Local Model Checking of Weighted CTL

Jonas Finnemann Jensen,
Lars Kærlund Østergaard

Department of Computer Science, Aalborg University, Denmark

Abstract. Local or "on-the-fly" model checking techniques have become attractive over the years, due to the way they explore the state-space incrementally, potentially reducing space and time requirements for analysis and verification. We demonstrate how Liu and Smolkas dependency graph framework and local algorithm for evaluating alternating-free fixed points can be used to decide satisfaction of weighted CTL formulae over weighted Kripke structures. This is achieved by encoding the problem as a dependency graph that is pseudo-polynomial in the size of the bound of the formula. We propose an extension of dependency graphs called symbolic dependency graphs and a local algorithm for this framework. We show that the symbolic encoding is polynomial in the size of the formula and model. Lastly, we demonstrate the advantages of our approach through experiments.

1 Introduction

CTL model checking is the process of determining whether a given finite structure is a model of a given formula expressed in branching time temporal logic (CTL). Like many other model checking problems, CTL model checking suffers from the well-known problem of state space explosion, because the state space of the model of interest may grow exponentially in the number of concurrently executing components. Since the state-space problem has been a major concern for model checking, several techniques have been proposed to combat this issue. For instance, Burch, Clarke and McMillan suggest symbolic model checking, using BDDs allowing many practical systems of enormous size to become verifiable using the mu-calculus [3]. Another approach is to attempt to limit the size of the state space by only exploring what is required to determine the satisfaction of some property.

Often in model checking we are only interested in the satisfiability of a property for a particular state, rather than for every state of the system. So, instead of executing a so-called global algorithm that computes the value for every state by generating the entire state space, it is possible to limit the scope to a single state and generate the state-space incrementally in a need-driven fashion. The latter approach is what we refer to as a local algorithm. This technique also applies to the state-space explosion problem, i.e. it may simply be intractable to determine the satisfaction of some property for the entire system, hence we may be forced to use a local strategy and restrict our attention to a particular state, and hopefully the subset of the state-space generated is small enough for verification to succeed.

Many verification problems such as liveness, safety, and fairness properties can be expressed using fixed points. This is of particular interest with respect to the modal mu-calculus that enables the formulation of various properties of systems, using smallest and greatest fixed points. Hence, the satisfaction of formulae expressed using the modal mu-calculus can be determined by evaluating fixed these points. The full modal mu-calculus is general enough to express properties formulated in either CTL or Linear Temporal Logic (LTL). Practical solutions for the mu-calculus are often also applicable to these other logics. So, some of the same fixed point evaluation techniques to implement algorithms for verifying these logics. One such technique for evaluating fixed points was proposed by Liu and Smolka proposed a technique in [7]. They describe an abstract framework called dependency graphs that encodes boolean equation systems and present a global and local algorithm for computing alternating-free fixed points of dependency graphs in linear time. In turn this technique can be used for model checking the alternation-free modal mu-calculus.

We define weighted CTL (WCTL); an extension of CTL, where formulae are parametrized with constants that define an upper-bound on the cost for which a particular formula must be satisfied. The motivation for this extension is the need for being able to express quantities, such as costs or distances when verifying models with transitions carrying weights. We formulate such weighted models using weighted Kripke structures, which simply extend usual Kripke structures with a weighted transition relation.

We demonstrate how WCTL model checking of weighted Kripke structures can be performed by encoding the problem using Liu and Smolka's dependency graph framework. As it turns out, we demonstrate that for our particular weighted formalism, the size of the dependency graphs is pseudo-polynomial in the weight parameter of a WCTL formula. We achieve a reduction in the size of the input dependency graphs by extending them using symbolic dependency graphs. The main advantage of this new formalism is that size of the graph is only polynomial in the size of the input. Finally, we describe a local algorithm for evaluating minimal fixed points on symbolic dependency graphs in pseudo code and provide experimental results comparing the two approaches described in this paper using both a local and global algorithm.

Related Work

Fahrenberg et. al. define a quantitative weighted CTL (WCTL) for reasoning about quantitative aspects regarding weighted Kripke structures in [5], where truth values of formulae are in the domain $\mathcal{R}_{\geq 0} \cup \{\infty\}$. In contrast, the satisfaction of formulae is still interpreted in the boolean domain in the variant of WCTL described in this paper. Buchholz and Kemper propose a valued computation tree logic (CTL\$) interpreted over a general set of weighted automata that includes CTL in the logic as a the special case over the boolean semiring in [2]. 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 an on-the-fly technique for verification.

Several approaches to on-the-fly/local algorithms for model checking the modal mu-calculus have been proposed. H. Andersen 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 labeled transition system) in [1]. Liu and Smolka improve on the complexity of this approach with a local algorithm for evaluating alternating-free fixed points that runs in $O(n)$, where n is the size of the input graph [7]. This is the algorithm we apply for WCTL model checking and the one we extend for symbolic dependency graphs. Cassez et. al. present a symbolic extension for the algorithm of Liu and Smolka; a zone-based forward, local algorithm for solving timed reachability games [4]. Liu, Ramakrishnan and Smolka also introduce a local algorithm in [6] 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. However, we do not consider the evaluation of alternating fixed points in this paper.

Outline

Weighted Kripke structures and weighted CTL are presented in Section 2. Section 3 introduces dependency graphs and model checking with this framework is covered in Section 4. In Section 5 we propose symbolic dependency graphs and discuss how they are used for model checking in Section 6. Experimental results are presented in Section 7 Lastly, Section 8 concludes.

2 Weighted Kripke Structures

In this section we introduce weighted Kripke structures (WKS) and weighted computation tree logic (WCTL). We denote the set of natural numbers, including zero, by $\mathbb{N} = \{0, 1, 2, \ldots\}$.

Definition 1 (Weighted Kripke Structure). *A Weighted Kripke Structure (WKS) is a quadruple $\mathcal{K} = (S, \mathcal{AP}, L, \to)$, where*

- *S is a finite set of states,*
- *$\mathcal{AP}$ is a finite set of atomic propositions,*
- *$L : S \to \mathcal{P}(\mathcal{AP})$ is a mapping from states to sets of atomic propositions, and*
- *$\to \subseteq S \times \mathbb{N} \times S$ is a transition relation.*

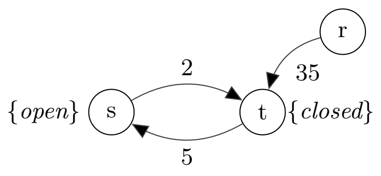


Fig. 1: A WKS that models a mechanical window that can open or close. It consumes 5 units of power when it goes to the *open* state and 2 power units to enter the *closed* state.

Example 1 Figure 1 shows a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, where

$$\begin{aligned} S &= \{s, t, r\} \\ \mathcal{AP} &= \{\mathit{open}, \mathit{closed}, \mathit{bad}\} \\ L &= \{s \mapsto \{\mathit{open}\}, t \mapsto \{\mathit{closed}\}, r \mapsto \{\mathit{bad}\}\} \\ \rightarrow &= \{(s, 2, t), (t, 5, s), (r, 35, t)\} \end{aligned}$$

The WKS models a window opener with the three states s, t and r. The model has atomic propositions *open*, *closed* and *bad* indicating the state of the window controlled by the window opener. The weights on the transition indicates the amount of power required to perform the transition. Notice that it requires less power to close the window, than it requires to open it.

If $(s, w, s') \in \rightarrow$ we say that s evolves to s' with weight w, denoted $s \xrightarrow{w} s'$. A WKS is said to be non-blocking if for any $s \in S$, there exists an s', such that $s \xrightarrow{w} s'$. For simplicity we shall assume that all WKSs are non-blocking. By introducing a dead state with no atomic propositions with a zero-weight self-loop and adding a zero-weight transition from every terminal state to the dead state, any WKS can be rendered non-blocking.

Definition 2 (Run). *Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, a run is an infinite path σ in $\mathcal{K}$.*

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

A position $p \in \mathbb{N}$ is the index of state s along σ and its weight is $W_\sigma(p) = \Sigma_{i=0}^{p-1} w_i$. We write $\sigma(p)$ to denote that state s is at position p in σ, i.e. $\sigma(i) = s_i$.

Definition 3 (Weighted Computation Tree Logic). *The set of weighted computation tree logic (WCTL) formulae over a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ is given by the following grammar.*

$$\begin{aligned} \varphi ::= &\ \boldsymbol{true} \mid \boldsymbol{false} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \\ &\ E\ \varphi_1\ U_{\leq k}\ \varphi_2 \mid A\ \varphi_1\ U_{\leq k}\ \varphi_2 \mid EX_{\leq k}\ \varphi \mid AX_{\leq k}\ \varphi \end{aligned}$$

where $k \in \mathbb{N}$ and $a \in \mathcal{AP}$.

Note that the negation-free CTL fragment expressing safety properties is subsumed by WCTL, because we may also express regular CTL properties for unweighted models using this logic. For instance, the CTL formula $EX\varphi$ has the form $EX_{\leq 0}\ \varphi$ in WCTL.

2.1 Semantics

We write $s \models \varphi$, if state s in $\mathcal{K}$ satisfies WCTL formula φ. The satisfaction relation is defined inductively as follows.

$s \models \mathbf{true}$	
$s \models a$	iff $a \in L(s)$
$s \models \varphi_1 \wedge \varphi_2$	iff $s \models \varphi_1$ and $s \models \varphi_2$
$s \models \varphi_1 \vee \varphi_2$	iff $s \models \varphi_1$ or $s \models \varphi_2$
$s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$	iff there exists a run σ starting from s and a position $p \geq 0$ s.t. $\sigma(p) \models \varphi_2, W_\sigma(p) \leq k$ and $\sigma(p') \models \varphi_1$ for all $p' < p$
$s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$	iff for any run σ starting from s, there exists a position $p \geq 0$ s.t. $\sigma(p) \models \varphi_2, W_\sigma(p) \leq k$ and $\sigma(p') \models \varphi_1$ for all $p' < p$
$s \models EX_{\leq k}\ \varphi$	iff $\exists s'$ s.t. $s \xrightarrow{w} s', s' \models \varphi$ and $w \leq k$
$s \models AX_{\leq k}\ \varphi$	iff $\forall s'$ s.t. $s \xrightarrow{w} s'$ where $w \leq k$ it holds that $s' \models \varphi$

3 Dependency Graphs

In this section we present the dependency graph framework and a local algorithm for minimal fixed point evaluation, both of which were originally introduced by Liu and Smolka in [7]. This framework can be applied to model checking of the alternation-free modal mu-calculus, hence it may also be used for model checking CTL. In section 4 we demonstrate how to model check a WKS with WCTL using this framework.

