the function F) on symbolic dependency graphs runs in polynomial time.

Theorem 3 *Computation of the minimum prefixed-point assignment for an SDG $G = (V, H, C)$ by repeated application of the function F runs in time $O(|V| \cdot |C| \cdot (|H| + |C|))$.*

Proof Let us first realize that a single iteration of F takes $O(|H| + |C|)$ as we go through all the edges and and for each such edge update the value of the source configuration. To prove our claim, it is so enough to establish that the algorithm terminates after no more than $|V| \cdot |C|$ iterations.

Let us first consider a symbolic dependency graph without any cover-edges $G = (V, H, \emptyset)$. We shall argue that the minimum prefixed-point assignment $A_{\min}$ is reached in at most $|V|$ iterations. For this purpose, let us write $u \Rightarrow v$ for $v, u \in V$, whenever there is $(v, T) \in H$ and $(w, u) \in T$ such that $A_{\min}(v) = A_{\min}(u) + w$. In other words, for any assignment A where $A(u) = A_{\min}(u)$, one additional iteration of F on A will guarantee that $F(A)(v) = A_{\min}(v)$. Clearly, all configurations v such that $(v, \emptyset) \in H$ get their final value $A_{\min}(v) = 0$ already in the first iteration of the function F; let us call such configurations terminal. Now any configuration v such that $A_{\min}(v) \neq \infty$ gets its final value $A_{\min}(v)$ in the number of iterations that corresponds to the shortest path from v to some terminal configuration in the graph $(V, \Rightarrow)$. Such a path surely contains at most $|V|$ configurations. Finally, any configuration v such that $A_{\min}(v) = \infty$ gets its final value already in the initial assignment A_0, implying that in any case we need to perform at most $|V|$ iterations of the function F before $A_{\min}$ is computed.

Assume now that the symbolic dependency graph contains cover-edges too. It is clear that when a cover-edge causes its source configuration to be updated, the configuration gets the value 0 and hence cannot be improved any more. Hence, after at most $|V|$ iterations, at least one cover-edge sets the value of its source configuration to 0 (unless the algorithm already found the minimum prefixed-point and terminated). After this we need to perform at most $|V|$ iterations before the same happens for another cover-edge, etc. Hence the total number of iterations is $O(|V| \cdot |C|)$ as required for establishing the claim of the theorem. □

We now propose a local algorithm for the computation of minimum prefixed-points on symbolic dependency graphs, motivated by the fact that in the model-checking we are often interested in the value of a single given configuration only, hence we may be able (depending on the formula we want to verify) to explore only a part of the reachable state space. This means that the dependency graph does not have to be constructed beforehand and new successors are generated only when needed (on-the-fly).

Algorithm 2: Symbolic local algorithm

Input: An SDG $G = (V, H, C)$ and a start configuration $v_0 \in V$.
Output: Minimum prefixed-point assignment $A_{\min}(v_0)$ for the configuration v_0.

```
1  Let A(v) = ⊥ for all v ∈ V
2  A(v0) = ∞; W = succ(v0)
3  while W ≠ ∅ do
4      Pick e ∈ W
5      W = W \ {e}
6      if e = (v, T) is a hyper-edge then
7          if ∃(w, u) ∈ T where A(u) = ∞ then
8              D(u) = D(u) ∪ {e}
9
10         else if ∃(w, u) ∈ T where A(u) = ⊥ then
11             A(u) = ∞; D(u) = {e}; W = W ∪ succ(u)
12         else
13             a = max{A(u) + w | (w, u) ∈ T}
14             if a < A(v) then
15                 A(v) = a; W = W ∪ D(v)
               let (w, u) = arg max_{(w,u)∈T} A(u) + w
16             if A(u) > 0 then
17                 D(u) = D(u) ∪ {e}
18
19
20     else if e = (v, k, u) is a cover-edge then
21         if A(u) = ⊥ then
22             A(u) = ∞; D(u) = {e}; W = W ∪ succ(u)
23
24         else if A(u) ≤ k < ∞ or A(u) < k = ∞ then
25             if A(v) > 0 then
26                 W = W ∪ D(v)
27             A(v) = 0
28         else
29             D(u) = D(u) ∪ {e}
30 return A(v0)
```

Given a symbolic dependency graph $G = (V, H, C)$, Algorithm 2 computes the minimum prefixed-point assignment $A_{\min}(v_0)$ of a configuration $v_0 \in V$. The algorithm is an adaptation of Algorithm 1. We use the same data-structures as in Algorithm 1. However, the assignment $A(v)$ for each configuration v now ranges over $\mathbb{N}_0 \cup \{\bot, \infty\}$ where $\bot$ once again indicates that the configuration has not been explored yet.

Table 1 lists the values of the assignment A, the waiting set W (implemented as a queue) and the dependency set D during the execution of Algorithm 2 on the SDG from Fig. 7a. Each row displays the values before the i'th iteration of the while-loop. The value of the dependency set $D(a)$ for a is not shown in the table because it remains empty. Initially, as we start the exploration from the configuration a, we change

Table 1 Execution of Algorithm 2 on SDG from Fig. 7a

i	$A(a)$	$A(b)$	$A(c)$	$A(d)$	W	$D(b)$	$D(c)$	$D(d)$
1	∞	$\bot$	$\bot$	$\bot$	$(a, 5, b)$			
2	∞	∞	$\bot$	$\bot$	$(b, \{(0, c), (3, d)\})$	$(a, 5, b)$		
3	∞	∞	∞	$\bot$	$(c, \{(0, d)\})$	$(a, 5, b)$	$(b, \{(0, c), (3, d)\})$	
4	∞	∞	∞	∞	$(d, \emptyset)$	$(a, 5, b)$	$(b, \{(0, c), (3, d)\})$	$(c, \{(0, d)\})$
5	∞	∞	∞	0	$(c, \{(0, d)\})$	$(a, 5, b)$	$(b, \{(0, c), (3, d)\})$	$(c, \{(0, d)\})$
6	∞	∞	0	0	$(b, \{(0, c), (3, d)\})$	$(a, 5, b)$	$(b, \{(0, c), (3, d)\})$	$(c, \{(0, d)\})$
7	∞	3	0	0	$(a, 5, b)$	$(a, 5, b)$	$(b, \{(0, c), (3, d)\})$	$(c, \{(0, d)\})$
8	0	3	0	0		$(a, 5, b)$	$(b, \{(0, c), (3, d)\})$	$(c, \{(0, d)\})$

