On-The-Fly Model Checking of Weighted Computation Tree Logic

Jonas Finnemann Jensen and Lars Kærlund Østergaard

Department of Computer Science, Aalborg University
Selma Lagerlöfs Vej 300, 9220 Aalborg, Denmark
{jopsen, larsko}@gmail.com

Abstract. We present min-max graphs; a technique to encode the model checking problem of weighted Computation Tree Logic (CTL), with non-negative constraints on the modalities, against weighted Kripke structures. We outline previous efforts to encode this problem using dependency graphs and their symbolic extension. We demonstrate how to model check weighted CTL through fixed-point computation on a graph that encodes the problem, using global and local/on-the-fly fixed-point algorithms. Moreover, we have implemented the algorithms and evaluated their performance through experiments. In turn, we conclude that the local approach is often very advantageous.

1 Introduction

Model checking is a technique for automatically determining whether a model of a system adheres to a specification. For instance, there are often certain safety, correctness and performance requirements associated with the development of complex hardware and software systems. To this extent model checking offers a means to verify whether the model of some system satisfies the constraints of a specification. Such specifications may be defined formally as propositions in temporal logic and validated against the model through a model checker. If there is a violation of the specification, a counter-example is provided and the model can then be redesigned. Hence, certain functional properties of the system may be verified in a precise, unambiguous and rigorous way.

Although it is possible to express functional properties of systems using temporal logic, other aspects such as timing and resource constraints (e.g. bandwith, memory, power usage, etc) should also be captured. In particular, for embedded systems, these constraints must necessarily be adressed. A number of modeling formalisms have been proposed in an effort to accomodate the need for expressing such quantitative properties. Notably, timed automata (TA) [1] and variants of weighted timed automata (WTA), with cost information both on locations and edges [6,2], have received considerable interest, including tool support [20,25,12].

Well-established temporal logics such as branching time (CTL) and linear time (LTL) temporal logic permit natural weighted extensions, making it is possible to describe quantitative behavorial properties such as cost constraints on

modalities. Moreover, CTL and LTL have been extended with time-constrained modalities in the form of TCTL and MTL, respectively. WCTL and WMTL are similar extensions weighted, yet interpreted over WTAs. Unfortunately, the addition of weight to TAs has a severe limitation. While the model checking TAs using TCTL and MTL is decidable, it has been proved to be undecidable for WTAs (with at least three clocks) with respect to WCTL [8].

We direct our attention to model checking untimed models with the addition of weights. Specifically, models are described using weighted Kripke structures and behavioural properties are expressed in weighted CTL with bounded constraints on the weights. Our main contributions are an efficient symbolic encoding of the WCTL model checking problem, and a local/on-the-fly model checking algorithm. In addition, we show that our approach is in PTIME.

The results of this paper are based off ideas from Liu and Smolka's dependecy graphs (DG) [22]. They essentially encode boolean equation systems using DGs and describe local and global algorithms for computing alternation-free fixed-points on DGs in linear time. However, DGs lead to a pseduo-polynomial encoding with the addition of weights to the formulas. To remedy this, DGs are lifted to the weighted setting with a symbolic extension, called symbolic dependency graphs (SDG), to encode the WCTL (negation-free and with upper-bound constraints) model checking problem more efficiently in [13]. Still, this approach is limited to an alternation- and negation-free WCTL sub-logic.

In this paper we take the idea of a symbolic encoding a step further in the shape of min-max graphs (MMGs). The advantage of this technique is that we can support a more expressive WCTL logic, a polynomial encoding, and still solve the model checking problem using on-the-fly fixed-point evaluation. We provide experimental results that compare MMGs to the previously attempted techniques and evaluate their performance on a large set of models. These results demonstrate that the local MMG algorithm is generally just as efficient as the symbolic local algorithm. Furthermore, the algorithm preserves the desirable property of the symbolic local one in that is often an order of magnitude faster for positive outcomes.

Related Work

Laroussinie, Markey and Oreiby [17] consider the problem of model checking durational concurrent game structures with respect to timed ATL properties. They describe a PTIME procedure for verifying formulae with non-punctual constraints, i.e. without equality. By limiting the game structure to a single player we are essentially left with WCTL. Nevertheless, the approach in [17] is limited to non-zero transition weights, has no weak until modality and the algorithm is global, requiring a full state-space exploration. We do, however, allow zero-weights in the model, resulting in the need for fixed-point evaluation. Hence, our approach subsumes regular unweighted CTL model checking, in constrast to [17]. Moreover, [17] provide no implementation nor any local algorithm, which our experiments show is often superior.

Lauroussinie, Schnoebelen and Turuani [16] study the expressive power of CTL extended with discrete time. The elapse of time is encoded with a proposition *tick*. Extending this technique for the general weighted setting results in a pseudo-polynomial algorithm.

Buchholz and Kemper [9] propose a valued computation tree logic (CTL$) that is interpreted over a general set of weighted automata. Regular CTL is included as a special case over the boolean semiring. Model checking CTL$ is then carried out through a matrix-based approach. Their algorithm supports a fully nested logic, including negation and all of the comparison operators with respect to the bounds in the formula. However, no local algorithm is provided.

Laroussinie, Meyer and Petonnet [18] demonstrate P-completeness of TCTL model checking against durational Kripke structures with additive transition weights belonging to $\{-1, 0, 1\}$. The until modality with upper- and lower-bound constraints is supported by checking the unconstrained CTL formula and running the Floyd-Warshall algorithm to compute the all-pairs shortest paths over those states that satisfy the formula. This approach can be extended to handle weights in $\mathbb{Z}$, but has cubic complexity and is inherently global, requiring a complete state-space exploration.

A number of techniques for local model checking of the modal mu-calculus have been proposed. A local procedure running in $O(n \cdot log(n))$ for model checking the modal mu-calculus with alternation depth one is described in Andersen [3]. Liu and Smolka [22] improve with a linear-time local algorithm (in the size of the input graph) for evaluating alternation-free fixed points on dependency graphs. We adopt the ideas of [22] and tailor them for WCTL model checking.

Liu, Ramakrishnan and Smolka introduce a local and efficient algorithm for the evaluation of alternating fixed-points in [21]. As a result, this can be applied to model checking of the full modal mu-calculus. The central idea is partition the vertices of a dependency graph into blocks, labelled with the fixed-point operator μ or ν, where the ith block represents the ith-most nested fixed-point. The fixed-point is then computed iteratively, avoiding a priori construction of the state space. Nevertheless, this framework is subject to a exponentially large graphs with the addition of weights (as for dependency graphs).

Consequently, we employ a different strategy to handle both the weights and the nesting of fixed points. Rather, weights are handled with a symbolic encoding and alternation is achieved with special "cover"-edges. The nesting is handled by ensuring that the bottom-most fixed-points have been established before the values are propagated upwards. Despite this restriction we are still able to offer an efficient local algorithm.

Outline. Weighted Kripke structures and Weighted CTL (WCTL) are presented in Section 2. In Section 3 dependency graphs are introduced and a reduction from a negation-free subset of the WCTL model checking problem is covered. We expand upon dependency graphs in the form of symbolic dependency graphs, present a local algorithm and discuss this approach in terms of negation-free WCTL model checking in Section 4. A more general technique in the form of min-max graphs is proposed in Section 5. Section 6 introduces a global algorithm for fixed-point computation on min-max graphs, and Section 7 provides details for a local algorithm. A reduction from the full WCTL model checking problem is then described in Section 8. Section 9 presents a weighted variant of CCS. Experimental results are presented in Section 10 and Section 11 concludes the paper. Section 12 includes bibliographical remarks.