Definition 4 (Dependency Graph). *A dependency graph $G = (V, E)$ is a pair, where*

- *V is a finite set of configurations, and*
- *$E \subseteq V \times \mathcal{P}(V)$ is a 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 set of e. The set of successors $succ(v) = \{(v, T) \in E\}$ of a configuration v is the set of hyper-edges with v as the source configuration. The size of a dependency graph $G = (V, E)$ is denoted $|G|$ and formally defined as follows.

$$|G| = |V| + \sum_{(v,T)\in E} (|T| + 1)$$

An assignment $A : V \to \{0, 1\}$ is a function that assigns values to configurations of G. A post fixed-point assignment F of G is an assignment such that for every configuration $v \in V$, if $(v, T) \in E$ and for all $u \in T$ it holds that $F(u) = 1$, then $F(v) = 1$.

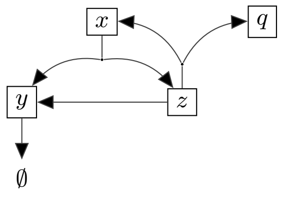
By taking the standard partial order $\sqsubseteq$ between assignments, such that $A \sqsubseteq A'$ if and only if $A(v) \leq A'(v)$ for all $v \in V$ (using the usual order $0 < 1$), then by the Knaster-Tarski fixed-point theorem, there exists a unique minimum post-fixed point assignment, denoted F_{min}.

The minimum post fixed-point assignment F_{min} of G can be computed by repeated application of the following functor, starting from $\mathcal{F}^0(v) = 0$ for all $v \in V$.

$$\mathcal{F}^{i+1}(v) = \bigvee_{(v,T)\in E} \left(\bigwedge_{u\in T} \mathcal{F}^i(u) \right)$$

This is a monotonic function evaluated on a finite complete lattice, hence by the Knaster-Tarski fixed-point theorem we reach a fixed point after a finite number of iterations. Thus, there exists an $m \in \mathbb{N}$, such that $\mathcal{F}^m(v) = \mathcal{F}^{m+1}(v)$ for all $v \in V$, in which case we have $\mathcal{F}^m = F_{min}$ is the minimum post-fixed point assignment of G.

$x = y \land z$
$z = y \lor (x \land q)$
$y = \textbf{true}$



(a) A boolean equation system and its dependency graph encoding.

i	$\mathcal{F}^i(x)$	$\mathcal{F}^i(y)$	$\mathcal{F}^i(z)$	$\mathcal{F}^i(q)$
0	0	0	0	0
1	0	1	0	0
2	0	1	1	0
3	1	1	1	0

(b) Computation of the minimum post fixed-point on the dependency graph shown to the left using the functor.

Fig. 2: A boolean equation system with its dependency graph encoding. The table to the right lists the iterations of the functor for computing the minimum fixed point.

Example 2 Figure 2a shows a boolean equation system and the equivalent dependency graph. Configurations are illustrated as labeled squares and hyper-edges are drawn as lines, where every configuration in their respective target sets is pointed to by an arrow.

Every variable of the boolean equation system is encoded as a configuration, where every outgoing hyper-edge represents a disjunction of conjunctions over the variables in its target set. For the variable y which is trivially true, a hyper-edge with an empty target set is added.

The minimum post fixed-point assignment of the dependency graph in Figure 2a is $F_{min}(x) = 1$, $F_{min}(y) = 1$, $F_{min}(z) = 1$ and $F_{min}(q) = 0$.

Computing the minimum post-fixed assignment by repeated application of $\mathcal{F}$ takes $O(|G|^2)$ time. Liu and Smolka propose global and local algorithms, running in $O(|G|)$ time, for computing the minimum post-fixed point assignment of dependency graphs in [7]. For model checking applications we are often only interested in the minimum post-fixed point assignment $F_{min}(v)$ of a specific configuration $v \in V$. For this purpose, Liu and Smolka proposed a local algorithm, which is reproduced in Algorithm 1 with minor modifications. Note that the complexity of the local algorithm is still $O(|G|)$ (like the global algorithm), because in the worst case it may have to explore the entire graph to compute the fixed point.

Algorithm 1 maintains three data-structures throughout execution; the assignment A, the dependency set D for every configuration and a queue of hyper-edges W. The dependency set $D(v)$ for some configuration v maintains a list of hyper-edges that were processed under the assumption the $A(v) = 0$. If at some point the value of $A(v)$ changes to 1, these hyper-edges must be re-processed, since we may be able to force their respective source configurations to be assigned 1.

We have made some slight modifications to the algorithm. In particular, in line 15, we add the current hyper-edge e to the dependency set $D(u)$ of the successor configuration u, i.e. $D(u) = \{e\}$. The original pseudo code does not perform this step, rather the dependency set is assigned the empty set [7], in which case line 15 would read $D(u) = \emptyset$. We believe this is simply a typographical error on the authors' part. Without this modification the algorithm would not propagate correctly.

Algorithm 1: Liu-Smolka Local Algorithm

```
   Input: Dependency graph $G = (V, E)$ and initial configuration $v_0 \in V$
   Output: Minimum post fixed-point assignment of $v_0$, $F_{min}(v_0)$
1  let $A(v) = \bot$ for all $v \in V$
2  $A(v_0) = 0$
3  $D(v_0) = \emptyset$
4  $W = succ(v_0)$
5  while $W \neq \emptyset$ do
6      let $e = (v, T) \in W$
7      $W = W \setminus \{e\}$
8      if $\forall u \in T$ it holds that $A(u) = 1$ then
9          $A(v) = 1$
10         $W = W \cup D(v)$
11     else if $\exists u \in T$ where $A(u) = 0$ then
12         $D(u) = D(u) \cup \{e\}$
13     else if $\exists u \in T$ where $A(u) = \bot$ then
14         $A(u) = 0$
15         $D(u) = \{e\}$
16         $W = W \cup succ(u)$
17 return $A(v_0)$
```

Theorem 1 (Local Algorithm Correctness). *Given a dependency graph $G = (V, E)$ and an initial configuration $v_0 \in V$, Algorithm 1 computes the minimum post-fixed point assignment of v_0.*

Proof. Correctness of Algorithm 1 is proved in [7].

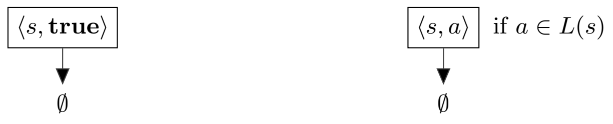
Theorem 2 (Local Algorithm Complexity). *Given an input dependency graph $G = (V, E)$ and initial configuration $v_0 \in V$, Algorithm 1 runs in $O(|G|)$ time.*

Proof. The complexity for Algorithm 1 is proved in [7].

4 Model Checking with Dependency Graphs

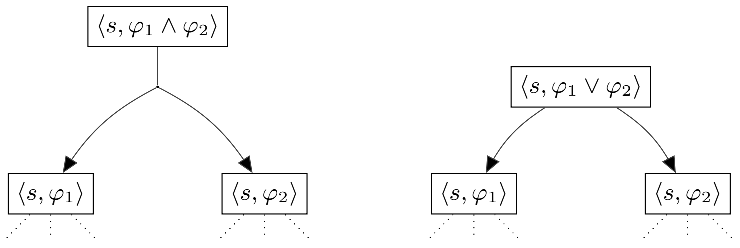
In this section we present an encoding of a WKS and a WCTL formula as a dependency graph. We then show how minimum post fixed-point computation of a dependency graph can be used to answer model checking questions.

Definition 5 (Formula Satisfaction Encoding). *Given WKS $\mathcal{K}$, state s and a WCTL formula φ, we can encode the model checking problem as a dependency graph. Every configuration in the graph consists of a state and a formula. The dependency graph is expanded from the initial configuration $\langle s, \varphi \rangle$ using the rules illustrated in Figures 3a, 3b, 4a, 4b, 5, 6, 7 and 8.*



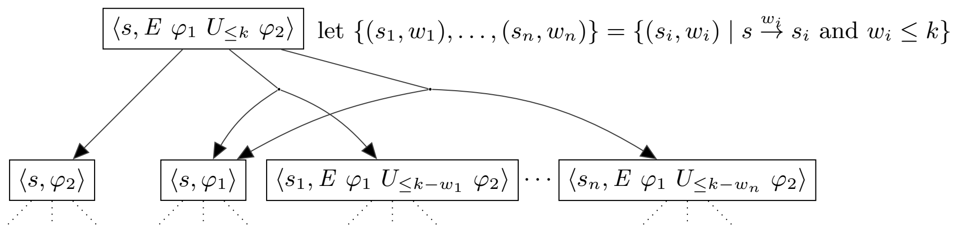
(a) A true configuration. (b) A configuration with an atomic proposition that holds.

Fig. 3: Configurations that are true have a hyper-edge with an empty target set.


$\langle s, \varphi_1 \wedge \varphi_2 \rangle$
$\langle s, \varphi_1 \rangle$
$\langle s, \varphi_2 \rangle$
$\langle s, \varphi_1 \vee \varphi_2 \rangle$
$\langle s, \varphi_1 \rangle$
$\langle s, \varphi_2 \rangle$

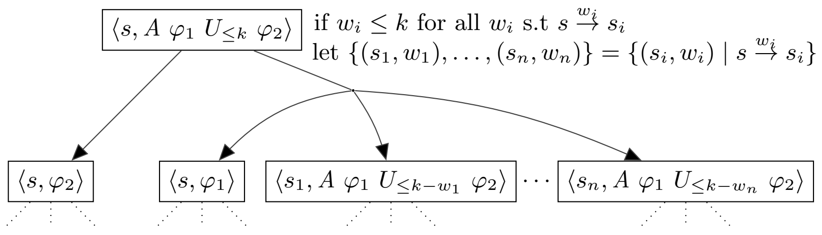
(a) A conjunctive configuration. (b) A disjunctive configuration.

Fig. 4: Successors of configurations with a composite logical operator.