its value in the current assignment from undefined to infinity and place its outgoing cover-edge to the waiting set W. In the next round, this edge is removed from W and the assignment for the configuration b is updated. The hyper-edge outgoing from b is then added to the waiting set and it is noted in the dependency set $D(b)$ that if the value of b ever changes in the future, it should be back propagated along the cover-edge $(a, 5, b)$ to the configuration a. The computation continues in this manner until the assignment of the configuration d is eventually updated to 0 in the fifth iteration. This means that the edge $(c, \{(0, d\})$ that depends on d is now reinserted into the waiting set W and in the next iteration of the algorithm the value of d is propagated to the configuration c. After this, the assignment of the configuration b gets updated to 3 and because $A(b) \leq 5$, the configuration a finally gets the value 0.

To prove the correctness of Algorithm 2, we extend the loop invariant for standard dependency graphs from [20] so that the weights are also considered.

Lemma 1 *The while-loop in Algorithm 2 satisfies the following loop-invariants (for all configurations $v \in V$):*

1. *If $A(v) \neq \bot$ then $A(v) \geq A_{\min}(v)$.*
2. *If $A(v) \neq \bot$ and $e = (v, T) \in H$, then either*
   (a) *$e \in W$,*
   (b) *$e \in D(u)$ for some $(w, u) \in T$ such that $A(v) \leq A(u) + w = \max\{A(u') + w' \mid (w', u') \in T\}$, or*
   (c) *$A(v) = 0$.*
3. *If $A(v) \neq \bot$ and $e = (v, k, u) \in C$, then either*
   (a) *$e \in W$,*
   (b) *$e \in D(u)$ and $A(u) > k$, or*
   (c) *$A(v) = 0$.*

Proof We shall now argue for the initialization and maintenance of the first invariant, followed by the arguments regarding the remaining two invariants.

Invariant (1). Initially, $A(v) = \bot$ for all $v \in V \backslash \{v_0\}$ and $A(v_0) = \infty$ for the initial configuration v_0. Hence the invariant trivially holds before the while-loop is entered. For the maintenance of the invariant, observe that the assignment A is updated only at lines 10, 14, 20 and 24. At lines 10 and 20 the value $A(v)$ updated to ∞ and clearly $\infty \geq A_{\min}(v)$. At line 14, the value of $A(v)$ is updated to $\max\{A(u) + w \mid (w, u) \in T\}$ for some hyper-edge (v, T). This update only happens if $A(u) \neq \bot$ for all $(w, u) \in T$ and by the invariant assumption, $A(u) \geq A_{\min}(u)$ for all such u. This implies that $A(v) \geq A_{\min}(v)$ after the update—compare this with the "otherwise" case in Eq. (4). Finally, at line 24, the value of $A(v)$ is updated to 0, provided that there exists a cover-edge (v, k, u) where $A(u) \leq k$. Clearly, $A(u) \neq \bot$ and we can use the invariant assumption to conclude that $A(u) \geq A_{\min}(u)$. As $A(u) \leq k$ then also $A_{\min}(u) \leq k$ and clearly $A_{\min}(v) = 0$ by the first case of the function in Eq. (4). Hence after the update of $A(v)$ we get that $A(v) \geq A_{\min}(v) = 0$ also at this line. In summary, Invariant (1) is preserved at any line where $A(v)$ changes its value.

Invariants (2) and (3). The two invariants hold initially, because $A(v) = \bot$ for all $v \in V \backslash \{v_0\}$ and for the initial configuration v_0, we have that $W = succ(v_0)$, meaning that every hyper-/cover-edge with the source configuration v_0 is in W, satisfying the two invariants in cases (2a) and (3a). For the maintenance of the invariants, observe that whenever a hyper-/cover-edge e is removed from W, it is added to the dependency set $D(u)$ of at least one target configuration u in edge e, unless it is the case that $A(v)$ gets the value 0. Also, whenever we explore a new configuration u by setting $A(u) = \infty$, we always add $succ(u)$ to W. By examining the pseudo-code, the conditions in cases (2b) and (3b) are also satisfied as requested by the lemma. We can so conclude that Invariants (2) and (3) are maintained during the execution of the while-loop. □

These loop-invariants allow us to conclude with a theorem proving the correctness of the local algorithm.

Theorem 4 (Correctness of local algorithm) *Algorithm 2 terminates and computes an assignment A such that $A(v) \neq \bot$ implies $A(v) = A_{\min}(v)$ for all $v \in V$. In particular, the*

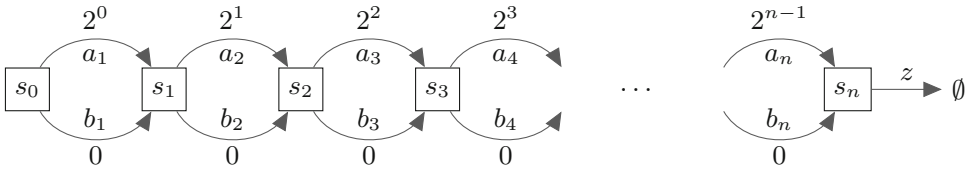


Fig. 8 An SDG where the local algorithm can run in exponential time

returned value $A(v_0)$ is the minimum prefixed-point assignment of v_0.

Proof Let us first discuss termination of Algorithm 2. The while-loop in Algorithm 2 terminates once the waiting set W is empty. Clearly, at every execution of the body of the while-loop, there is one edge removed from W. To argue that W eventually becomes empty, observe that whenever cover-/hyper-edges are added to W, then in the same iteration there is a configuration v such that the value of $A(v)$ decreases (assuming that $\bot > \infty > n$ for any integer n). Together with the fact that for each v the value $A(v)$ is nonincreasing, while the minimum possible value is 0, we get that cover-/hyper-edges are added to W only finitely many times. This implies the termination of the algorithm.