2 Weighted Kripke Structure

We use Kripke structures, well-known model for temporal logic, to derive a model for the weighted setting. *Weighted* Kripke structures extends this model with a weighted transition relation. A natural interpretation is to consider the weights as the cost associated with taking transitions. This extension is similar to that in [19], yet the weights in our version are nonnegative, rather than real-valued. In addition, we provide a definition of weighted computation tree logic and its semantics. Let $\mathbb{N}_0$ denote the set of natural numbers including zero, a weighted Kripke structure is defined as follows.

Definition 1 (Weighted Kripke Structure). *A* Weighted Kripke Structure *(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,*
- *$L : S \rightarrow \mathcal{P}(\mathcal{AP})$ is a mapping from states to sets of atomic propositions, and*
- *$\rightarrow \subseteq S \times \mathbb{N}_0 \times S$ is a transition relation.*

Whenever $(s, w, s') \in \rightarrow$ for some WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, we say that there is a transition from s to s' with the weight w, denoted $s \xrightarrow{w} s'$. A WKS is said to be *non-blocking* if for every $s \in S$ there exists $s' \in S$ such that $s \xrightarrow{w} s'$ for some weight $w \in \mathbb{N}_0$. From now on we shall only consider non-blocking structures.

Remark 1. A blocking WKS can be transformed to a non-blocking WKS by introducing an auxiliary state $s_\perp$, such that $s_\perp$ has no atomic propositions, a zero-weight self-loop and an ingoing zero-weight transition from every blocking state.

A *run* in a 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 is then

$$W_\sigma(p) = \sum_{i=0}^{p-1} w_i$$

We now define Weighted Computation Tree Logic (WCTL) with weight upper- and lower-bounds. The set of WCTL formulas over the set of atomic propositions $\mathcal{AP}$ is given by the abstract syntax

$$\varphi ::= \mathbf{true} \mid \mathbf{false} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid QX_{\bowtie k}\ \varphi \mid Q\ \varphi_1\ U_{\leq k}\ \varphi_2 \mid Q\ \varphi_1\ W_{\geq k}\ \varphi_2 \mid \neg\varphi$$

where $Q \in \{E, A\}$, $k \in \mathbb{N}_0 \cup \{\infty\}$, $a \in \mathcal{AP}$, and $\bowtie \in \{\leq, \geq\}$.

Note that we omit the "=" operator, as the model checking problem for temporal formulas with "=" is EXPTIME-complete [17]. We denote the negation-free WCTL fragment, without the weak until modality, and which only permits upper-bounds as $\text{WCTL}_{\leq}$.

The standard abbreviated CTL operators are derived: $EF_{\leq k}\ \varphi \equiv E\ \mathbf{true}\ U_{\leq k}\ \varphi$, $AF_{\leq k}\ \varphi \equiv A\ \mathbf{true}\ U_{\leq k}\ \varphi$, $EG_{\leq k}\ \varphi \equiv \neg AF_{\leq k}\ \neg\varphi$ and $AG_{\leq k}\ \varphi \equiv \neg EF_{\leq k}\ \neg\varphi$, as shown for unweighted CTL in [11]. Note that the usual CTL operators are included, since U is equivalent to $U_{\leq\infty}$.

Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, a state $s \in S$ and a WCTL formula φ over $\mathcal{AP}$, we write $s \models \varphi$ if the state s satisfies φ. The semantics of WCTL is given inductively as follows.

$s \models \mathbf{true}$	
$s \models a$	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 s.t. $\sigma \models \varphi_1\ U_{\leq k}\ \varphi_2$
$s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$	if for any run σ starting from s we have $\sigma \models \varphi_1\ U_{\leq k}\ \varphi_2$
$s \models E\ \varphi_1\ W_{\geq k}\ \varphi_2$	if there exists a run σ starting from s s.t. $\sigma \models \varphi_1\ W_{\geq k}\ \varphi_2$
$s \models A\ \varphi_1\ W_{\geq k}\ \varphi_2$	if for any run σ starting from s we have $\sigma \models \varphi_1\ W_{\geq k}\ \varphi_2$
$s \models EX_{\bowtie k}\ \varphi$	if there exists s' s.t. $s \xrightarrow{w} s' and s' \models \varphi$ with $w \bowtie k$
$s \models AX_{\bowtie k}\ \varphi$	if for all s' s.t. $s \xrightarrow{w} s'$ with $w \bowtie k$ it holds that $s' \models \varphi$
$s \models \neg\varphi$	if $s \not\models \varphi$
$\sigma \models \varphi_1\ U_{\leq k}\ \varphi_2$	if there is 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 $p' \leq p$
$\sigma \models \varphi_1\ W_{\geq k}\ \varphi_2$	if $\sigma(p) \models \varphi_1$ for all p, or if there is a position $p \geq 0$ s.t. $\sigma(p) \models \varphi_2, W_\sigma(p) \geq k$ and $\sigma(p') \models \varphi_1$ for all $p' \leq p$

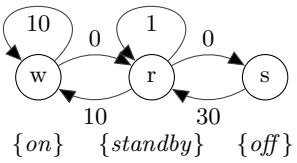


Fig. 1. A simple WKS model of a three-state controller

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

$$
\begin{aligned}
S &= \{s, t, r\} \\
\mathcal{AP} &= \{\mathit{off}, \mathit{on}, \mathit{standby}\} \\
L &= \{w \mapsto \{\mathit{on}\}, r \mapsto \{\mathit{standby}\}, s \mapsto \{\mathit{off}\}\} \\
\rightarrow &= \{(w, 10, w), (w, 0, r), (r, 1, r), (r, 10, w), (r, 0, s), (s, 30, r)\}
\end{aligned}
$$

The WKS models a simple three-state controller. It may alternate between three states. Intuitively, it may either be performing work in state w, idle in r or powered down in s.

There is a cost in resources associated with doing work in state w. Also, there is a relatively high cost of 30 resource units incurred when transitioning from the *off*-state s to the *standby*-state r, because the controller is entirely powered down in s.

Finally, there is a small cost associated with polling for tasks in state r. Hence, there is a trade-off between having the controller in the idle state or turning it off, depending on the amount of work that arrives.

For the WKS in Figure 1 we can express the property that we can transition from *off* to *on* using less than 50 resource units as $s \models \mathit{EF}_{<50}\ \mathit{on}$. While WCTL as introduced above does not allow for strict comparison operators, this has no practical limitations, because weights are discrete, hence, the statement $s \models \mathit{EF}_{<50}\ \mathit{on}$ is equivalent to $s \models \mathit{EF}_{\leq 49}\ \mathit{on}$.

3 Dependency Graph

In this section we shall introduce Liu and Smolka's dependency graph framework for alternation-free fixed-point computation, as presented in [22]. An application of dependency graphs is model checking of the alternation-free modal mu-calculus. We demonstrate that the framework can be straightforwardly adapted to CTL model checking with the addition of weights.

Definition 2 (Dependency Graph). *A* dependency graph *(DG) 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. A hyper-edge $e = (v, T)$ consists of a source configuration v and target set T. The set of successors of a configuration is $succ(v) = \{(v, T) \in E\}$. An assignment $A : V \to \{0, 1\}$ is a mapping from configurations in G to boolean values. A pre fixed-point assignment of G is an assignment A, such that for every $v \in V$, if $(v, T) \in E$ and $A(u) = 1$ for every $u \in T$, then also $A(v) = 1$.

By the standard component-wise ordering $\sqsubseteq$ (with $0 < 1$) on assignments, where $A \sqsubseteq A'$ if and only if $A(v) \leq A'(v)$ for all $v \in V$, we have by the Knaster-Tarski fixed-point theorem that there exists a unique minimum pre fixed-point assignment, denoted A^{min}. We denote the set of all assignments by *Assign*. It can be computed by repeated application of the monotonic function $F : Assign \to Assign$, listed below. The function is evaluated with the initial assignment $A_0(v) = 0$ for all $v \in V$.