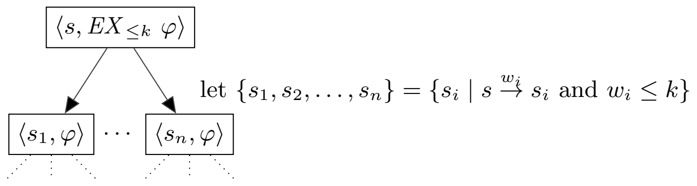
$\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle$
let $\{(s_1,w_1),\ldots,(s_n,w_n)\} = \{(s_i,w_i) \mid s \xrightarrow{w_i} s_i \text{ and } w_i \leq k\}$
$\langle s, \varphi_2\rangle$
$\langle s, \varphi_1\rangle$
$\langle s_1, E\ \varphi_1\ U_{\leq k-w_1}\ \varphi_2\rangle$
$\cdots$
$\langle s_n, E\ \varphi_1\ U_{\leq k-w_n}\ \varphi_2\rangle$

Fig. 5: Successors of a existential until configuration.



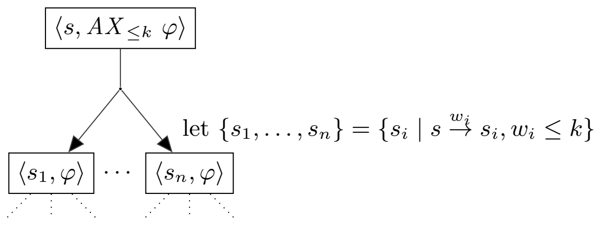
$\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle$
if $w_i \leq k$ for all w_i s.t $s \xrightarrow{w_i} s_i$
let $\{(s_1,w_1),\ldots,(s_n,w_n)\} = \{(s_i,w_i) \mid s \xrightarrow{w_i} s_i\}$
$\langle s, \varphi_2\rangle$
$\langle s, \varphi_1\rangle$
$\langle s_1, A\ \varphi_1\ U_{\leq k-w_1}\ \varphi_2\rangle$
$\cdots$
$\langle s_n, A\ \varphi_1\ U_{\leq k-w_n}\ \varphi_2\rangle$

Fig. 6: Successors of a universal until configuration. Notice that the set of successors of s is never empty, as we are only considering non-blocking WKSs.



$\langle s, EX_{\leq k}\ \varphi\rangle$
let $\{s_1,s_2,\ldots,s_n\} = \{s_i \mid s \xrightarrow{w_i} s_i \text{ and } w_i \leq k\}$
$\langle s_1, \varphi\rangle$
$\cdots$
$\langle s_n, \varphi\rangle$

Fig. 7: Successors of an existential next configuration.



$\langle s, AX_{\leq k}\ \varphi\rangle$
let $\{s_1,\ldots,s_n\} = \{s_i \mid s \xrightarrow{w_i} s_i, w_i \leq k\}$
$\langle s_1, \varphi\rangle$
$\cdots$
$\langle s_n, \varphi\rangle$

Fig. 8: Successors of a universal next configuration.

Theorem 3 (Encoding Correctness). *Let $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ be a WKS, $s \in S$ a state, φ a WCTL formula and G a dependency graph expanded from $\langle s, \varphi \rangle$ following Definition 5. We have that $s \models \varphi$ if and only if $F_{min}(\langle s, \varphi \rangle) = 1$.*

Proof. We prove Theorem 3 by structural induction on φ.

(I) For $\varphi = \textbf{true}$ we show that for all $s \in S$ we have $F_{min}(\langle s, \textbf{true} \rangle) = 1$ if and only if $s \models \textbf{true}$. But as $s \models \textbf{true}$ always holds, it is sufficient to show that $F_{min}(\langle s, \textbf{true} \rangle) = 1$ for any post fixed-point assignment F of G. In Figure 3a we add a hyper-edge from the configuration $\langle s, \textbf{true} \rangle$, to the empty target set. Thus, we have that $F(v) = 1$ for any post fixed-point assignment F of G, because all vertices in the empty set satisfy any property trivially.

(II) For $\varphi = a$ we prove that $F_{min}(\langle s, a \rangle) = 1$ if and only if $s \models a$ for all $s \in S$. If $a \in L(s)$ we have $s \models a$ and by Figure 3b, there is a hyper-edge from the configuration $\langle s, a \rangle$ to the empty target set. As in (I) this means that $F_{min}(\langle s, a \rangle) = 1$, which leaves us to consider $a \notin L(s)$. In this case we obviously have $s \not\models a$ and by the side-condition in Figure 3b, we can conclude that there is no hyper-edge from the configuration $\langle s, a \rangle$ when $a \notin L(s)$. Thus, we have $F_{min}(\langle s, a \rangle) = 0$ because F_{min} is the minimum post fixed-point assignment.

(III) For $\varphi = \varphi_1 \wedge \varphi_2$ we show that $F^G_{min}(\langle s, \varphi_1 \wedge \varphi_2 \rangle) = 1$ if and only if $s \models \varphi_1 \wedge \varphi_2$ for all $s \in S$. By Figure 4a, a configuration $\langle s, \varphi_1 \wedge \varphi_2 \rangle$ has a single hyper-edge with the target set $\{\langle s, \varphi_1 \rangle, \langle s, \varphi_2 \rangle\}$. With this observation it is easy to see that $F_{min}(\langle s, \varphi_1 \wedge \varphi_2 \rangle) = 1$ only holds if $F_{min}(\langle s, \varphi_1 \rangle) = 1$ and $F_{min}(\langle s, \varphi_2 \rangle) = 1$. By the induction hypothesis, $s \models \varphi_1$ and $s \models \varphi_2$, which following the semantics implies $s \models \varphi_1 \wedge \varphi_2$.

(IV) For $\varphi = \varphi_1 \vee \varphi_2$ we show that $F_{min}(\langle s, \varphi_1 \vee \varphi_2 \rangle) = 1$ if and only if $s \models \varphi_1 \vee \varphi_2$ for all $s \in S$. By Figure 4b, a configuration $\langle s, \varphi_1 \wedge \vee_2 \rangle$ has two hyper-edges with the target sets $\{\langle s, \varphi_1 \rangle\}$ and $\{\langle s, \varphi_2 \rangle\}$. With this observation, we have that $F_{min}(\langle s, \varphi_1 \vee \varphi_2 \rangle) = 1$ if and only if $F_{min}(\langle s, \varphi_1 \rangle) = 1$ or $F_{min}(\langle s, \varphi_2 \rangle) = 1$. By the induction hypothesis this is equivalent to $s \models \varphi_1$ or $s \models \varphi_2$, which following the semantics implies $s \models \varphi_1 \vee \varphi_2$.

(V) For $\varphi = E\ \varphi_1\ U_{\leq k}\ \varphi_2$ we show that $F_{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 1$ if and only if $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ for all $s \in S$. Recall the semantics for the satisfaction of formula $E\ \varphi_1\ U_{\leq k}\ \varphi_2$, requires that for some $k' \leq k$, there exists a run σ and a position $p \geq 0$ that satisfy the following conditions.

$$\sigma(p) \models \varphi_2 \tag{1}$$

$$\sigma(j) \models \varphi_1 \text{ , for all } j < p \tag{2}$$

$$W_\sigma(p) \leq k' \tag{3}$$

$\Rightarrow$: Assume that $F_{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 1$, we now show that this implies $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$.

We denote the iteration in which a configuration v was first assigned 1, as $Z(v)$, formally we write the auxiliary function Z as follows.

$$Z(v) = \begin{cases} i & \text{if } \mathcal{F}^i(v) \neq \mathcal{F}^{i-1}(v) \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

For any configuration v it holds that $Z(v) < \infty$ if and only if $F_{min}(v) = 1$, as a post fixed-point assignment must be reached in a finite number of iterations. Considering $Z(v)$ for a configuration $v = \langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$, where $F_{min}(v) = 1$, we see that in iteration $Z(v) - 1$, the assignment of some configuration in the target-set for a hyper-edge to v must have been changed to 1. From Figure 5 we see that there are two kinds of hyper-edges, leading us to conclude that at least one of the following two cases must hold.

A) $Z(\langle s, \varphi_2 \rangle) = Z(v) - 1$, or
B) $\max\{Z(\langle s, \varphi_1 \rangle), Z(\langle s', E\ \varphi_1\ U_{\leq k-w}\ \varphi_2 \rangle)\} = Z(v) - 1$, for some s', s.t. $s \xrightarrow{w} s'$.

We now show that $F_{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 1$ implies the existence of a run σ and a position p satisifying conditions 1, 2 and 3 for $k' \leq k$, by induction on $Z(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle)$.
First we observe that $Z(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle)$ is always greater than 1, as only configurations v having trivial hyper-edges $(v, \emptyset)$ are assigned 1 in the first iteration of $\mathcal{F}$.
Base Case ($Z(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 2$): In this case we know that case (A) must hold, seeing that no configuration $u = \langle s', E\ \varphi_1\ U_{\leq k-w}\ \varphi_2 \rangle$ can have $Z(u) = 1$. From case (A), we have that $Z(\langle s, \varphi_2 \rangle) = 1$, which means that $F_{min}(\langle s, \varphi_2 \rangle) = 1$. By structural induction, $F_{min}(\langle s, \varphi_2 \rangle) = 1$ gives us $s \models \varphi_2$. Thus, any run $\sigma = s \ldots$ and position $p = 0$ satisfy conditions 1, 2 and 3 for $k' = 0$, hence, it also holds for $k' \leq k$.
Inductive Step ($Z(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) > 2$): Again, we consider cases (A) and (B). If case (A) holds we can construct a run $\sigma = s \ldots$ and position $p = 0$ as before. If (B) is the case, we have that $F_{min}(\langle s, \varphi_1 \rangle) = 1$ and $F_{min}(\langle s', E\ \varphi_1\ U_{\leq k-w}\ \varphi_2 \rangle) = 1$. By structural induction it follows from $F_{min}(\langle s, \varphi_1 \rangle) = 1$ that $s \models \varphi_1$.
Because $Z(\langle s', E\ \varphi_1\ U_{\leq k-w}\ \varphi_2 \rangle) < Z(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle)$ it follows by induction that there is a run $\sigma = s' \ldots$ and a position p that satisfy conditions 1, 2 and 3 for $k' \leq k - w$. Considering the extension $\sigma' = s \xrightarrow{w} s' \ldots$ of σ and position $p' = p + 1$, we observe that σ' and p' also satisfy the conditions for $k' \leq k$.

- Condition 1 holds because $\sigma'(p') = \sigma(p)$ and $\sigma(p) \models \varphi_2$.
- Condition 2 holds since $\sigma(0) = s$, $s \models \varphi_1$ and for all $j < p$ we have $\sigma'(j+1) = \sigma(j)$ and $\sigma(j) \models \varphi_1$.
- Condition 3 holds due to the fact that $W_\sigma(p) \leq k - w$ implies $W_{\sigma'}(p') \leq k$, because $W_{\sigma'}(p') - W_\sigma(p) = w$.