We shall now argue about the correctness of the algorithm. By Invariant (1) of Lemma 1 we know that $A(v) \geq A_{\min}(v)$ for all $v \in V$ where $A(v) \neq \bot$. From the minimality of $A_{\min}$, it is now enough to show that A is a prefixed point assignment, i.e. $F(A)(v) \geq A(v)$ for all v where $A(v) \neq \bot$, to conclude that $A(v) = A_{\min}(v)$ for all v where $A(v) \neq \bot$. To do so, there are two cases according to Eq. (4) that we must verify for all v such that $A(v) \neq \bot$.

(i) Whenever $(v, k, u) \in C$ such that $A(u) \leq k < \infty$ or $A(u) < k = \infty$ then $A(v) = 0$.
(ii) Whenever $(v, T) \in H$ then $A(v) \leq \max\{A(u) + w | (w, u) \in T\}$.

For case (i), assume that there is a $(v, k, u) \in C$ such that $A(u) \leq k < \infty$ or $A(u) < k = \infty$. We consider now Invariant (3) from Lemma 1. As $W = \emptyset$ at the termination of the algorithm, we can eliminate part (3a) of the invariant. Clearly, part (3b) does not hold either due to our assumption, leaving us with the validity of part (3c) saying that $A(v) = 0$ as required.

For case (ii), let $(v, T) \in H$. We consider Invariant (2) from Lemma 1. As before part (2a) is eliminated due to the fact that $W = \emptyset$. In any of the two remaining options in parts (2b) and (2c) we get $A(v) \leq \max\{A(u) + w | (w, u) \in T\}$ as required.

Consequently, we can conclude that upon the termination of Algorithm 2, the assignment $A(v)$ is the minimum prefixed-point assignment for all $v \in V$ where $A(v) \neq \bot$. As $A(v_0)$ is initially assigned the value ∞, we get $A(v_0) = A_{\min}(v_0)$ as claimed by the theorem. □

We note that the termination argument in Theorem 4 is not completely straightforward because there is no guarantee that it terminates within a polynomial number of steps. This is depicted on the SDG in Fig. 8 where, for technical convenience, we named the hyper-edges $a_1, \ldots, a_n, b_1, \ldots, b_n$ and z. Consider now an execution of Algorithm 2 starting from the configuration s_0 where we pick the edges from the waiting set W at line 4 according to the strategy:

- if $z \in W$ then pick z, else
- if $a_i \in W$ for some i then pick a_i (there will be at most one such a_i), else
- pick $b_i \in W$ with the smallest index i.

Then the initial assignment of $A(s_0) = \infty$ is gradually improved to $2^n - 1$, $2^n - 2$, $2^n - 3, \ldots 1$, 0. Hence, in the worst case, the local algorithm can perform exponentially many steps before it terminates, whereas the global algorithm always terminates in polynomial time. On the other hand, as we will see in Sect. 7, the local algorithm is in practise performing significantly better despite its high theoretical complexity. This is also partly due to the fact that once we know that $A(v_0) = 0$, we can terminate the algorithm early even though $W \neq \emptyset$.

6 WCTL and symbolic dependency graphs

We are now ready to present an encoding of a WKS and a WCTL formula into symbolic dependency graphs to decide the model checking problem via the computation of the minimum prefixed-point assignment.

Given a WKS $\mathcal{K}$, a state s of $\mathcal{K}$ and a WCTL formula φ, we construct the corresponding symbolic dependency graph rooted at $\langle s, \varphi \rangle$ as before, with the exception of the existential and universal until operators that are now encoded by the rules in Fig. 9. Note that for the formulae $E\, \varphi_1\, U_{\leq \infty}\, \varphi_2$ and $A\, \varphi_1\, U_{\leq \infty}\, \varphi_2$ we still use the rules from Fig. 4 as the weights are not important in these cases.

In Fig. 10 we present the constructed symbolic dependency graph for the lawn mower example from Fig. 2. The values in the minimum prefixed-point assignment are depicted to the right of each configuration.

Theorem 5 (Correctness of the encoding) *Let $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ be a WKS, $s \in S$ a state, and φ a WCTL formula. Let G be the constructed symbolic dependency graph rooted with $\langle s, \varphi \rangle$. Then $s \models \varphi$ if and only if $A_{\min}(\langle s, \varphi \rangle) = 0$.*