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

We write $F^i(A)$ to denote the ith application of F on A, that is $F^i(A) = F(F^{i-1}(\ldots F^1(A)))$. Because F is evaluated on a finite complete lattice, we always to reach a fixed-point after a finite number of applications. Thus, we have that $F^m(A_0) = A^{min}$ for some $m \in \mathbb{N}_0$, such that $F^m(A_0) = F^{m+1}(A_0)$. We shall refer to repeated application of the functor F as the *global* algorithm.

3.1 Local Minimum Fixed-Point Algorithm

Often we are only interested in the minimum fixed-point for a specific configuration v_0, e.g. in many model checking questions. For this reason Liu and Smolka [22] propose a *local* algorithm to compute the value of $A^{min}(v_0)$ on-the-fly. Algorithm 1 lists the slightly modified[1] pseudo code of their algorithm.

[1] At line 1 we added the current hyper-edge e to the dependency set $D(u)$ of the successor configuration u, i.e. $D(u) = \{e\}$. The original algorithm sets the dependency set to be empty here, leading to incorrect propagation.

Algorithm 1: Liu-Smolka Local Algorithm

Input: Dependency graph $G = (V, E)$ and initial configuration $v_0 \in V$
Output: Minimum fixed-point assignment of v_0, $A^{min}(v_0)$
let $A(v) := \bot$ for all $v \in V$
$A(v_0) := 0$
$D(v_0) := \emptyset$
$W := succ(v_0)$
while $W \neq \emptyset$ **do**
  let $e := (v, T) \in W$
  $W := W \setminus \{e\}$
  if $\forall u \in T$ *it holds that* $A(u) = 1$ **then**
    $A(v) := 1$
    $W := W \cup D(v)$
  else if $\exists u \in T$ *where* $A(u) = 0$ **then**
    $D(u) := D(u) \cup \{e\}$
  else if $\exists u \in T$ *where* $A(u) = \bot$ **then**
    $A(u) := 0$
    $D(u) := \{e\}$
    $W := W \cup succ(u)$
return $A(v_0)$

The basic idea is to compute the minimum fixed-point assignment of the dependency graph for a specific configuration v_0, by only exploring those configurations actually needed in order to establish the value of $A^{min}(v_0)$. Three data-structures are maintained in the algorithm, the assignment A, the dependency set $D : V \to \mathcal{P}(V)$ and the waiting list of hyper-edges W.

To begin with the assignment of the inital configuration v_0 is 0, as it is assumed to be *false*. The outgoing hyper-edges of v_0 are then added to the waiting list W. Every configuration except v_0 has the initial assignment $\bot$ to indicate that the value is unknown, but assumed to be 0. Once a hyper-edge $e = (v, T)$ is removed, T is examined to see if $A(v)$ can be forced to become 1 for the source configuration v. This is the case when $A(u) = 1$ for all $u \in T$, as a hyper-edge intuitively represents a disjunction of conjunctions.

If a configuration u with the assignment $\bot$ is encountered, then its assignment is updated to 0 and its outgoing hyper-edges (u, T) are added to W, moreover the hyper-edge $e = (v, T)$ is added to $D(u)$. Hence, if $A(u)$ ever becomes 1, the dependencies (hyper-edges) are re-evaluated to check if it is possible to force the respective source configurations to be assigned 1. Moreover, if there is some $u \in T$ such that $A(u) = 0$ is encountered, the hyper-edge is also added to its dependency set. If $A(v_0) = 1$ at some point during execution, then it is possible to terminate early, because the initial configuration is satisfied and the assignment cannot become 0 again.

Given an input dependency graph $G = (V, E)$ and initial configuration $v_0 \in V$, the algorithm runs in $O(|G|)$ time, where $|G| = |V| + \sum_{(v,T)\in E}(|T| + 1)$. Correctness and complexity of Algorithm 1 is proved in [22].

3.2 Model Checking with Dependency Graphs

Here we propose a reduction from the model checking problem of $\text{WCTL}_{\leq}$ on WKS to minimum fixed-point computation of assignments on dependency graphs. Let $\mathcal{K}$ be a WKS, a state s in $\mathcal{K}$ and φ be a $\text{WCTL}_{\leq}$ formula. A dependency graph G is constructed, such that every configuration represents a state-formula pair, denoted $\langle s, \varphi \rangle$. With the initial pair $\langle s, \varphi \rangle$, the dependency graph constructed using the rules illustrated in Figure 2.

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

Proof. By structural induction on the formula φ. See Appendix A.1 for details. □

Note that the model checking reduction to dependency graphs is performed on-the-fly in a need-driven fashion as the local algorithm requests successor states (lines 1 and 1 of Algorithm 1), during fixed-point computation. This is an important point, as in order to benefit from local exploration, we want to avoid creating the entire dependency graph up-front. Section 10 presents experimental results demonstrating that the local algorithm is often more efficient than the global algorithm.