$\Leftarrow$: Assume that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$, we now show that this implies $F_{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 1$. From the semantics it follows that there is a run σ and position p satisfying conditions 1, 2 and 3 for $k' \leq k$ Let $s = s_0$, then we can write σ as follows.

$$\sigma = s_0 \overset{w_1}{\to} s_1 \dots s_{p-1} \overset{w_p}{\to} s_p \dots$$

We show that $F_{min}(\langle s_i, E\ \varphi_1\ U_{\leq k - W_\sigma(i)}\ \varphi_2\rangle) = 1$ by induction on i starting from p.

Base Case ($i = p$): By condition 1 of the semantics, $s_p \models \varphi_2$, which by structural induction on φ implies $F_{min}(\langle s_p, \varphi_2\rangle) = 1$. In Figure 5, we observe that there is a hyper-edge from $\langle s_p, E\ \varphi_1\ U_{\leq k-W_\sigma(i)}\ \varphi_2\rangle$ to $\langle s_p, \varphi_2\rangle$, thus, $F_{min}(\langle s_p, \varphi_2\rangle) = 1$ implies $F_{min}(\langle s_p, E\ \varphi_1\ U_{\leq k-W_\sigma(i)}\ \varphi_2\rangle) = 1$, which proves our base case.

Inductive Step ($i < p$): By condition 2 of the semantics, $s_i \models \varphi_1$, which by structural induction on φ implies $F_{min}(\langle s_i, \varphi_1\rangle) = 1$. By induction on i, we know that $F_{min}(\langle s_{i+1}, E\ \varphi_1\ U_{\leq k-W_\sigma(i+1)}\ \varphi_2\rangle) = 1$ holds. In Figure 5, we observe that there is a hyper-edge e from $\langle s_i, E\ \varphi_1\ U_{\leq k-W_\sigma(i)}\ \varphi_2\rangle$ to the target-set $\langle s_i, \varphi_1\rangle$ and $\langle s_{i+1}, E\ \varphi_1\ U_{\leq k-W_\sigma(i+1)}\ \varphi_2\rangle$, as $W_\sigma(i+1)-W_\sigma(i) = w_{i+1}$, which is exactly the transition weight between s_i and s_{i+1}. Since we know that $F_{min}(v) = 1$ for all configurations v of the target-set of the hyper-edge e, then it must follow that $F_{min}(\langle s_i, E\ \varphi_1\ U_{\leq k-W_\sigma(i)}\ \varphi_2\rangle) = 1$ for all $i \leq p$.

(VI) For $\varphi = A\ \varphi_1\ U_{\leq k}\ \varphi_2$ we have that $F_{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ if and only if $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$ for all $s \in S$. The proof strategy here is similar to the previously shown case for $\varphi = E\ \varphi_1\ U_{\leq k}\ \varphi_2$.
$\Rightarrow$: Assume that $F_{min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$, we now show that this implies $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$.
$\Leftarrow$: Assume that $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$, we now show that this implies $F_{min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$.

(VII) For $\varphi = EX_{\leq k}\ \varphi$ we show that $F_{min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$ if and only if $s \models EX_{\leq k}\ \varphi$ for all $s \in S$.
$\Rightarrow$:
Assume that $F_{min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$, then it holds that $s \models EX_{\leq k}\ \varphi$. In Figure 7, the configuration $\langle s, EX_{\leq k}\ \varphi\rangle$ has a hyper-edge for every $s_i \in \{s_i \mid s \overset{w_i}{\to} s_i \text{ and } w_i \leq k\}$. Clearly, $F_{min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$ if and only if $F_{min}(\langle s_i, \varphi\rangle) = 1$ is the case for any such s_i. By the induction hypothesis this is equivalent to $s_i \models \varphi$, which following the semantics implies that $s \models EX_{\leq k}\ \varphi$.
$\Leftarrow$:
Assume that $s \models EX_{\leq k}\ \varphi$, then it holds that $F_{min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$. From the semantics, it must be the case that there exists an s_i, such that $s \overset{w_i}{\to} s_i$, with $w_i \leq k$, it holds that $s_i \models \varphi$. By the induction hypothesis, this implies that $F_{min}(\langle s_i, \varphi\rangle) = 1$ for any such s_i. Since F_{min} is a minimum post fixed-point assignment the manner Figure 7 is constructed, we have that $F_{min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$.

(VIII) For $\varphi = AX_{\leq k}\ \varphi$ we show that $F_{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$ if and only if $s \models AX_{\leq k}\ \varphi$ for all $s \in S$.

$\Rightarrow$:
Assume that $F_{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$, then it holds that $s \models AX_{\leq k}\ \varphi$. In Figure 8, the configuration $\langle s, AX_{\leq k}\ \varphi\rangle$ has a single hyper-edge with a target set on the form $\{\langle s_1, \varphi\rangle, \dots, \langle s_n, \varphi\rangle\}$, for every s_i, such that $s \overset{w_i}{\to} s_i$ and $w_i \leq k$. It is clear that $F_{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$ if and only if $F_{min}(\langle s_i, \varphi\rangle) = 1$ for all such s_i. Given the induction hypothesis, we have that $s_i \models \varphi$ for $1 \leq i \leq n$, which implies that $s \models AX_{\leq k}\ \varphi$.
$\Leftarrow$:
Assume that $s \models AX_{\leq k}\ \varphi$, then it holds that $F_{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$. By the semantics it must be that case that $s_i \models \varphi$, for all s_i such that $s \overset{w_i}{\to} s_i$, where $w_i \leq k$. By the induction hypothesis this implies that $F_{min}(\langle s_i, \varphi\rangle) = 1$ for all such s_i. Since F_{min} is a minimum post fixed-point assignment and due to the construction in Figure 8, we have that $F_{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$.

□

Notice that the dependency graph for a WKS $\mathcal{K}$ and formula φ can be expanded on-the-fly when successor configurations are first considered, i.e. *succ*(v) used on line 4 and 16 in Algorithm 1. Configurations that have not yet been observed are assigned the unknown value, $\perp$.

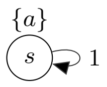


Fig. 9: A simple WKS that may yield a large unfolding.

Example 3 Consider the dependency graph shown in Figure 9 and the formula $\varphi = E\ a\ U_{\leq 1000}\ b$. In Figure 10 we see that the size of the dependency graph grows in proportion to the initial bound on the formula and every intermediate configuration reduces the bound by 1. This suggests that the size of the dependency graph encoding is pseudo-polynomial in the size of φ.

The size of a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \to)$ is denoted by $|\mathcal{K}|$. It is the sum of the number of states, atomic propositions, labeling of states and size of the transition system, $|S| + \sum_{s \in S} |L(s)| + |\mathcal{AP}| + |\to|$. The size of a WCTL formula φ is denoted $|\varphi|$. The maximum bound $\mathcal{B}(\varphi)$ of φ is the maximum bound k appearing in an existential or universal "until" sub-formula in φ.

Theorem 4. *Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \to)$, a state $s \in S$ and a WCTL formula φ, the size of the dependency graph encoding of $s \models \varphi$ from Definition 5 is $O(|\mathcal{K}| \cdot |\to| \cdot |\varphi| \cdot \mathcal{B}(\varphi))$.*

Proof. If we recall that all configurations in the encoding consist of a state and formula, it easy to see that the expansions in Figures 3a, 3b, 4a, 4b, 7 and 8

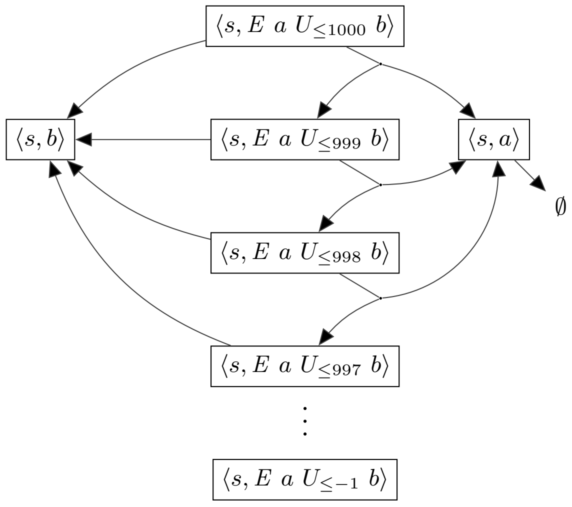

$\langle s, E\ a\ U_{\leq 1000}\ b\rangle$
$\langle s, b\rangle$
$\langle s, E\ a\ U_{\leq 999}\ b\rangle$
$\langle s, a\rangle$
$\emptyset$
$\langle s, E\ a\ U_{\leq 998}\ b\rangle$
$\langle s, E\ a\ U_{\leq 997}\ b\rangle$
$\langle s, E\ a\ U_{\leq -1}\ b\rangle$

Fig. 10: The dependency graph for the formula φ and example WKS in Figure 9.

give rise to at most $O(|\mathcal{K}| \cdot |\varphi|)$ configurations. This follows from the fact that successors in these expansions consist of a state and a sub formula of the source formula.

This leaves us to consider the expansions from Figures 5 and 6. In these expansions we have successor configurations with formulae that are not sub-formulae, however, in this case the bound of the formula is reduced by some non-negative weight. With $\mathcal{B}(\varphi)$ as a bound on the value of k in the existential and universal "until" expression, we have that these expansions give rise to at most $O(|\mathcal{K}| \cdot |\varphi| \cdot \mathcal{B}(\varphi))$.

As for the number of hyper-edges leaving a configuration $\langle s, \varphi\rangle$ in the dependency graph, we have that this is bounded by the number of outgoing edges from s. However, we can have multiple weighted edges between two states, so for a configuration, the number of hyper-edges is bounded by $|{\rightarrow}|$. Thus, we conclude that the size dependency graph is $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi| \cdot \mathcal{B}(\varphi))$. $\square$

Corollary 1. *Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, a state $s \in S$ and a WCTL formula φ, $s \models \varphi$ can be decided $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi| \cdot \mathcal{B}(\varphi))$ time.*

Proof. To decide $s \models \varphi$ we encode the problem using Definition 5, from Theorem 4 the resulting dependency graph has a size of $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi| \cdot \mathcal{B}(\varphi))$. By Theorem 2 Algorithm 1 we can compute the minimum post fixed-point assignment of $\langle s, \varphi\rangle$ in $O(|G|)$ time. Thus, as $|G| = O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi| \cdot \mathcal{B}(\varphi))$, we can decide $s \models \varphi$ in $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi| \cdot \mathcal{B}(\varphi))$ time. Correctness of this approach follows from Theorems 1 and 3. $\square$

Besides having an efficient algorithm, it is important to establish the complexity of the model checking problem. Because the encoding presented in this section is pseudo-polynomial, we cannot conclude that the WCTL model checking problem is decidable in polynomial time. Notwithstanding, we shall later show that the problem is, indeed, decidable in polynomial time.