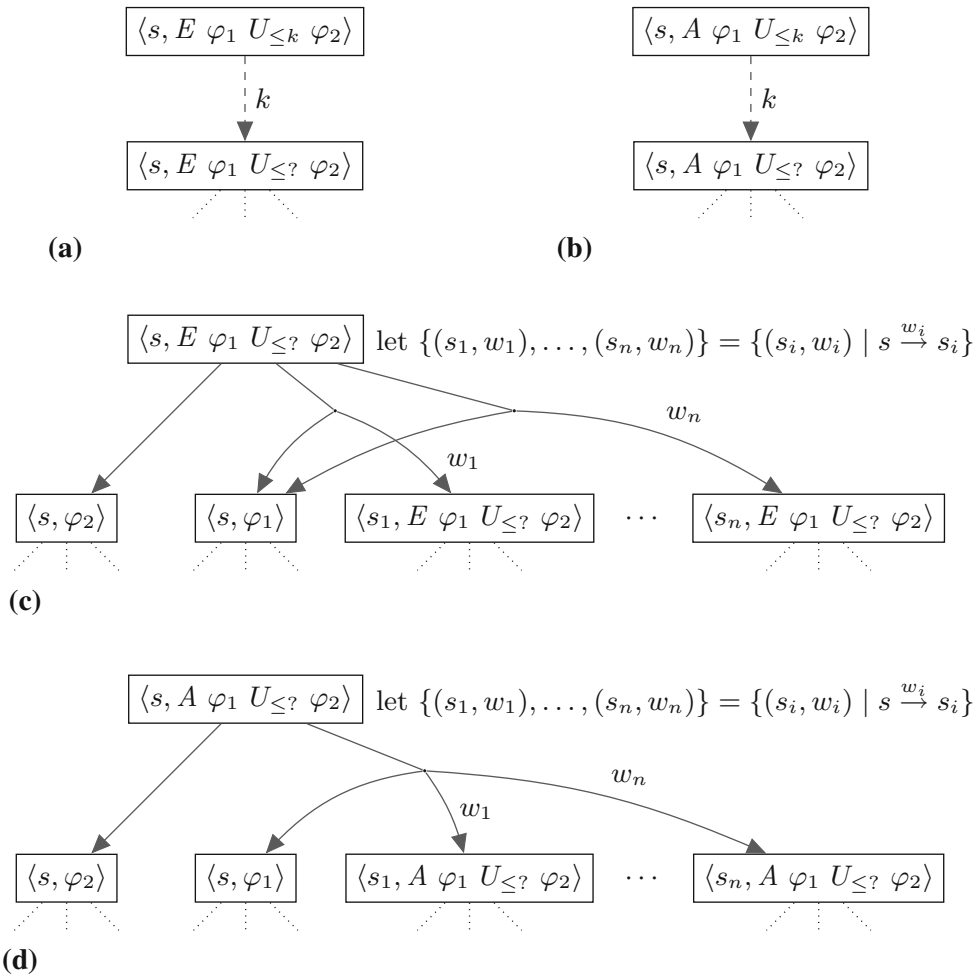


Fig. 9 SDG encoding of existential and universal 'until' formulas. **a** Existential until where $k \neq \infty$. **b** Universal until where $k \neq \infty$. **c** Existential until unfolding. **d** Universal until unfolding

Proof We first observe that there are two kinds of configurations in the symbolic dependency graph rooted with $\langle s, \varphi \rangle$. Configurations of the form $\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$ or $\langle s, A\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$ may have non-zero hyper-edge weights and we refer to them as *symbolic configurations*. All other configurations have zero-weighted outgoing hyper-edges or an outgoing cover-edge and we call them *concrete configurations*. Hence e.g. $\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$ is regarded as a concrete configuration.

We shall now argue for the following two properties of concrete and symbolic configurations.

(i) For any concrete configuration $v = \langle s, \varphi \rangle$ holds that either $A_{\min}(v) = 0$ or $A_{\min}(v) = \infty$. Moreover, $A_{\min}(\langle s, \varphi \rangle) = 0$ if and only if $s \models \varphi$.

(ii) For any symbolic configuration $v = \langle s, Q\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$ where $Q \in \{E, A\}$ holds that (a) if $A_{\min}(v) \in \mathbb{N}_0$ then $s \models Q\ \varphi_1\ U_{\leq A_{\min}(v)}\ \varphi_2$, and (b) if $s \models Q\ \varphi_1\ U_{\leq k}\ \varphi_2$ then $A_{\min}(v) \leq k$.

Theorem 5 is now implied by property (i). The fact that for any concrete configuration v either $A_{\min}(v) = 0$ or $A_{\min}(v) = \infty$ follows directly from the definition of $A_{\min}$ and the encoding rules; note that any cover edge can update the source configuration only to the value 0. The proofs of properties (i) and (ii) are now established by structural induction on φ.

The cases for $\varphi = \mathbf{true}$, $\varphi = a$, $\varphi = \varphi_1 \wedge \varphi_2$, $\varphi = \varphi_1 \vee \varphi_2$, $\varphi = E\ \varphi_1\ U_{\leq \infty}\ \varphi_2$, $\varphi = A\ \varphi_1\ U_{\leq \infty}\ \varphi_2$, $\varphi = EX_{\leq k}\ \varphi_1$ and $\varphi = AX_{\leq k}\ \varphi_1$ follow the same arguments as in the proof of Theorem 2.

Let us so assume that $\varphi = E\ \varphi_1\ U_{\leq k}\ \varphi_2$ where $k < \infty$. We want to argue for property (i), in other words we want to show that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 0$ if and only if $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$. From Fig. 9a we see that any concrete configuration $\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$ has a single cover-edge with the cover-condition k leading to the symbolic configuration $v = \langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$ for which we assume that property (ii) holds. If $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 0$ then necessarily $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle) \leq k$, which implies by property (ii) part (a) that $s \models E\ \varphi_1\ U_{\leq k'}\ \varphi_2$ where $k' = A_{\min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle)$ and hence also $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ as $k' \leq k$. On the other hand, if $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ then $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle) \leq k$ by property (ii) part (b). This implies by the semantics of the cover-edge that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 0$.