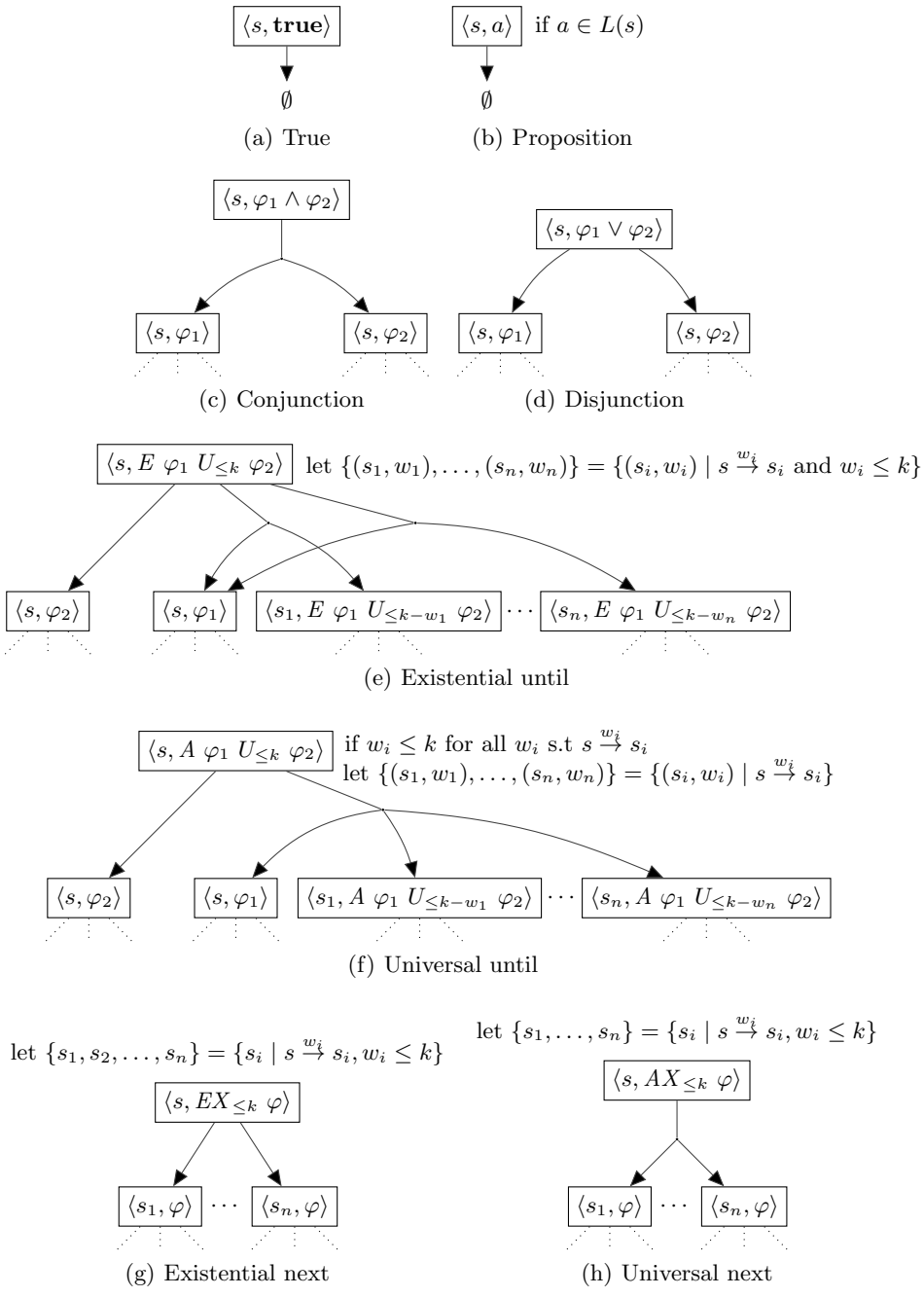


Fig. 2. Dependency graph encoding of state-formula pairs

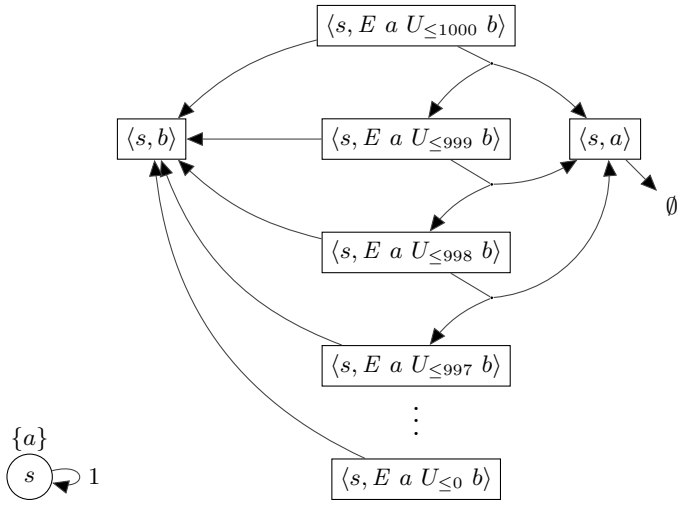


Fig. 3. A WKS and its dependency graph for the formula $E\ a\ U_{\leq 1000}\ b$

Still, the dependency graph technique has a downside. We can easily construct exponentially large dependency graphs, in terms of the weight bound in the formula, as shown in Figure 3. A single-state WKS with a self-loop is shown to the left and a large dependency graph to the right in the figure. Notice that the size of the dependency graph depends on the bound in the formula. Hence, we are left with a pseduo-polynomial algorithm for model checking $\text{WCTL}_{\leq}$. In the sequel we shall present a more efficient technique, allowing us to perform model checking in polynomial time.

4 Symbolic Dependency Graph

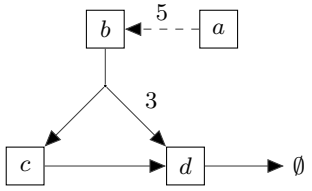
In the previous section we saw that the dependency graph method for WCTL$_\leq$ model checking has a drawback. Namely, it suffers from an exponential blow-up as the graph grows in proportion to the bounds in the provided formula. The unfolding of the until modality is the cause of this undesirable circumstance. However, we can make use of the basic observation that the validity of $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ also implies $s \models E\ \varphi_1\ U_{\leq k+1}\ \varphi_2$. In the following section we propose a *symbolic* extension of dependency graphs that capture this implication, which in turn reduces the size of the constructed graph. This extension, called symbolic dependency graphs, enables us to perform efficient (polynomial time) model checking of WCTL. Finally, this technique also lends itself well to on-the-fly model checking.

Definition 3 (Symbolic Dependency Graph). *A symbolic dependency graph (SDG) is a triple $G = (V, H, C)$, where V is a finite set of configurations, $H \subseteq \mathcal{P}(\mathbb{N}_0 \times V)$ is a finite set of hyper-edges, and $C \subseteq V \times \mathbb{N}_0 \times V$ is a finite set of cover-edges.*

SDGs differ from ordinary DGs in the way edges are defined. First of all every element in the target set of a hyper-edge is a pair composed of a weight and a configuration. Secondly, a new kind of edge is introduced which we refer to as a cover-edge. Finally, SDGs operate over the complete lattice $\mathbb{N}_0 \cup \{\infty\}$ in contrast to DGs, where the domain is boolean.

Let $G = (V, H, C)$ be an SDG. A hyper-edge $e = (u, T)$ consists of a source configuration u and target-set T. An element $(w, v) \in T$ is called a hyper-edge branch with weight w pointing to v. The set of successors of configuration u is $succ(u) = \{(u, T) \in H\} \cup \{(u, k, v) \in C\}$.

Figure 4(a) illustrates an example of an SDG. A hyper-edge is drawn using a solid line and every hyper-edge branch is annotated with its weight (which is omitted whenever the weight is zero). A cover-edge is depicted with a dashed line and a cover-condition.



(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) Fixed-point computation

Fig. 4. Computation of the minimum pre fixed-point assignment of an SDG

An assignment $A : V \to \mathbb{N}_0 \cup \{\infty\}$ of G is a function that maps configurations to values. The set of all assignments is denoted $Assign$. Let $F : Assign \to Assign$ be a function defined as follows.

$$F(A)(v) = \begin{cases} 0 & \text{if } \exists (v, k, v') \in C \text{ s.t. } A(v') \leq k < \infty, \text{ or } A(v') < k = \infty \\ \min\limits_{(v,T) \in H} \big(\max\{w + A(v') \mid (w, v') \in T\} \big) & \text{otherwise.} \end{cases} \tag{1}$$

An assignment $A \in Assign$ is a pre fixed-point assignment if $A = F(A)$. Let $\sqsubseteq$ be a partial order over assignments of G, such that $A \sqsubseteq A'$ if and only if $A(v) \geq A'(v)$ for all $v \in V$. Then F is clearly monotonic on the complete lattice $(Assign, \sqsubseteq)$. By the Knaster-Tarski fixed-point theorem, there is a unique minimum pre fixed-point assignment of G, denoted A^{min}.

Note that we write $A \sqsubseteq A'$ if for all $v \in V$, it holds that $A(v) \geq A'(v)$, in reversed order. Also, the smallest element in the lattice is $A_0(v) = \infty$ for all $v \in V$. The lattice is finite and there is no infinite increasing sequence of weights w.r.t. $\sqsubseteq$ (they are elements of $\mathbb{N}_0$). Thus, the minimum pre fixed-point assignment A^{min} can be obtained in a finite number of steps using F with the initial assignment A_0 as the bottom element. Consequently, we have that $F^m(A_0) = F^{m+1}(A_0)$ for some $m \in \mathbb{N}_0$, and hence $F^m(A_0) = A^{min}$ is the minimum pre fixed-point assignment of G.

Again, we shall refer to F as the *global* algorithm for computing fixed-points on SDGs. Figure 4 shows the sequence of assignments computed before reaching the minimum pre fixed-point assignment of the example SDG. The global algorithm runs in polynomial time, as stated by the following theorem.