5 Symbolic Dependency Graphs

With dependency graphs, we saw by example that it is easy to devise an "until" formula such that the size of the dependency graph encoding grows in proportion to the bound. In Example 3 this happens because we need to establish whether $s \models E\ a\ U_{\leq k-1}\ b$ in order to determine if $s \models E\ a\ U_{\leq k}\ b$ for any bound k. However, if we consider these formulas it is easy to see that $s \models E\ a\ U_{\leq k}\ b$ implies $s \models E\ a\ U_{\leq k+1}\ b$ and $s \not\models E\ a\ U_{\leq k}\ b$ implies $s \not\models E\ a\ U_{\leq k-1}\ b$.

In Section 6 we propose an encoding that captures the implications between $s \models E\ a\ U_{\leq k}\ b$ and $s \models E\ a\ U_{\leq k'}\ b$ by synthesizing the bound k for which $s \models E\ a\ U_{\leq k}\ b$ and $s \not\models E\ a\ U_{\leq k-1}\ b$. This is accomplished by storing a number for each configuration in an assignment, with ∞ as the initial value, which then works its way down to the smallest possible bound k.

In the following we introduce symbolic dependency graphs with a more powerful semantics, that will allow us to encode the entire WCTL model-checking problem as a minimum post fixed-point computation on a (symbolic) dependency graph of polynomial size.

Definition 6 (Symbolic Dependency Graph). *A symbolic dependency graph (SDG) is a pair $G = (V, H, C)$, where*

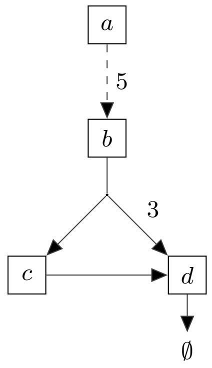
- *V is a finite set of configurations,*
- *$H \subseteq V \times \mathcal{P}(\mathbb{N} \times V)$ is a finite set of hyper-edges, and*
- *$C \subseteq V \times \mathbb{N} \times V$ is a finite set of cover-edges.*

Let $G = (V, H, C)$ be a symbolic dependency graph. For a hyper-edge $e = (v, T) \in H$ we call v the source configuration and T the target set of e, we say that $(w, u) \in T$ is a hyper-edge branch with weight w to target configuration u. For a cover-edge $(v, k, u) \in C$, we call v the source configuration, k the cover condition and u the target configuration. The 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.

An assignment $A : V \to \mathbb{N} \cup \{\infty\}$ is a mapping from configurations to values. We denote the set of all assignments $Assign = V \times \mathbb{N} \cup \{\infty\}$. A post fixed-point assignment is an assignment $A \in Assign$, such that $A = F(A)$ where $F : Assign \to Assign$ is defined as follows.

$$F(A)(v) = \begin{cases} 0 & \textbf{if } \exists (v, k, v') \in C \text{ s.t. } k \geq A(v') \\ & \textbf{otherwise} \\ \min\limits_{(v,T) \in H} & \max\{w + A(v') \mid (w, v') \in T\} \end{cases} \tag{5}$$

If we consider the partial order $\sqsubseteq$ over assignments of a symbolic dependency graph G, such that $A \sqsubseteq A'$ if and only if $A(v) \geq A'(v)$ for all $v \in V$. Then it follows from the Knaster-Tarski fixed-point theorem that there exists a unique minimum post fixed-point assignment of G, denoted F_{min}.



(a) Example of an SDG. Hyper-edges are solid lines. Hyper-edge branches have weight 0 unless if annotated with a number. Cover-edges are dashed lines annotated with a cover-condition.

i	$\mathcal{F}^i(a)$	$\mathcal{F}^i(b)$	$\mathcal{F}^i(c)$	$\mathcal{F}^i(d)$
0	∞	∞	∞	∞
1	∞	∞	∞	0
2	∞	∞	0	0
3	∞	3	0	0
4	0	3	0	0
5	0	3	0	0

(b) Computation of the minimum post fixed-point on the WKS from Figure 11a using the functor. In this example the minimum post fixed-point assignment is reached within four iterations.

Fig. 11: Symbolic dependency graph and table with intermediate assignments from minimum post fixed-point computation by repeated functor application.

Notice that we say an assignment A is smaller than another assignment A', if values assigned by A are greater than or equal to those assigned by A'. Thus, we have $A_{min}(v) = \infty$ for all $v \in V$, is the smallest possible assignment.

The minimum post fixed-point assignment F_{min} of G can be computed by repeated application of the functor $\mathcal{F}^{i+1} = F(\mathcal{F}^i)$ starting from $\mathcal{F}^0(v) = \infty$ for all $v \in V$.

This is a monotonic function and although evaluated on an infinite complete lattice, there is no infinite decreasing sequence natural numbers, hence, we must reach a fixed point after a finite number of iterations. Thus, there exists an $m \in \mathbb{N}$ such that $\mathcal{F}^m(v) = \mathcal{F}^{m+1}(v)$ for all $v \in V$, in which case we have $\mathcal{F}^m = F_{min}$ is the minimum post-fixed point assignment of G.

Example 4 Figure 11a shows a symbolic dependency graph $G = (V, H, C)$, where

$$\begin{aligned} V &= \{a, b, c, d\} \\ H &= \{(b, \{(0, c), (3, d)\}), (d, (0, \emptyset)\}), (c, \{(0, d)\})\} \\ C &= \{(a, 5, b)\} \end{aligned}$$

The intermediate results during computation of the minimum post-fixed point assignment are displayed in Table 11b. From this table we can see that the minimum post fixed-point assignment is reached within four iterations. This follows from the fact that $\mathcal{F}^4 = \mathcal{F}^5$ in this example. Notice while $\mathcal{F}^i(v) \geq \mathcal{F}^{i+1}(v)$, we have $\mathcal{F}^i \sqsubseteq \mathcal{F}^{i+1}$ as we move up the lattice.

Lemma 1. *Let $G = (V, H, \emptyset)$ be an SDG without cover-edges and c_i denote a configuration which assignment changed to the smallest value in the i'th iteration of the functor, formally written as follows.*

$$c_i = \underset{v \in \{v \in V \mid \mathcal{F}^{i-1}(v) > F^i(v)\}}{\arg\min} \mathcal{F}^i(v)$$

It holds that $\mathcal{F}^i(c_i) = F_{min}(c_i)$.

Proof. To prove that $F_{min}(c_i) = \mathcal{F}^i(c_i)$, we show that Equation (13) holds. It then trivially follows that $\mathcal{F}^i(c_i)$ is the minimum post fixed-point assignment of c_i, because no future smallest assignment in any iteration $j > i$ becomes less than $\mathcal{F}^i(c_i)$.

To show that Equation (13) holds, we observe that when the assignment of configuration c_{i+1} is changed to the smallest value in the $i+1$'th iteration, then its assignment must have become smaller in iteration $i+1$, written as Equation (6).

$$\mathcal{F}^i(c_{i+1}) > \mathcal{F}^{i+1}(c_{i+1}) \tag{6}$$

$$\mathcal{F}^{i+1}(c_{i+1}) = \max\{w' + \mathcal{F}^i(u') \mid (w', u') \in T\} \tag{7}$$

$$\mathcal{F}^{i-1}(u) > \mathcal{F}^i(u) \tag{8}$$

$$\mathcal{F}^i(u) \geq \mathcal{F}^i(c_i) \tag{9}$$

This implies that there exists a hyper-edge $(c_{i+1}, T) \in H$ such that Equation (7) holds. Because the value $\mathcal{F}^{i+1}(c_{i+1})$ was not reached in the i'th iteration, there must be a hyper-edge branch $(w, u) \in T$ such that the assignment of configuration u changed from the $i-1$'th to the i'th iteration, which yields Equation (8).

We know that the smallest assignment changed from the $i-1$'th to the i'th iteration is $F^i(c_i)$. Hence, we get Equation (9), because no other assignment made in the i'th iteration is smaller than $F^i(c_i)$.

$$\max\{w' + \mathcal{F}^i(u') \mid (w', u') \in T\} \geq w + F^i(u) \tag{10}$$

$$F^{i+1}(c_{i+1}) \geq w + F^i(u) \tag{11}$$

$$F^{i+1}(c_{i+1}) \geq w + F^i(c_i) \tag{12}$$

$$F^{i+1}(c_{i+1}) \geq F^i(c_i) \tag{13}$$

As the hyper-edge branch (w, u) for which the value of u changed is in T, we observe that $w + \mathcal{F}^i(u)$ must be less than equal to the right hand side of Equation (7) giving us Equation (10). Substituting this back into Equation (7) and we get Equation (11). We now recall the lower-bound on $\mathcal{F}^i(u)$ from Equation (9) in order to write Equation (12). Thus, we get Equation (13) as w must be non-negative. □

Theorem 5 (Complexity of Functor Application). *Computing the minimum post fixed-point assignment by repeated application of the functor can be done in $O(|V|)$ iterations and takes $O(|V|^3 \cdot |C| + |V|^4 \cdot |H|)$ time.*

Proof. Computing a single iteration of the functor takes $O(|V| \cdot |C| + |V|^2 \cdot |H|))$ time, leaving us to show that the minimum post fixed-point assignment is reached within $O(|V|^2)$ iterations.

If we consider a symbolic dependency graph without cover-edges $G = (V, H, \emptyset)$, we have that the minimum post-fixed point assignment is reached within $|V|$ iterations. This follows from Lemma 1 which states that for each iteration, there is at least one configuration that reaches its minimum post fixed point assignment.