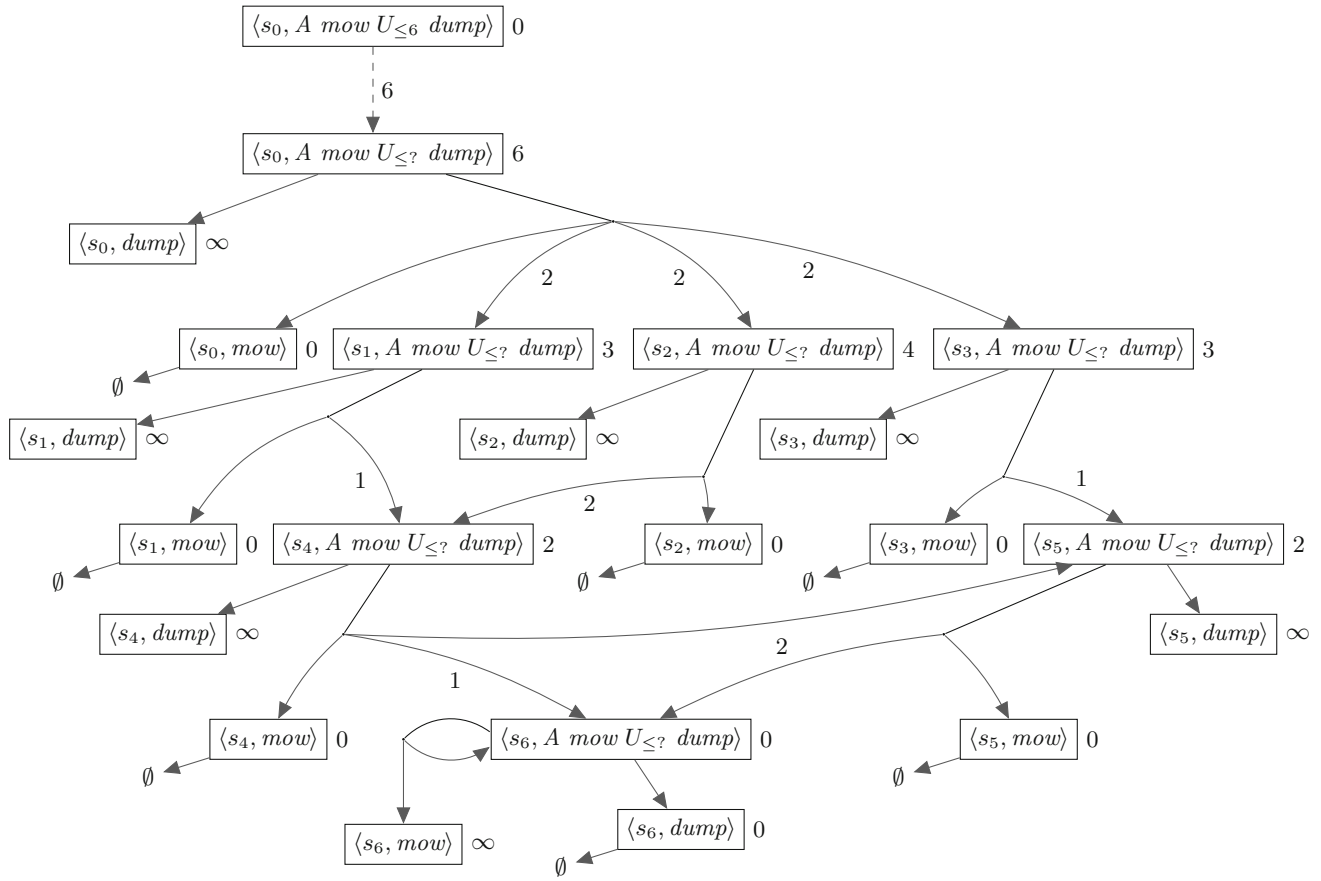


Fig. 10 Symbolic dependency graph for the lawn mower example (all missing weights are by default 0)

Let $\varphi = E\ \varphi_1\ U_{\leq ?}\ \varphi_2$. We want to show property (ii). Let us first argue about part (b). Assume that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$. Hence there exists a run σ and a position $p \geq 0$ such that $\sigma(p) \models \varphi_2$, $\sigma(p') \models \varphi_1$ for all $p' < p$, and $W_\sigma(p) \leq k$. By induction on p we show that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) \leq k$. Clearly, if $p = 0$ then $\sigma(0) = s \models \varphi_2$, which by the left-most edge in Fig. 9c implies that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) = 0 \leq k$. If $p > 0$ then $\sigma(0) = s \models \varphi_1$ and there is a transition $\sigma(0) \xrightarrow{w_i} s_i$ such that $s_i \models E\ \varphi_1\ U_{\leq k - w_i}\ \varphi_2$. By structural induction $\sigma(0) = s \models \varphi_1$ implies that $A_{\min}(\langle s, \varphi_1\rangle) = 0$ due to property (i) and the mathematical induction on p gives us that $A_{\min}(\langle s_i, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) \leq k - w_i$. By combining these two facts using the rule in Fig. 9c, we conclude that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) \leq k$. Let us now focus on part (a) of property (ii). Assume that $A_{\min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) = k \in \mathbb{N}_0$. We want to prove that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$. Considering the hyper-edges in Fig. 9c, the reason for $k \in \mathbb{N}_0$ can be due to the fact that $A_{\min}(\langle s, \varphi_2\rangle) = 0$ (note that $\langle s, \varphi_2\rangle$ is a concrete configuration) but then we are done as $s \models E\ \varphi_1\ U_{\leq 0}\ \varphi_2$. The other reason for $k \in \mathbb{N}_0$ can be that $A_{\min}(\langle s, \varphi_1\rangle) = 0$ (again this is a concrete configuration) and there is a transition $s \xrightarrow{w_i} s_i$ such that $A_{\min}(\langle s_i, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) = k - w_i$; this follows from the rule in Fig. 9c. We can repeat the same argument for $\langle s_i, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle$. If all such successors never contain a hyper-edge to a node of the form $\langle s', \varphi_2\rangle$ where $A_{\min}(\langle s', \varphi_2\rangle) = 0$ then clearly the minimum prefixed-point assignment never gets a value different from ∞, as we only have cyclic dependencies. The rest of the property is now established by similar arguments as in the proof of Theorem 2.

For $\varphi = A\ \varphi_1\ U_{\leq k}\ \varphi_2$ where $k < \infty$ and $\varphi = A\ \varphi_1\ U_{\leq ?}\ \varphi_2$ we can use similar arguments as for the existential until cases discussed above. □

In Fig. 11 we depict the symbolic dependency graph encoding of $E\ a\ U_{\leq 1,000}\ b$ for the configuration s in the single-state WKS from Fig. 6. This clearly illustrates the succinctness of SDG compared to the standard dependency graphs. The minimum prefixed-point assignment of this symbolic dependency graph is now reached in two iterations of the function F defined in Eq. (4).

We note that for a given WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ and a formula φ, the size of the constructed symbolic dependency graph $G = (V, H, C)$ can be bounded as follows: $|V| = O(|S| \cdot |\varphi|)$, $|H| = O(|\rightarrow| \cdot |\varphi|)$ and $|C| = O(|\varphi|)$. In combination with Theorem 3 and the fact that $|C| \leq |H|$ (due to the rules for construction of G), we conclude with a

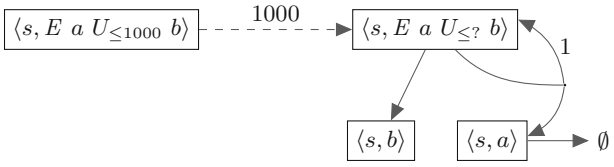


Fig. 11 SDG for $s \models E\ a\ U_{\leq 1,000}\ b$ and the WKS from Fig. 6

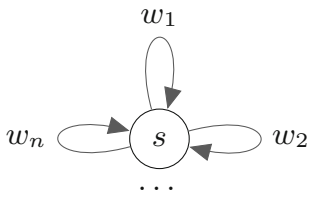


Fig. 12 Subset-sum construction

theorem stating a polynomial time complexity of the global model-checking algorithm for WCTL.