Theorem 2. *Computing the minimum post fixed-point assignment of an SDG $G = (V, H, C)$ by repeated application of F takes time $O(|V| \cdot |C| \cdot (|H| + |C|))$.*

Proof. Details are provided in appendix A.2. □

4.1 Local Algorithm for Symbolic Dependency Graphs

In this section we describe a local algorithm for computing A^{min} of an SDG. Here we show an algorithm where the search originates from some initial configuration of interest and which generates the state-space in a need-driven fashion. The interest for local exploration is spurred by the fact that for many model checking questions we are only interested in the fixed-point assignment of a single configuration. Depending on the formula we want to verify, we may in some cases be able to explore a smaller fraction of the reachable state space.

Algorithm 2 takes an SDG $G = (V, H, C)$ and an initial configuration $v_0 \in V$ as input and computes $A^{min}(v_0)$. The algorithm expands upon the idea behind Liu and Smolka's local algorithm [22]. The data-structures are similar, yet the algorithm is adapted to handle assignments that range over $\mathbb{N}_0 \cup \{\bot, \infty\}$. Here $\bot$ is a special element used to indicate that the value of the assignment of a particular configuration is currently unknown.

Algorithm 2: Symbolic Local Algorithm

Input: A SDG $G = (V, H, C)$ and a configuration $v_0 \in V$
Output: $A^{min}(v_0)$
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 $(w, u) := \underset{(w,u) \in T}{\arg\max}\ A(u) + w$
      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 $A(u) \leq k < \infty$ *or* $A(u) < k = \infty$ **then**
      $A(v) := 0$
      if $A(v)$ *was changed* **then**
        $W := W \cup D(v)$
    else
      $D(u) := D(u) \cup \{e\}$
return $A(v_0)$

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

Table 1. Execution of Algorithm 2 on the SDG in Figure 4(a)

Example 2. Table 1 shows that values of assignment A, the waiting list W and the dependency set of each configuration throughout execution of Algorithm 2 on the SDG from Figure 4(a). Every row lists the state of these data-structures for the ith iteration of the while-loop. The column for $D(a)$ has been omitted because it stays empty.

Correctness of Algorithm 2 can be established by extending the loop-invariants for the local algorithm for dependency graphs, shown in [22], by also taking weights into account.

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) $e \in D(u)$ 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) $A(v) = 0$.

Correctness of the local algorithm can be established using these loop-invariants, as shown in [13].

Theorem 3. *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 return value $A(v_0)$ is equal to $A^{min}(v_0)$.*

Proof. Correctness is proved in [13]. □

We point out that the termination argument is not entirely clear-cut. The algorithm may require more than a polynomial number of steps before termination. This is exemplified in Figure 5. For convenience, we name the hyper-edges $a_1, \ldots, a_n, b_1, \ldots, b_n$ and z. Suppose we consider an execution of Algorithm 2, where s_0 is the initial configuration and edges are picked from W in line 2 in agreement with the following 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 assignment of s_0 decreases bit by bit as the sequence $A(s_0) = \infty$, $2^n - 1$, $2^n - 2$, $2^n - 3$, …, 1, 0.

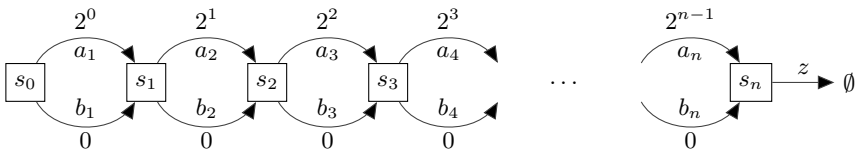


Fig. 5. An SDG where the local algorithm may take exponential running time

Thus, it is possible for Algorithm 2 to take an exponential number of steps before terminating, despite the SDG being polynomial in the size of n. In contrast, the global algorithm is guaranteed to terminate in polynomial time. Nevertheless, experimental results (see Section 10) demonstrate that the local algorithm is in practice significantly more efficient than the global algorithm, notwithstanding its theoretically high complexity. Finally, we believe that the local algorithm is unlikely to exhibit this degenerate behavior if the edges are picked in FIFO or FILO order.

4.2 Model Checking with Symbolic Dependency Graphs

Now we present a symbolic encoding of the WCTL$_\leq$ model checking problem. Once again the problem is reduced to that of computing the minimum pre fixed-point assignment. In this case, however, we construct a symbolic dependency graph. Given a WKS $\mathcal{K}$, a state s of $\mathcal{K}$, and φ a WCTL$_\leq$ formula, the SDG is constructed as shown in Figure 2, except the until formulas follow the rules in Figure 7.

Theorem 4 (Encoding Correctness). *Let $\mathcal{K} = (S, \mathcal{AP}, L, \to)$ be a WKS, $s \in S$ a state, and φ a WCTL$_\leq$ formula. Let G be the constructed symbolic dependency graph rooted with $\langle s, \varphi \rangle$. Then $s \models \varphi$ if and only if $A^{min}(\langle s, \varphi \rangle) = 0$.*

Proof. By structural induction on φ. See appendix A.2. □

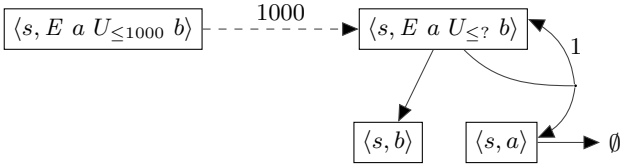


Fig. 6. SDG for the formula $s \models E\ a\ U_{\leq 1000}\ b$ and the WKS in Figure 3

Figure 6 shows an SDG encoding of the formula that was used earlier as an example to demonstrate that the dependency graph encoding is pseduo-polynomial in the bound of a formula, see Figure 3. Notice that the SDG encoding, shown in Figure 6, is much more compact than the dependency graph encoding. In this case we reach the minimum fixed-point assignment after just two iterations of F (see Equation (1)).

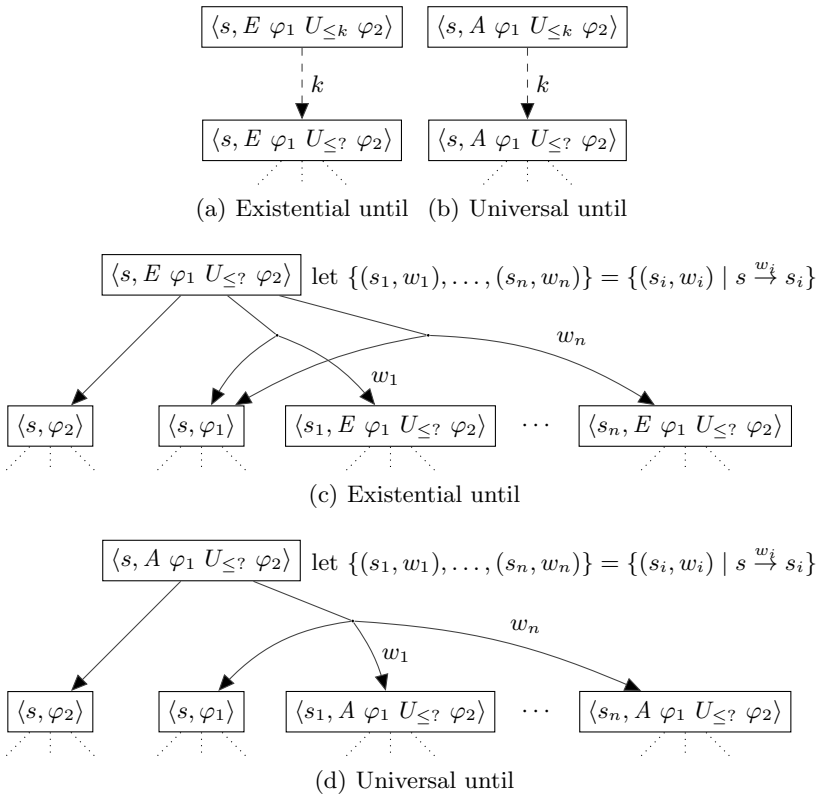


Fig. 7. SDG encoding of existential and universal 'until' formulas

We notice that for a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ and a formula φ, the size of the components of the constructed SDG $G = (V, H, C)$ is $|V| = O(|S| \cdot |\varphi|)$, $|H| = O(|\rightarrow| \cdot |\varphi|)$ and $|C| = O(|\varphi|)$. Due to Theorem 2 and because $|C| \leq |H|$ (this follows from the rules used to construct G), we are able to present the following theorem, stating that global model checking of $\text{WCTL}_{\leq}$ takes polynomial time.