Now consider a symbolic dependency graph $G = (V, H, C)$, where only one configuration v has cover-edges, i.e. $C \subseteq \{(v, k, u) \mid k \in \mathbb{N} \text{ and } u \in V\}$. In this, the case a cover-edge can change $A(v)$ to zero, however, this can only happen once. If this does not occur, we have the minimum post fixed-point assignment after $|V|$ iterations. However, should it happen that $A(v)$ is set to zero by a cover-edge within $|V|$ iterations, no other cover-edge can affect the computation (i.e. the value of $A(v)$) any further, and we have the minimum post fixed-point assignment $|V|$ iterations later. Thus, after $|V| + |V|$ iterations, we must have the minimum post fixed-point assignment.

For the general case of symbolic dependency graphs, where any configuration may have cover-edges, we consider that if no cover-edge (v, k, u) changes $A(v)$ to zero, then we reach the minimum post fixed-point assignment in $|V|$ iterations. However, if $A(v)$ is set to zero, then no cover-edge can affect $A(v)$ again, as it remains zero. We see that cover-edges can affect the computation at most $|V|$ configurations. Thus, after $|V|^2$ iterations we must have the minimum post fixed-point assignment.

Hence, we conclude that after $O(|V|^2)$ iterations, the minimum post fixed-point assignment must be reached. Thus, we have shown that the minimum post fixed-point assignment can be computed by repeated application of the functor in $O(|V|^3 \cdot |C| + |V|^4 \cdot |H|)$ time. $\square$

We now propose a local algorithm for minimum post fixed-point computation on symbolic dependency graphs. Given a symbolic dependency graph $G = (V, H, C)$, Algorithm 2 computes the minimum post fixed-point assignment $F_{min}(v_0)$ of a configuration $v_0 \in V$. This algorithm is an adoption of Algorithm 1, by Liu and Smolka, for minimum post fixed-point assignment computation on dependency graphs.

Algorithm 2 uses the same data-structures as Algorithm 1. However, the assignment A ranges over $\mathbb{N} \cup \{\perp, \infty\}$, where $\perp$ once again indicates that the value is unknown at a particular configuration.

Algorithm 2: Symbolic Local Algorithm

```
Input: An SDG $G = (V, H, C)$ and initial configuration $v_0 \in V$
Output: Minimum post fixed-point assignment of $v_0$, $F_{min}(v_0)$
Let $A(v) = \bot$ for all $v \in V$
$A(v_0) = \infty$
$W = succ(v_0)$
while $W \neq \emptyset$ do
    Pick $e \in W$
    $W = W \setminus \{e\}$
    if $e = (v, T)$ is a hyper-edge then
        if $\exists (w, u) \in T$ where $A(u) = \infty$ then
            $D(u) = D(u) \cup \{e\}$
        else if $\exists (w, u) \in T$ where $A(u) = \bot$ then
            $A(u) = \infty$
            $D(u) = \{e\}$
            $W = W \cup succ(u)$
        else
            $a = \max\{A(u) + w \mid (w, u) \in T\}$
            if $a < A(v)$ then
                $A(v) = a$
                $W = W \cup D(v)$
            let $(w, u) = \arg\max_{(w,u) \in T} A(u) + w$
            if $A(u) > 0$ then
                $D(u) = D(u) \cup \{e\}$
    else if $e = (v, k, u)$ is a cover-edge then
        if $A(u) = \bot$ then
            $A(u) = \infty$
            $D(u) = \{e\}$
            $W = W \cup succ(u)$
        else if $k \geq A(u)$ then
            $A(v) = 0$
            if $A(v)$ was changed then
                $W = W \cup D(v)$
        else
            $D(u) = D(u) \cup \{e\}$
return $A(v_0)$
```

Example 5 Table 1 lists the values of the assignment A, the queue W and dependency set D while executing Algorithm 2 on the SDG from Example 4 (illustrated in Figure 11a). Each row displays the values for a given iteration, e.g. row 1 contains that values upon first entering the while-loop of Algorithm 2. The value of the dependency set $D(a)$ for a is not shown in the table because it remains empty.

Notice that the values of A are strictly non-increasing as the number of iterations increases, and the final value of A is the minimum post fixed-point assignment for the configurations that were examined, in order to determine the value of $F_{min}(a)$.

	$A(a)$	$A(b)$	$A(c)$	$A(d)$	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		$(a,5,b)$	$(b,\{(0,c),(3,d)\})$	$(c,\{(0,d)\})$

Table 1: Execution steps of Algorithm 2 when executed on the symbolic dependency graph from Example 4 illustrated in Figure 11a.

Lemma 2. *For every configuration $v \in V$, the while-loop in Algorithm 2 has the following invariants.*

1) If $A(v) \neq \bot$ then $A(v) \geq F_{min}(v)$,
2) If $A(v) \neq \bot$ and $e = (v, T) \in H$, then either
  a) $e \in W$,
  b) $e \in D(u)$ and $A(v) \leq x$ for some $(w, u) \in T$ s.t. $x = A(u) + w$, where $x \geq A(u') + w'$ for all $(w', u') \in T$, 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 prove the invariants with the inductive argument that if the invariant holds at the beginning of the every iteration of the while-loop, it also holds at the end of every iteration.

Invariant (1): Initially, we have that $A(v) = \bot$, for all $v \in V \setminus \{v_0\}$, and $A(v_0) = \infty$ for the initial configuration v_0. Hence, the invariant holds trivially the first time the while-loop is entered.

We observe that the assignment A is only updated in lines 11, 17, 24 and 28 of Algorithm 2. From this, there are three different cases to consider regarding the updated value of A.

In lines 11 and 24, $A(v)$ is assigned the value ∞. Because $A(v) = \infty \geq F_{min}(v)$, it is clear that the invariant holds.

In line 17, $A(v)$ is assigned the value $\max\{A(u) + w \mid (w, u) \in T\}$ for a hyper-edge (v, T), if the value of this expression is strictly smaller than the current value of the assignment of v. This corresponds to the "otherwise" case of the functor in Equation 5, hence the invariant holds.

In line 28, $A(v)$ is assigned the value 0, if there exists a cover-edge (v, k, u) where $A(u) \leq k$, which corresponds to the first case of the functor in Equation 5. Thus, we have shown that invariant (1) holds.

Invariants (2) and (3): The two invariants hold initially, because $A(v) = \perp$ for all $v \in V \setminus \{v_0\}$ and for the initial configuration v_0, we have that $W = succ(v_0)$. So, every hyper-/cover-edge with the source configuration v_0 is in W, which gives rise to cases (2a) and (3a).

We observe that, whenever a hyper-/cover-edge e is removed from W, it is added to the dependency set $D(u)$ of a target configuration u in e, unless it is the case that $A(v)$ has the value 0. With this observation, and the fact that when we explore a new configuration u by setting $A(u) = \infty$, we always add $succ(u)$ to W. It is easy to see that invariants 2 and 3 hold. ☐

Theorem 6 (Algorithm 2 Termination). *Algorithm 2 terminates.*

Proof. The while-loop in Algorithm 2 finishes when the queue W is empty ($W = \emptyset$), resulting in the termination of Algorithm 2. To prove that this eventually occurs, we 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 or $A(v)$ changes from $\perp$ to ∞. Moreover, we notice that $A(v)$ is always non-increasing and once the value of $A(v)$ changes from $\perp$, it is never assigned the value $\perp$ again. Due to the fact that it is always the case that $A(v) \geq 0$, then it follows that in Algorithm 2, the cover-/hyper-edges are only added to W a finite number of times. Thus, Algorithm 2 must terminate. ☐

Theorem 7 (Algorithm 2 Correctness). *Upon termination of Algorithm 2 on the input a symbolic dependency graph $G = (V, H, C)$, it holds that $A(v) \neq \perp$ implies $A(v) = F_{min}(v)$ for all $v \in V$.*

Proof. We prove correctness of Algorithm 2 by examining the cases of Lemma 2.

From invariant (1) of Lemma 2, we have that for all $v \in V$, where $A(v) \neq \perp$, it holds that $A(v) \geq F_{min}(v)$, leaving us to show that $A(v)$ is also a post fixed-point assignment.

To prove that $A(v)$ is post fixed-point assignment for all $v \in V$ where $A(v) \neq \perp$, we must show that $A(v) = F(A)(v)$. From the definition of the functor (Equation 5) we see that the two following cases must be considered.

i) If there exists $(v, k, u) \in C$ and $A(u) \leq k$, then $A(v) = 0$.

ii) For any $(v, T) \in H$ and $x = \max\{A(u') + w' \mid (w', u') \in T\}$, then $A(v) \leq x$.

First we consider **case (i)**. We prove by contradiction that $A(v) = 0$. Assume that $A(v) > 0$. Considering invariant 3, we observe that the algorithm has terminated, thus, $W = \emptyset$ and case 3a cannot hold. This means that either case 3b or case 3c must hold. Since $A(v) \neq 0$, we know that case 3c does not hold, leaving us to conclude that case 3b holds. By case 3b, we have $A(u) > k$, which contradicts $A(u) \leq k$. Thus, we must have $A(v) = 0$, proving (i).

For **case (ii)**. We prove by contradiction that $A(v) \leq x$. Assume that $A(v) > x$. Considering invariant 2, we observe as before that the algorithm has terminated, thus, $W = \emptyset$ and case 2a cannot hold. Hence, either case 2b or case 2c must hold. Because $x \geq 0$ and we assumed $A(v) > x$, it must be the case that $A(v) \neq 0$, so we know that case 2c cannot hold. This leaves us with case 2b, which by Lemma 2 must hold.

By case 2b, we have that there exists a hyper-edge branch $(w, u) \in T$, s.t. $A(v) \leq x'$, where $x' = A(u) + w$ and $x' \geq A(u') + w'$ for all $(w', u') \in T$. As both x and x' are the maximum value of the set $\{A(u') + w' \mid (w', u') \in T\}$, it must be the case that $x = x'$. Thus, $A(v) \leq x'$ contradicts our assumption that $A(v) > x$. Therefore, it must be the case that $A(v) \leq x$.