Theorem 6 *Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, a state $s \in S$ and a WCTL formula φ, the model-checking problem $s \models \varphi$ is decidable in time $O(|S| \cdot |\rightarrow| \cdot |\varphi|^3)$.*

Contrary to the polynomial running time of the global algorithm, the local model-checking approach from Algorithm 2 can run (in the worst case) in exponential time. Nevertheless, the experiments in the section to follow show that this does not happen in practise and that the local algorithm is the preferred choice for practical applications.

We conclude this section with the fact that model-checking of until-formulae with interval bounds (both lower and upper bounds) is already NP-hard. The proof is by reduction from the well-known NP-complete problem subset-sum [21, Chap. 5]: given a set of integers $W = \{w_1, \ldots, w_n\}$ and a target integer T, is there a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in {\mathbb{N}_0}^n$ such that

$$\sum_{i=1}^{n} w_i \cdot x_i = T \quad ?$$

Given an instance W and T of the subset-sum problem, we construct a WKS $\mathcal{K}$ as shown in Fig. 12 such that for every $w \in W$ we add a self-loop labelled with w. It is now easy to check that the formula $s \models E\ \mathbf{true}\ U_{[T,T]}\mathbf{true}$ is satisfied if and only if W and T is a positive instance of the subset-sum problem. The model-checking problem is hence NP-hard.

7 Experiments

To compare the performance of the algorithms for model checking WCTL, we developed a prototype tool implementation. There is a web-based front-end written in CoffeeScript available at http://wktool.jonasfj.dk/ and the tool is entirely browser-based, requiring no installation. Screenshots of the tool in the editor mode (showing our running lawn mower example) and the visualization of the corresponding weighted Kripke structure are showed in Fig. 13.

In the experiments, the model-checking algorithms run with limited memory resources but the tool allows a fair comparison of the performance for the different algorithms. All experiments were run on a standard laptop (Intel Core i7) running Ubuntu Linux.

To experiment with larger, scalable models consisting of parallel components, we extend the process algebra CCS [22] with weight prefixing as well as proposition annotations. The CCS syntax for our running example is given below.

```
S0 := mow:(<go,2>.S1 + <go,2>.
S2 + <go,2>.S3);
S1 := mow:<go,1>.S4;
S2 := mow:<go,2>.S4;
S3 := mow:<go,1>.S5;
S4 := mow:(<go,0>.S5 + <go,1>.S6);
S5 := mow:<go,2>.S6;
S6 := dump:<go,0>.S6;
```

The weighted CCS defining equations and the annotations with atomic propositions should be self-explanatory. We just point to the fact that in weighted CCS transitions are additionally labelled by actions (in our case by a single action `go`) so that a synchronization among different parallel components is possible. The possibility to communicate is also exploited in the weighted models of Leader Election [13], Alternating Bit Protocol [7], and Task Graph Scheduling problems for two processors [17] used in our experiments. All weighted CCS models of the experiments are available within the tool. The weight (communication cost) is associated with sending messages in the first two models while in the task graph scheduling the weight represents clock ticks of the processors.

7.1 Standard vs. symbolic dependency graphs

In Table 2 we compare the direct (standard dependency graph) algorithms with the symbolic ones. The execution times are in seconds and OOM indicates when verification runs out of memory. For a fixed size of the problems, we scale the bound k in the WCTL formulae.

In the leader election protocol we have a ring network where each processes has a unique id (natural number). Initially, each process sends to its right neighbour its id and then keeps forwarding to the right neighbour any number received from its left neighbour, provided that it is larger than its own id. The communication is asynchronous and the process that receives from its left neighbour its own id claims to become the leader. This is signalled by setting the proposition *leader* to true. In the protocol with eight processes we verified a

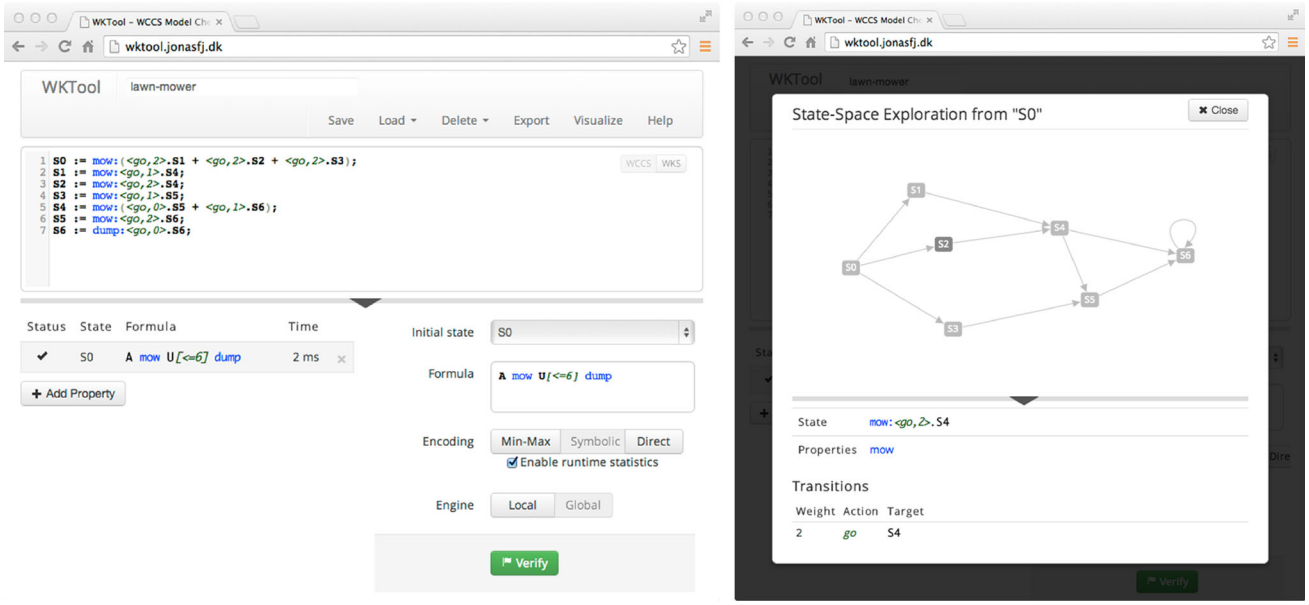


Fig. 13 WKTool screenshots (editor and visualizer)