Theorem 5. *Given a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$, a state $s \in S$ and a WCTL$_{\leq}$ formula φ, the model checking problem $s \models \varphi$ is decidable in time $O(|S| \cdot |\rightarrow| \cdot |\varphi|^3)$.*

In the section to follow, we expand upon the idea of symbolic dependency graphs and introduce min-max graphs, which enable us to encode are more expressive properties, i.e. full WCTL.

5 Min-Max Graph

In this section we introduce min-max graphs. With this extended framework, it is possible to handle nested fixed-points and thus model check the full nested WCTL logic. All the same, we are able to perform verification in polynomial time.

Let $\mathbb{N}_\infty = \mathbb{N}_0 \cup \{\infty, -\infty\}$ be the set of nonnegative integers including positive and negative infinity. We define the minimum and maximum value over the empty set as $\min(\emptyset) = \infty$ and $\max(\emptyset) = -\infty$. In the following we have two kinds of edge-types: *weighted edges*, annotated with a weight $w \in \mathbb{N}_0$ and *cover-edges* annotated with a triple $(k, w_1, w_2) \subseteq (\mathbb{N}_\infty \times \mathbb{N}_\infty \times \mathbb{N}_\infty)$.

Intuitively, ∞ corresponds to *true* and $-\infty$ corresponds to *false*. This essentially enables us to encode the boolean values as the bottom and top elements of the lattice, respectively. In between, the domain of $\mathbb{N}_0$ is be interpreted based on the application of the framework. A weighted edge has a potential increase in cost associated with propagating a value back through it. In this way, additive weights can be modelled. Whereas a cover-edge is used to check if the value of the target node is less than some k. One of two values is then offered to the target node, depending on the validity of the inequality.

The intention here is to be able to handle nested fixed points. As mentioned in the following, we only operate in terms of maximum fixed-points. Minimum fixed-points are instead emulated through the usage of these cover-edges, as they give us the ability to swap the monotonicity of assignments. Note that this forces us to impose an ordering on nodes for the fixed-point to be well-defined, hence the notion of *cover-level* being related to the alternation-depth of the fixed-point operator.

Definition 4 (Min-Max Graph). *A min-max graph (MMG) is a directed graph $G = (V_{min}, V_{max}, E, T, c\ell)$, where*

- *V_{min} is a finite set of min-nodes,*
- *V_{max} is a finite set of max-nodes,*
- *$E \subseteq (V_{min} \cup V_{max}) \times (V_{max} \cup V_{min})$ is a set of edges,*
- *$T : E \to \mathbb{N}_0 \cup (\mathbb{N}_\infty \times \mathbb{N}_\infty \times \mathbb{N}_\infty)$ is a mapping from edges to types, and*
- *$c\ell : V_{min} \cup V_{max} \to \mathbb{N}_0$ is a mapping from nodes to cover-level, where*
  - *for all $(u, v) \in E$ we have that $c\ell(u) \leq c\ell(v)$, and*
  - *whenever $T(u, v) \in (\mathbb{N}_\infty \times \mathbb{N}_\infty \times \mathbb{N}_\infty)$ it holds that $c\ell(u) < c\ell(v)$.*

Let $G = (V_{min}, V_{max}, E, T, c\ell)$ be an MMG. We write $u \to v$ whenever $(u, v) \in E$. When $T(u, v) = w$ we say that the edge from u to v has weight w, denoted $u \xrightarrow{w} v$. Similarly, if $T(u, v) = (k, w_1, w_2)$, we say that the edge from u to v is a cover-edge, written as $u \xrightarrow{k \mapsto w_1; w_2} v$, where the triple (k, w_1, w_2) is a ternary operator that returns w_1 if the cover-condition k is satisfied and w_2 otherwise.

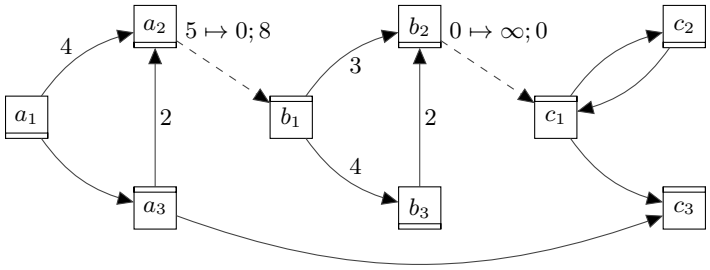


Fig. 8. Example of a min-max graph

Example 3. Figure 8 illustrates an MMG, where min/max nodes are depicted as squares with double border in the bottom or top, respectively. Edges are drawn as solid lines where the weight is omitted if it is zero, while cover-edges are drawn with a dashed line annotated with the cover-condition. For example, we have $a_1 \xrightarrow{0} a_2$ and $a_2 \xrightarrow{5\mapsto 0;\infty} b_1$.

Cover-levels of the nodes in Figure 8 may be given as follows.

$$c\ell(a_1) = c\ell(a_2) = c\ell(a_3) = 0$$
$$c\ell(b_1) = c\ell(b_2) = c\ell(b_3) = 1$$
$$c\ell(c_1) = c\ell(c_2) = c\ell(c_3) = 2$$

The set of nodes with a cover-level of at least j is denoted CL_j, formally written

$$CL_j = \{u \in V_{min} \cup V_{max} \mid c\ell(u) \geq j\}$$

An assignment $A_j : CL_j \to \mathbb{N}_\infty$ of a cover-level j is a mapping from nodes with a cover-level of at least j to values. The set of all assignments over cover-level j is denoted $Assign_j$. We define the partial order $\sqsubseteq$ over $Assign_j$ such that $A_j \sqsubseteq A'_j$, if $A_j(v) \leq A'_j(v)$ for all $v \in CL_j$.

An assignment $A_j \in Assign_j$ is a post fixed-point assignment if $A_j \sqsubseteq F_j(A_j)$, where F_j is defined in the following and A^{max}_{j+1} is the maximum post fixed-point assignment of cover-level $j+1$. Notice that for the maximum cover-level m, we have that $CL_{m+1} = \emptyset$, thus, A^{max}_{m+1} is the empty mapping, and F_j is well-defined.

$$wgt_j(u, v, A) = \begin{cases} w + A(v) & \text{if } u \xrightarrow{w} v \text{ and } c\ell(v) = j \\ w + A^{max}_{j+1}(v) & \text{if } u \xrightarrow{w} v \text{ and } c\ell(v) > j \\ w_1 & \text{if } u \xrightarrow{k\mapsto w_1;w_2} v \text{ and } A^{max}_{j+1}(v) < k \\ w_2 & \text{if } u \xrightarrow{k\mapsto w_1;w_2} v \text{ and } A^{max}_{j+1}(v) \geq k \end{cases} \tag{2}$$

$$F_j(A)(u) = \begin{cases} A^{max}_{j+1}(u) & \text{if } c\ell(u) > j \\ \min\{wgt_j(u, v, A) \mid u \to v\} & \text{if } u \in V_{min} \\ \max\{wgt_j(u, v, A) \mid u \to v\} & \text{if } u \in V_{max} \end{cases} \tag{3}$$

Because $(Assign_j, \sqsubseteq)$ is a complete lattice and F_j is a monotonic function, we have by the Knaster-Tarski fixed-point theorem that there exists a unique maximum post fixed-point assignment, denoted A_j^{max}.