Consequently, we conclude that upon termination of Algorithm 2, it holds that for all $v \in V$, where $A(v) \neq \perp$, the assignment $A(v)$ is the minimum post fixed-point assignment of v. □

Corollary 2. *Given a symbolic dependency graph $G = (V, H, C)$ and an initial configuration $v_0 \in V$, Algorithm 2 computes the minimum post fixed-point assignment of v_0 in G.*

Proof. We only assign $\perp$ in line 1 and since $A(v_0)$ is assigned ∞ initially, we cannot have $A(v_0) = \perp$ upon when finishing the while-loop. Thus, in line 33 of Algorithm 2 we have $A(v_0) = F_{min}(v_0)$ by Theorem 7. Consequently, Algorithm 2 returns the minimum post fixed-point assignment of v_0, $F_{min}(v_0)$. □

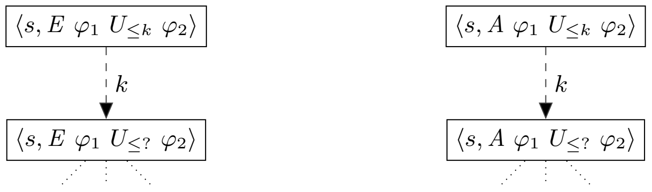
6 Model Checking with Symbolic Dependency Graphs

In this section we present an encoding of a WKS and a WCTL formula as a symbolic dependency graph. In turn this is used for checking whether or not the WKS satisfies the WCTL formula, by computation of the minimum post fixed-point assignment.

Definition 7 (Formula Satisfiability Encoding). *Given WKS $\mathcal{K}$, state s and a WCTL formula φ, we can encode the model checking problem as a symbolic dependency graph. Every configuration in the graph consists of a state and a formula. The dependency graph is expanded from the initial configuration $\langle s, \varphi \rangle$ using the rules illustrated in Figures 3a, 3b, 4a, 4b, 7, 8, 12a, 12b, 13 and 14.*

Theorem 8 (Encoding Correctness). *Let $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ be a WKS, $s \in S$ a state, φ a WCTL formula and G be a symbolic dependency graph expanded from $\langle s, \varphi \rangle$ following Definition 7. We have that $s \models \varphi$ if and only if $F_{min}(\langle s, \varphi \rangle) = 0$.*

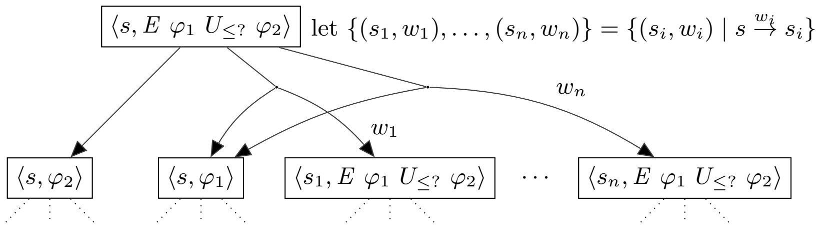
Proof. The proof is similar to that of Theorem 3.



$\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$
k
$\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$
$\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$
k
$\langle s, A\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$

(a) The existential until case. (b) The universal until case.

Fig. 12: Expansion of until expression configurations with specific cover-condition k.



$\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$
let $\{(s_1, w_1), \ldots, (s_n, w_n)\} = \{(s_i, w_i) \mid s \xrightarrow{w_i} s_i\}$
w_n
w_1
$\langle s, \varphi_2 \rangle$
$\langle s, \varphi_1 \rangle$
$\langle s_1, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$
$\cdots$
$\langle s_n, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$

Fig. 13: Successors of a existential until configuration.

Example 6 Figure 15 illustrates the symbolic dependency graph encoding of $E\ a\ U_{\leq 1000}\ b$ for the configuration s in the WKS in Figure 9, Example 3. The dependency graph encoding of the same query is shown in Figure 10.

It is clear that the symbolic dependency graph encoding is much smaller than the dependency graph encoding in Example 3 unfolds the bound 1000 one at a time. The minimum post fixed-point assignment of this symbolic

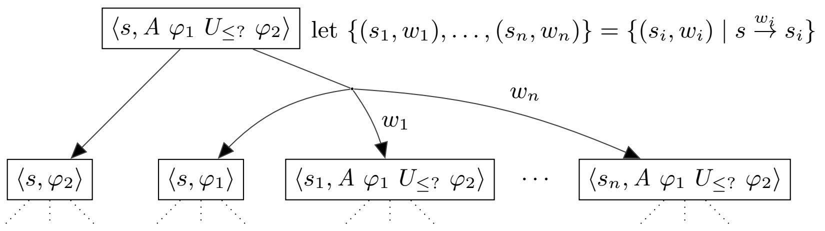

$\langle s, A\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$
let $\{(s_1,w_1),\dots,(s_n,w_n)\} = \{(s_i,w_i) \mid s \xrightarrow{w_i} s_i\}$
w_n
w_1
$\langle s, \varphi_2 \rangle$
$\langle s, \varphi_1 \rangle$
$\langle s_1, A\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$
$\cdots$
$\langle s_n, A\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$

Fig. 14: Successors of a universal until configuration. Notice that the set of successors is never empty as we are only considering non-blocking WKSs.

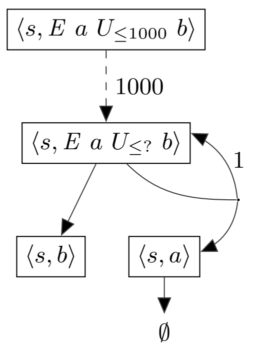

$\langle s, E\ a\ U_{\leq 1000}\ b \rangle$
1000
$\langle s, E\ a\ U_{\leq ?}\ b \rangle$
1
$\langle s, b \rangle$
$\langle s, a \rangle$
$\emptyset$

Fig. 15: Symbolic dependency graph encoding of the formula $s \models E\ a\ U_{\leq 1000}\ b$ for the WKS in Figure 9.

dependency graph is reached with a single iteration of the functor (Equation 5). From this we can conclude that $s \not\models E\ a\ U_{\leq 1000}\ b$.

Theorem 9 (Encoding Size). *Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, a state $s \in S$ and a WCTL formula φ, the size of the symbolic dependency graph encoding of $s \models \varphi$ from Definition 7 is $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi|)$.*

Proof. If we recall that all configurations in the encoding consists of a state and formula, it easy to see that the expansions gives rise to at most $O(|\mathcal{K}| \cdot |\varphi|)$ configurations. This follows from the fact that the only new formulas introduced in the expansions are existential and universal 'until' formulas with the bound '?'. However, this makes the number of formulas bounded by $2 \cdot |\varphi|$, thus, the number of configurations is bounded by $|\mathcal{K}| \cdot 2 \cdot |\varphi|$, which is $O(|\mathcal{K}| \cdot |\varphi|)$.

This leaves us to consider the number of hyper-edges leaving a configuration $\langle s, \varphi \rangle$ in the SDG, and we have that this is bounded by the number of outgoing edges from s. However, we can have multiple weighted edges between two states, so for a configuration the number of hyper-edges is bounded by $| \rightarrow |$. Therefore, we conclude that the size of the SDG is $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi|)$. $\square$

Theorem 10 (WCTL Satisfiability is in $\mathbb{P}$). *Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, a state $s \in S$ and a WCTL formula φ, $s \models \varphi$ can be decided in $O(|\mathcal{K}|^5 \cdot |\varphi|^5 \cdot |{\rightarrow}|^5)$ time.*

Proof. To decide $s \models \varphi$, we encode the problem using Definition 7. From Theorem 9, the size of the resulting symbolic dependency graph is $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi|)$. By Theorem 5, the minimum post fixed-point assignment of $\langle s, \varphi \rangle$ can be computed in $O(|V|^3 \cdot |C| + |V|^4 \cdot |H|)$ time. Thus, as $|V|$, $|C|$ and $|H|$ are $O(|\mathcal{K}| \cdot |{\rightarrow}| \cdot |\varphi|)$, we have decided $s \models \varphi$ in $O(|\mathcal{K}|^5 \cdot |\varphi|^5 \cdot |{\rightarrow}|^5)$ time. Correctness of this approach follows from Theorem 8. $\square$

Notice that the complexity bounds in Theorem 9 and 10 are by no means tight. Nevertheless, these results are sufficient for what we wanted to show and avoid unnecessary technicalities. After all the main result is that WCTL formulae satisfiability can be decided in polynomial time. As we shall see in the following, Algorithm 2 may exhibit exponential running time, still, we strongly believe that with further improvements to this algorithm, it is possible to provide a better complexity bound than in Theorem 5.

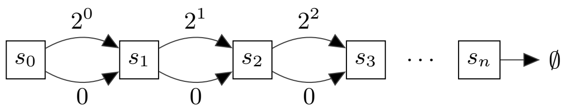

$$\begin{tikzcd} \boxed{s_0} \arrow[r, bend left, "2^0"] \arrow[r, bend right, "0"'] & \boxed{s_1} \arrow[r, bend left, "2^1"] \arrow[r, bend right, "0"'] & \boxed{s_2} \arrow[r, bend left, "2^2"] \arrow[r, bend right, "0"'] & \boxed{s_3} & \cdots & \boxed{s_n} \arrow[r] & \emptyset \end{tikzcd}$$

Fig. 16: A symbolic dependency graph that can take exponential running time with Algorithm 2, if edges are chosen according to the strategy outlined in Equation 14.

Theorem 11 (Lower Bound Complexity of Local Algorithm). *The running time of Algorithm 2 can be exponential in the size of the symbolic dependency graph.*

Proof. Consider the execution of Algorithm 2 on a symbolic dependency graph with $n+1$ configurations on the form shown in Figure 16, with the initial configuration s_0. We pick edges from W in line 5 of Algorithm 2 using the following strategy.

$$Pick(W) = \begin{cases} (s_i, \{(2^i, s_{i+1})\}) & \text{if } \exists (s_i, \{(2^i, s_{i+1})\}) \in W \\ \displaystyle\operatorname*{arg\,min}_{(s_i, \{(0, s_{i+1})\}) \in W} i & \text{otherwise} \end{cases} \tag{14}$$

Under these conditions the initial assignment ∞ of $A(s_0)$ is followed by $2^{n+1}-1$, which is then followed by $2^{n+1}-2, 2^{n+1}-3, \ldots$, and so on, counting all the way down to 0. Obviously, counting this takes $O(2^{n+1})$, while the size of the symbolic dependency graph is only polynomial in n. Thus, Algorithm 2 can take exponential time. $\square$

While Theorem 11 states that the local algorithm for minimum post fixed-point computation on symbolic dependency graphs can take exponential time, this is a degenerate case. In practice these cases are rare and if edges are picked from the queue W in a first-in first-out order, or first-in last-out order, we do not believe that the algorithm exhibits exponential running time.

7 Experimental Results

In this section we present, WKTool, a proof of concept implementation of the algorithms and encodings presented in this paper. We then evaluate the execution time of the different algorithms, concluding that the local algorithm on a symbolic dependency graph encoding of the model checking problem is the most efficient.