Table 2 Scaling of bounds in WCTL formula (time in seconds)

k	Direct		Symbolic	
	Global	Local	Global	Local
Leader election				
Satisfied				
200	3.88	0.23	0.26	0.02
400	8.33	0.25	0.26	0.02
600	OOM	0.24	0.26	0.02
800	OOM	0.25	0.26	0.02
1,000	OOM	0.26	0.27	0.02
Unsatisfied				
200	7.76	8.58	0.26	0.26
400	17.05	20.23	0.26	0.26
600	OOM	OOM	0.26	0.26
800	OOM	OOM	0.26	0.26
1,000	OOM	OOM	0.26	0.26
Alternating bit protocol				
Satisfied				
100	3.87	0.05	0.23	0.03
200	8.32	0.06	0.23	0.03
300	OOM	0.10	0.28	0.04
400	OOM	0.11	0.23	0.03
500	OOM	0.13	0.23	0.03
Unsatisfied				
100	3.39	3.75	0.27	0.23
200	6.98	8.62	0.30	0.25
300	OOM	15.37	0.28	0.24
400	OOM	OOM	0.27	0.24
500	OOM	OOM	0.27	0.22

satisfiable formula E **true** $U_{\leq k}$ *leader*, asking if a leader can be determined within k message exchanges, and an unsatisfiable formula E **true** $U_{\leq k}$ *leader* > 1, asking if there can be more than one leader selected within k message exchanges.

Alternating bit protocol is a simple communication protocol that guarantees a correct message exchange between a sender and a receiver that communicate over unreliable (lossy) FIFO channels. The sender can be in a state s_0 that signals that it is appending the bit 0 to the transmitted message, or in the state s_1 where the bit 1 is appended. Similarly, the receiver is expected to receive a message with appended bit 0 in its state d_0 and the bit 1 in the state d_1. Once a message is delivered, the proposition *delivered* is set to true. For an instance of the protocol with a communication buffer of size four, we verified a satisfied formula E **true** $U_{\leq k}$ *delivered* $= 1$, asking if a message can be delivered within k communication steps, and an unsatisfied formula E **true** $U_{\leq k}$ $(s_0 \wedge d_1) \vee (s_1 \wedge d_0)$, asking whether the sender and the receiver can get out of synchrony within the first k communication steps.

For the satisfied formulae in these two experiments, the direct global algorithm (global fixed-point computation on dependency graphs) runs out of memory as the bound k in the formulae is scaled. The advantage of Liu and Smolka [20] local algorithm is obvious, as on positive instances it performs (using a DFS search strategy) about as well as the global symbolic algorithm. The local symbolic algorithm clearly performs best. We observed a similar behaviour also for other examples we tested and the symbolic algorithms were regularly performing better than the ones using the direct translation of WCTL formulae into dependency graphs. Hence, from now on, we shall focus on a more

detailed comparison of the local vs. global symbolic algorithms.

7.2 Local vs. global symbolic dependency graphs

In Table 3 we return to the leader election and alternating bit protocol but we scale the sizes (number of processes and buffer capacity, resp.) of these models rather than the bounds in formulae. The satisfiable and unsatisfiable formulae are as before. In the leader election the verification of a satisfiable formula using the local symbolic algorithm is consistently faster as the instance size is incremented, while for unsatisfiable formulae the verification times are essentially the same. For the alternating bit protocol, we present the results for the bound k equal to 10, 20 and ∞. While the results for unsatisfiable formulae do not change significantly, for the positive formula the bound 10 is very tight in the sense that there are only a few executions or "witnesses" that satisfy the formula. As the bound is relaxed, more solutions can be found which is reflected by the improved performance of the local algorithm, in particular in the situation where the upper-bound is ∞.

We also tested the algorithms on a larger benchmark of task graph scheduling problems [6]. The task graph scheduling problem asks about schedulability of a number of parallel tasks with given precedence constraints and processing times that are executed on a fixed number of homogeneous

Table 3 Scaling the model size for the symbolic algorithms (time in seconds)

n	$k = 200$	
	Global	Local
Leader election		
Satisfied		
7	0.08	0.01
8	0.26	0.02
9	1.06	0.03
10	5.18	0.03
11	23.60	0.03
12	Timeout	0.04
Unsatisfied		
7	0.08	0.08
8	0.26	0.26
9	1.05	1.06
10	4.97	4.96
11	23.57	24.07
12	Timeout	Timeout

n	$k = 10$		$k = 20$		$k = \infty$	
	Global	Local	Global	Local	Global	Local
Alternating bit protocol						
Satisfied						
5	0.33	0.10	0.33	0.07	0.33	0.04
6	0.78	0.18	0.77	0.17	0.80	0.06
7	1.88	0.34	1.92	0.14	1.96	0.05
8	4.82	0.82	4.71	0.72	4.78	0.09
9	13.91	10.60	12.41	1.67	12.92	0.20
10	OOM	OOM	OOM	6.29	OOM	0.23
Unsatisfied						
4	0.27	0.24	0.27	0.23	0.29	0.24
5	0.54	0.43	0.51	0.37	0.57	0.40
6	1.42	0.98	1.21	0.93	1.31	1.02
7	2.70	2.05	2.93	2.06	3.14	2.21
8	6.15	4.98	7.08	5.57	6.86	5.34
9	OOM	OOM	OOM	OOM	OOM	OOM

Table 4 Scaling task graphs by the number of initial tasks (time in seconds)

n	T0		T1		T2	
	Global	Local	Global	Local	Global	Local
Satisfied						
2	0.24	0.04	0.06	0.01	0.07	0.01
3	3.11	0.01	0.15	0.08	0.19	0.01
4	4.57	1.13	0.18	0.08	0.88	0.19
5	6.09	0.03	2.73	0.01	7.05	0.02
6	OOM	OOM	5.27	1.08	OOM	1.44
7	OOM	0.02	OOM	0.02	OOM	0.01
8	OOM	0.03	OOM	OOM	OOM	2.75
9	OOM	OOM	OOM	OOM	OOM	1.86
10	OOM	0.03	OOM	OOM	OOM	OOM
Unsatisfied						
2	0.22	0.20	0.05	0.05	0.08	0.01
3	2.91	2.55	0.14	0.13	0.20	0.01
4	6.35	4.45	0.16	0.14	0.91	0.20
5	7.45	5.00	2.31	1.69	7.48	0.03
6	OOM	OOM	4.67	4.40	OOM	1.40
7	OOM	OOM	OOM	OOM	OOM	OOM