Furthermore, we have that the maximum post fixed-point assignment A_j^{max} of CL_j can be computed by repeated application of the function F_j, starting with the assignment $A_j^0(v) = \infty$ for all $v \in CL_j$. Because $CL_i \subseteq CL_j$ for $i \leq j$ we have that A_0^{max} is precisely the maximum post fixed-point assignment of every cover-level i. For convenience we write A^{max} rather than A_0^{max}.

Intuitively, computation of the maximum fixed-point assignment A_0^{max}, is carried out in a bottom-up fashion, starting with A_m^{max} where m is the highest cover-level. A global algorithm based off the functor is introduced in Section 6.

A_2^{max}			
i	c_1	c_2	c_3
0	∞	∞	∞
1	∞	∞	$-\infty$
2	∞	∞	$-\infty$

A_1^{max}			
i	b_1	b_2	b_3
0	∞	∞	∞
1	∞	0	∞
2	∞	0	2
3	6	0	2
4	6	0	2

A_0^{max}			
i	a_1	a_2	a_3
0	∞	∞	∞
1	∞	8	∞
2	12	8	10
3	10	8	10
4	10	8	10

Table 2. Fixed-point computation of the three cover-levels in Example 3

Example 4. Table 2 lists the computation steps for determining the maximum post fixed-point assignment of the MMG in Example 3. As presented in Example 3, there are three cover-levels: $CL_2 = \{c_1, c_2, c_3\}$, $CL_1 = \{b_1, b_2, b_3\} \cup CL_2$ and $CL_0 = \{a_1, a_2, a_3\} \cup CL_1$.

We start with A_2^{max}, as we require A_2^{max} in order to compute A_1^{max}, which in turn is needed to compute A_0^{max}. In the tables for A_1^{max} and A_0^{max}, we omit nodes that were already computed in A_2^{max} and A_1^{max}, respectively, since the assignments of these nodes do not change.

6 Global MMG Fixed-Point Algorithm

In this section we present a global algorithm for fixed-point computation on MMGs. The algorithm is based on the functor defined in Equation (3) and it makes use of an auxiliary function *wgt*, shown below in Equation (4). It may be observed that this function is similar to the function wgt_j in Equation (2), which is used in the functor F_j.

$$wgt(u, v, A) = \begin{cases} w + A(v) & \text{if } u \xrightarrow{w} v \\ w_1 & \text{if } u \xrightarrow{k \mapsto w_1; w_2} v \text{ and } A(v) < k \\ w_2 & \text{if } u \xrightarrow{k \mapsto w_1; w_2} v \text{ and } A(v) \geq k \end{cases} \tag{4}$$

Algorithm 3: MMG Global Algorithm

Input: MMG $G = (V_{min}, V_{max}, E, T, c\ell)$
Output: A^{max}
for $v \in V_{min} \cup V_{max}$ **do**
  $A(v) := \infty$ `// Assignment`
Let j be the maximum cover-level
`Main-loop:` **while** $j \geq 0$ **do**
  $W := \{u \in V_{min} \cup V_{max} \mid c\ell(u) = j\}$
  `Repeat:` **repeat**
    for $u \in W$ **do**
      `Assign:` $A(u) := \begin{cases} \min\{wgt(u, v, A) \mid u \to v\} & \text{if } u \in V_{min} \\ \max\{wgt(u, v, A) \mid u \to v\} & \text{if } u \in V_{max} \end{cases}$
  until A *is unchanged*;
  $j := j - 1$
return A

Algorithm 3 takes as input an MMG G and outputs the maximum fixed-point assignment of G. Initially every node is assigned the value ∞, we then set $j = m$ where m is the maximum cover-level for any node in G. Subsequently, the maximum post fixed-point assignment is computed for all nodes one cover-level at the time.

In each iteration of the `Main-loop`, the waiting list W is instantiated to contain the set of nodes at cover-level j. The repeat-loop then applies F_j until a fixed-point is reached. At this point the the maximum fixed-point assignment of the nodes in W has been computed, and j is decremented before the `Main-loop` repeats.

Example 5 (Global Algorithm). Table 3 shows the execution of Algorithm 3 on the MMG in Example 8. The column labelled i shows the iteration number of the `Repeat` loop, for each cover-level j during execution of the algorithm.

j		$A(u)$								
	i	a_1	a_2	a_3	b_1	b_2	b_3	c_1	c_2	c_3
2	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	1	∞	∞	∞	∞	∞	∞	∞	∞	$-\infty$
2	2	∞	∞	∞	∞	∞	∞	∞	∞	$-\infty$
1	0	∞	∞	∞	∞	0	2	∞	∞	$-\infty$
1	1	∞	∞	∞	6	0	2	∞	∞	$-\infty$
1	2	∞	∞	∞	6	0	2	∞	∞	$-\infty$
0	0	∞	8	10	6	0	2	∞	∞	$-\infty$
0	1	10	8	10	6	0	2	∞	∞	$-\infty$
0	2	10	8	10	6	0	2	∞	∞	$-\infty$

Table 3. Execution of the global algorithm on Example 8

Lemma 2 (Global Algorithm Invariant). *It holds invariantly in Algorithm 3 that whenever $j < c\ell(u)$, we have that $A(u) = A^{max}(u)$ for any node u.*

Proof. We show $j < c\ell(u)$ implies $A(u) = A^{max}(u)$ for every node u by induction in the cover-level $c\ell(u)$ of u, starting from the maximum cover-level m.

Initialization ($c\ell(u) = m$): In this case we show that when j becomes less than m, it holds that $A(u) = A^{max}(u)$ for every node u with cover-level $c\ell(u) = m$. We quite simply observe that when j becomes less than m, after the `Repeat` loop, we have that `Assign` has been executed for every node with cover-level m, until no changes occur in A.
It is easy to see that an execution of `Assign` for each node with cover-level m is equivalent to an application of the functor F_j (Equation (3)). Notice that the auxiliary function wgt_j (Equation (2)), as used in the functor, may reference $A_i^{max}(v)$ of a node v with a greater cover-level $i > m$, however, because m is the maximum cover-level, no such node v exists.

Maintenance ($c\ell(u) < m$): This case follows by arguments similar to the base case. Yet, in this case the auxiliary function wgt_j may reference $A_i^{max}(v)$ of nodes v with a greater cover-level $i > c\ell(u)$. But in this case it follows by induction that $A(v) = A^{max}(v)$, for any node v with cover-level $i > c\ell(u)$. Thus, the execution of `Assign` for every node u with cover-level $c\ell(u)$ is equivalent to an application of the functor. By repeated application of the functor, as ensured by the guard of the `Repeat` loop, the maximum post fixed-point must be reached in a finite number of iterations.