WKTool takes a WKS, a state s and a WCTL formula φ as input to evaluate whether $s \models \varphi$. This can be done with four strategies, first the model checking problem is translated to a dependency graph or a symbolic dependency graph, using Definition 5 and Definition 7, respectively. Then the minimum post fixed-point assignment of the (symbolic) dependency graph is computed, either by repeated application of the functor or using a local algorithm for the configuration $\langle s, \varphi \rangle$.

The local algorithm for dependency graphs is Algorithm 1 and Algorithm 2 for symbolic dependency graphs. In the implementation of these algorithms the translation to dependency graphs or symbolic dependency graphs is executed on-the-fly during exploration. In this paper we have used $succ(v)$ in both Algorithm 1 and Algorithm 2 to denote the edges leaving configuration v. In the implementation *succ* is implemented to create the edges and configurations the first time they are requested. In addition, references to them are kept, so they can be reused for later requests.

The implementation of repeated functor application for dependency graphs and symbolic dependency graph in WKTool is relatively straightforward. Though, the implementation does have some trivial optimizations, such as skipping configurations that are already satisfied. The implementation also updates the same assignment it is writing to, since maintaining two and swapping them at the end of each functor application would be more complicated. This just to say, that although the implementation is fairly simple, it is not unreasonbly naive.

WKTool has a web-based front-end, is written in CoffeeScript and available at `http://jonasfj.github.com/WKTool/`. The tool is entirely browser-based, only using a small Java applet-based nanotimer for precise benchmarking. WKTool also has a command-line client running on Node.js (≥ 0.8), which was used to conduct the experiments presented in this section.

Experiments were conducted on a laptop (Intel Core i7) running Ubuntu. Benchmarks were run on 1000 randomly generated WKS models, where $s_1 \models E\ a\ U_{\leq 6}\ b$ and $s_1 \not\models A\ a\ U_{\leq 10}\ b$ holds. The number of states of each model was between 40 and 80 and all states were labeled with the atomic proposition a and had a 10% chance of also being labeled with the atomic proposition b. Every state had 1 to 5 outgoing transitions with weight between 0 and 8. Execution times were measured in real-time using a high resolution nanosecond timer. The initial setup and time needed to parse models and formulae is not included in the measurements.

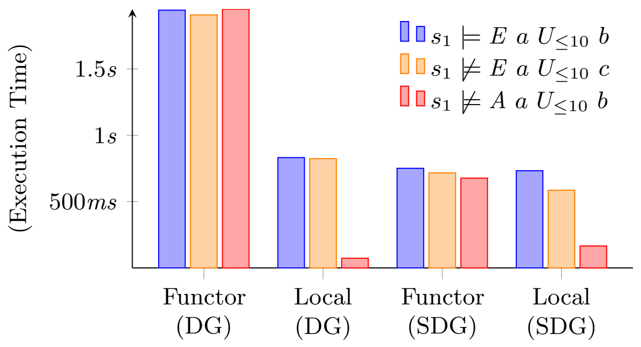


Fig. 17: Comparison of execution times for functor and local algorithm, using dependency graph (DG) and symbolic dependency graph (SDG) encodings.

In Figure 17 we see the execution time needed to conclude $s_1 \models E\ a\ U_{\leq 10}\ b$, $s_1 \not\models E\ a\ U_{\leq 10}\ c$ and $s_1 \not\models A\ a\ U_{\leq 10}\ b$ on the 1000 randomly generated WKS models. From this figure we see that the local algorithms are better than repeated functor application, especially, when checking $A\ a\ U_{\leq 10}\ b$. If we recall the expansion for configurations on the form $\langle s, A\ a\ U_{\leq k}\ b \rangle$, we notice that it only have has two hyper-edges, and if $s \not\models b$ it requires all of its successor configurations in the other hyper-edge to hold. The local algorithms can exploit this to finish early, if it can conclude that the property does not hold in just one of these successors.

From Figure 17 we are also hinted that this symbolic dependency graph encoding and accompanying algorithms are faster than the dependency graph encoding. This is not surprising, as Theorem 4 concludes that the size of the dependency graph encoding is pseudo-polynomial in the size of the input. Next we shall see that, if we increase the bound k in the WCTL formula $E\ a\ U_{\leq k}\ b$, then the execution time of the dependency graph-based grows exponentially.

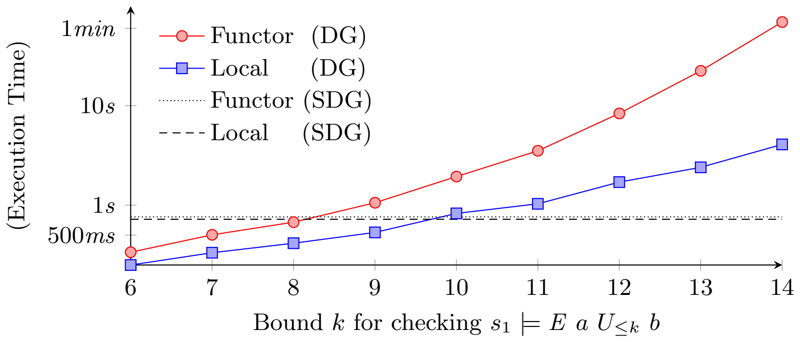


Fig. 18: A semi-log plot of the execution time as a function of the bound given in an existential "until" property. The y-axis is logarithmic in the exeuction time, while the x-axis is linear in the bound k.

In Figure 18 we see how the execution time evolves as the bound k grows in the WCTL formula $E\ a\ U_{\leq k}\ b$. Keeping in mind that the y-axis of the plot scales logarithmicly, it is easy to see that the execution time is exponential in the bound k for algorithms running on the dependency graph encoding. Meanwhile, the execution time of the strategies using the symbolic dependency graph encoding are virtually unaffected of the bound.

The reader may notice that the dependency graph encoding is faster for small bounds. This is the case because the dependency graph encoding does not expand further when the bound becomes zero. In contrast, the symbolic dependency graph encoding continues exploring states, even though they are not reachable within the bound. This suggests that further optimization of the symbolic dependency graph encoding and accompanying algorithms, might be interesting.

The experiments presented in this section are by no means exhaustive, yet we believe that they indicate that local fixed-point algorithms and symbolic encoding for WCTL model checking are worthwhile. Nevertheless, it is still important to investigate the performance of these algorithms on more complex models and properties.

8 Conclusion

We have formalized weighted Kripke structures, weighted computation tree logic (WCTL) and described a semantics for this logic. Then we demonstrated how to solve the model checking problem of WCTL formulae w.r.t. weighted Kripke structures using Liu and Smolkas local algorithm for dependency graphs. We argued that this approach is pseudo-polynomial in the upper bounds of WCTL formulae and improve on this complexity by extending dependency graphs in the form of symbolic dependency graphs. A local algorithm for model checking WCTL formulae on weighted Kripke structures using symbolic dependency graphs is then presented. We proved its correctness and showed that the size of the symbolic encoding is polynomial in the input and that WCTL formulae can be satisfied in polynomial time. Experimental results are encouraging, as they indicate that the symbolic algorithm yields substantial improvements over the dependency graph-based approach in relation to WCTL model checking. Lastly, we summarize the complexity results of this paper in Table 2

	Encoding size	Satisfiability of formulae
Dependency Graphs	$O(\lvert\mathcal{K}\rvert \cdot \lvert\rightarrow\rvert \cdot \lvert\varphi\rvert \cdot \mathcal{B}(\varphi))$	$O(\lvert\mathcal{K}\rvert \cdot \lvert\rightarrow\rvert \cdot \lvert\varphi\rvert \cdot \mathcal{B}(\varphi))$
Symbolic Dependency Graphs	$O(\lvert\mathcal{K}\rvert \cdot \lvert\rightarrow\rvert \cdot \lvert\varphi\rvert)$	polynomial

Table 2: A summary of the complexity results in this paper, where $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ is the input model, φ is the WCTL formula and $\mathcal{B}(\varphi)$ is the upper bound on the k parameter of a formula in φ.

8.1 Future Work

In the following we propose possible directions for future work. First of all, it is of interest to define a process algebra like CCS for weighted Kripke structures as it would allow one to express weighted systems running in parallel. Hence, it would make modeling less cumbersome and provide more interesting systems to verify. A weighted CCS could simply be an extension of the usual CCS with weights on the action-prefixes of processes, e.g. if an action a of some process P should carry weight 5, we could write it as $a[5].P$.

It is worth investigating if it is possible to adapt our framework to support alternating fixed points, because it could allow one to model check a more expressive weighted logic. Alternating fixed points could allow us to describe liveness and fairness properties. Liu, Ramakrishnan and Smolka demonstrate how a local algorithm for partitioned dependency graphs enables the evaluation of alternating fixed points in [6]. A starting point could be to examine this framework and determine if it can be put to practical use for our logic.

The logic described in this paper only supports upper bounds on the weights of formulae. It makes sense to extend this logic to also support lower bounds, because it enables us to pose other kinds of queries. There are two obvious ways to extend the logic. The first way is to augment the syntax of the "until" and "next" operators with the usual comparison operators $\bowtie = \{<, \leq, =, \geq, >\}$. The second way is to replace the bounds on formulae with an interval $[k_1, k_2]$. What remains to be answered is whether the latter approach is strictly more expressive than the former and perhaps more importantly how to adapt the algorithm for the extended logic.

Finally, we could take the extended logic a step further and introduce parameters to the formulae and perhaps also the model. The algorithm could then be altered to also synthesize the parameters, if any, such that a given formula is satisfied.

References

1. Henrik Reif Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3 – 30, 1994.
2. Peter Buchholz and Peter Kemper. Model checking for a class of weighted automata. *Discrete Event Dynamic Systems*, 20:103–137, 2010.
3. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 1020 states and beyond. *Information and Computation*, 98(2):142 – 170, 1992.
4. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *IN CONCUR 05, LNCS 3653*, pages 66–80. Springer, 2005.
5. Kim G. Larsen, Uli Fahrenberg, and Claus R. Thrane. A quantitative characterization of weighted kripke structures in temporal logic. In *MEMICS*, 2009.
6. Xinxin Liu, C.R. Ramakrishnan, and ScottA. Smolka. Fully local and efficient evaluation of alternating fixed points. In Bernhard Steffen, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *Lecture Notes in Computer Science*, pages 5–19. Springer Berlin Heidelberg, 1998.
7. Xinxin Liu and Scott A. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In *ICALP*, pages 53–66, 1998.