Table 5 Summary of task graphs verification (180 cases in total)

180 task graphs for Algorithm	$k = 30$		$k = 60$		$k = 90$	
	Global	Local	Global	Local	Global	Local
Number of finished tasks	32	85	32	158	32	178
Accumulated time (s)	50.4	12.9	47.6	2.30	47.32	0.44

processors [17]. We automatically generate models for two processors from the benchmark containing in total 180 models and scaled them by the number of initial tasks that we include from each case into schedulability analysis.

The first three task graphs (T0, T1 and T2) are presented in Table 4. We model check nested formulae and the satisfiable one is E **true** $U_{\leq 90}$ $\left(t_{n-2}^{ready} \wedge A \textbf{ true } U_{\leq 80}\ done\right)$ asking whether there is, within 90 clock ticks, a configuration where the task t_{n-2} can be scheduled such that then we have a guarantee that the whole schedule terminates within 80 ticks. When the upper-bounds are decreased to 5 and 10, the formula becomes unsatisfiable for all task graphs in the benchmark.

Finally, we verify the formula E **true** $U_{\leq k}$ *done* asking whether the task graph can be scheduled within k clock ticks. We run the whole benchmark through the test (180 cases) for values of k equal to 30, 60 and 90, measuring the number of finished verification tasks (without running out of resources) and the total accumulated time it took to verify the whole benchmark for those cases where both the global and local algorithms provided an answer. The results are listed in Table 5. This provides further evidence for the claim that the local algorithm profits from the situation where there are more possible schedules as the bound k is being relaxed.

8 Conclusion

We suggested a symbolic extension of dependency graphs to verify negation-free weighted CTL properties where temporal operators are annotated with upper-bound constraints on the accumulated weight. Then we introduced global and local algorithms for the computation of fixed points to answer the model checking problems for the logic. The algorithms were implemented and experimented with, coming to the conclusion that the local symbolic algorithm is the preferred one, providing order of magnitude speedup in the cases where the bounds in the logical formula allow for a larger number of possible witnesses of satisfiability of the formula.

In the future work, we will study a weighted CTL logic with negation that combines lower- and upper-bounds (the model-checking problem for a logic containing weight intervals is already NP-hard as showed in Sect. 6). From the practical point of view it would be worth designing good heuristics that can guide the search in the local algorithm to find faster the witnesses of satisfiability of a formula. Another challenging problem is to adapt our technique to support alternating fixed points.

Acknowledgments We thank the anonymous reviewers for their useful comments and suggestions. The research leading to these results has received funding from the EU Seventh Framework Programme (FP7/2007–2013) under Grant Agreement No. 601148 (CASSTING).

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. **104**(1), 2–34 (1993)
2. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Paterson, M. (ed.) ICALP, vol. 443 of LNCS, pp. 322–335. Springer, New York (1990)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM **43**(1), 116–146 (1996)
4. Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01), vol. 2034 of LNCS, pp. 49–62. Springer, New York (2001)
5. Andersen, H.R.: Model checking and boolean graphs. Theor. Comput. Sci. **126**(1), 3–30 (1994)
6. Kasahara Laboratory at Waseda University. Standard task graph set. http://www.kasahara.elec.waseda.ac.jp/schedule/
7. Bartlett, K.A., Scantlebury, R.A., Wilkinson, P.T.: A note on reliable full-duplex transmission over half-duplex links. Commun. ACM **12**(5), 260–261 (1969)
8. Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01), vol. 2034 of LNCS, pp. 147–161. Springer, New York (2001)
9. Bouyer, P., Larsen, K.G., Markey, N.: Model checking one-clock priced timed automata. Log. Methods Comput. Sci. **4**(2) (2008)
10. Brihaye, T., Bruyère, V., Raskin, J.-F.: Model-checking for weighted timed automata. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT, vol. 3253 of LNCS, pp. 277–292. Springer, New York (2004)
11. Buchholz, P., Kemper, P.: Model checking for a class of weighted automata. Discret. Event Dyn. Syst. **20**, 103–137 (2010)
12. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Proceedings of the 16th International Conference on Concurrency Theory CONCUR'05, vol. 3653 of LNCS, pp. 66–80. Springer, New York (2005)
13. Chang, E., Roberts, R.: An improved algorithm for decentralized extrema-finding in circular configurations of processes. Commun. ACM **22**(5), 281–283 (1979)
14. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theor. Comput. Sci. **380**(1–2), 69–86 (2007)
15. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata, 1st edn. Springer Publishing Company, Incorporated (2009)
16. Knaster, B.: Un théorème sur les fonctions d'ensembles. Ann. de la Société Polonaise de Mathématique **6**, 133–134 (1928)
17. Kwok, Y.-K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. J. Parallel Distrib. Comput. **59**(3), 381–422 (1999)
18. Laroussinie, F., Markey, N., Oreiby, G.: Model-checking timed ATL for durational concurrent game structures. In: Asarin, E., Bouyer, P. (eds.) FORMATS, vol. 4202 of LNCS, pp. 245–259. Springer, New York (2006)
19. Liu, X., Ramakrishnan, C.R., Smolka, S.A.: Fully local and efficient evaluation of alternating fixed points. In: Tools and Algorithms for the Construction and Analysis of Systems. vol. 1384 of LNCS, pp. 5–19. Springer, Berlin (1998)
20. Liu, X., Smolka, S.A.: Simple linear-time algorithms for minimal fixed points (extended abstract). In: Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98), vol. 1443 of LNCS, pp. 53–66. Springer, New York (1998)

21. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, New York (1990)
22. Milner, R.: A calculus of communicating systems. In: LNCS, vol. 92 (1980)
23. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pac. J. Math. **5**(2), 285–309 (1955)