□

Theorem 6 (Global Algorithm Correctness). *Given an MMG G, Algorithm 3 returns A^{max} of G.*

Proof. Correctness follows from Lemma 2, since $j < 0$ upon termination. □

Theorem 7 (Global Algorithm Complexity). *The running time of Algorithm 3 in a MMG $G = (V_{min}, V_{max}, E, T, c\ell)$ takes $O(n^2 + n \cdot |E|)$ time, where $n = |V_{min} \cup V_{max}|$.*

Proof. Let $c_j = |CL_j \setminus CL_{j+1}|$ denote the number of nodes u with cover-level $c\ell(u) = j$. We have n nodes partitioned into m cover-levels, hence, $n = c_0 + c_1 + \ldots + c_m$. We have m iterations of the `Main-loop`, and in the jth iteration we have c_j iterations of `Repeat` each taking $O(c_j + |E|)$ time. This gives us a complexity as given in Equation (5), which we may rewrite as in Equations (5) through (7). Because $n = c_0 + c_1 + \ldots + c_m$, we may also write Equation (8).

$$O\left(\sum_{j=0}^{m} c_j \cdot (c_j + |E|)\right) \tag{5}$$

$$O\left(\sum_{j=0}^{m} c_j^2 + c_j \cdot |E|\right) \tag{6}$$

$$O\left(\left(\sum_{j=0}^{m} c_j^2\right) + \left(\sum_{j=0}^{m} c_j\right) \cdot |E|\right) \tag{7}$$

$$O\left(\left(\sum_{j=0}^{m} c_j^2\right) + n \cdot |E|\right) \tag{8}$$

Obviously, we may write as in Equation (9), which gives us Equation (10), as c_j is nonnegative for any cover-level j.

$$\left(\sum_{j=0}^{m} c_j\right)^2 = \left(\sum_{j=0}^{m} c_j^2\right) + 2 \cdot \left(\sum_{j=0}^{m} c_j\right) \tag{9}$$

$$\sum_{j=0}^{m} c_j^2 \leq \left(\sum_{j=0}^{m} c_j\right)^2 \tag{10}$$

We now substitute Equation (10) into Equation (8) giving us Equation (11). Again, we have that $n = c_0 + c_1 + \ldots + c_m$, which gives us Equation (12), thus, proving Theorem 7.

$$O\left(\left(\sum_{j=0}^{m} c_j\right)^2 + n \cdot |E|\right) \tag{11}$$

$$O(n^2 + n \cdot |E|) \tag{12}$$

□

7 Local MMG Fixed-Point Algorithm

In this section we present a local algorithm for fixed-point computation on min-max graphs. The basic idea behind the algorithm is related to the local algorithms presented earlier. As we saw for the global MMG algorithm, the fixed-point was computed in a bottom-up manner. The strategy of the local algorithm is to perform a top-down search, starting from the initial node v_0. The intent is to only explore as little as possible, or rather, only explore the nodes in the graph that are actually needed in order to compute the value for v_0.

Algorithm 4: Local Fixed-Point Algorithm for Min-Max Graphs

Input: MMG $G = (V_{min}, V_{max}, E, T, c\ell)$, node $v_0 \in V_{min} \cup V_{max}$
Output: $A^{max}(v_0)$

```
Init:  for v ∈ V_min ∪ V_max do
           A(v) := ∞                                   // Assignment
           X(v) := ff                                  // Explored
           D(v) := ∅                                   // Dependency set
       W_i := {v_0}     , where i = cℓ(v_0)            // Waiting list at top level
       X(v_0) := tt
Main-loop:  while W_i ≠ ∅, where i = cℓ(v_0) do
           Pick u ∈ W_j for some j                     // Heuristic choice of j and u
Case min:  if u ∈ V_min then
               A(u) := min{weight(u, v) | u → v}
               for v s.t. u → v do
                   explore(u, v)
               if ∀v s.t. u → v we have finished(u, v) = tt then
                   W_j := W_j \ {u}
Case max:  else if u ∈ V_max then
               A(u) := max{weight(u, v) | u → v}
               if ∃v s.t. u → v then
                   Pick v, s.t. weight(u, v) = A(u)
                   explore(u, v)
                   if finished(u, v) = tt then
                       W_j := W_j \ {u}
               else
                   W_j := W_j \ {u}
Propagate:  if A(u) was changed then
               for v ∈ D(u) do
                   W_i := W_i ∪ {v}     , where i = cℓ(v)
       return A(v_0)
```

However, because we are dealing with nested fixed-points, it is not as straightforward to derive an efficient local algorithm as for the unnested setting. Namely, we must ensure that the assignments are only propagated upwards when it is

safe to do so. On the other hand, it is desirable to propagate assignments as early as possible, as this may lead to early termination. Due to these aspects, the algorithm becomes slightly more elaborate than, for instance, the local symbolic algorithm. Therefore, correctness of the algorithm is not immediately apparent, so we provide a rigorous proof of correctness.

Algorithm 4 lists the pseduo code for the local algorithm for min-max graphs. Just as for the local symbolic algorithm, nodes are picked according to some pre-defined heuristic or regular search strategy, e.g. depth-first or breadth-first search. Moreover, Algorithm 4 can perform an exponential number of steps, as it is easy to construct an example similar to that discussed in Section 4.1. Further, the algorithm may not terminate if a degenerate strategy is used to select nodes from the waiting list. Nevertheless, we shall prove that the algorithm does indeed terminate, provided that a fair strategy is used.

The algorithm maintains a number of data-structures throughout execution. $A(u)$ is the current assignment of a node u. It is initialized to ∞ and may decrease to $-\infty$. To ensure monotonicity, the value of $A(u)$ may only decrease. $D(u)$ is the dependency set of node u keeps track of the nodes that must be re-processed whenever the assignment of u changes. $X(u)$ is a marking indicating whether or not node u has been explored. W_i is the waiting list containing nodes at cover-level i.

Listing 5: Auxiliary Functions for Algorithm 4

```
Aux. 1: function explore(u, v):
            D(v) := D(v) ∪ {u}
            if X(v) = ff then
                X(v) := tt
                W_j := W_j ∪ {v}    , where j = cℓ(v)

Aux. 2: function weight(u, v):
            if u --w--> v then
                return A(v) + w
            else if u --k↦w1;w2--> v then
                return
                    w1                  if A(v) < k
                    w2                  if A(v) ≥ k, X(v) = tt and W_i = ∅ , where i = cℓ(v)
                    max{w1, w2}         otherwise

Aux. 3: function finished(u, v):
            return
                tt   if u --w--> v and cℓ(v) = cℓ(u)
                tt   if u --k↦w1;w2--> v and A(v) < k
                tt   if X(v) = tt and W_i = ∅ , where i = cℓ(v)
                ff   otherwise
```

The algorithm starts with the initialization of the data-structures in the `Init`-loop. Every node u is assigned the value ∞, marked as unexplored and the dependency set is initially empty. The top-most waiting list is instantiated with the initial node v_0, which is also marked as explored. Afterwards the `Main-loop` is entered and a node u is picked from some waiting list W_j. Depending on whether u is a min- or max-node, the control enters `Case min` or `Case max`, respectively. The first statement executed for node u is then the assignment of the minimum or maximum value (resp.) of the function `weight`, over the set of edges originating from u.

This auxiliary function is defined in Listing 5. Given a source node u and a target node v, it essentially returns the value offered by v, depending on the type of $u \to v$. If it is simply a weighted edge, the assignment of v, plus the weight is returned.

Whereas, if it is a cover-edge, there are three cases to consider. The first case corresponds to the cover-condition being satisfied, and hence w_1 is returned. The second returns if the conver-condition is not satisfied, with the additional requirement that the target node v has been explored and there are no nodes in the waiting list for nodes at the same cover-level as v. This ensures that by returning w_2 the monotonicity requirement is not violated, since there are no nodes that can satisfy the cover-condition at a later point.

The third case returns the maximum of w_1 and w_2, as the final value offered is at least as large either of the two values. The important point here is that it should not offer the smallest value, as this could potentionally violate the monotonicity of A, since there is no way to predict that the final value offered by the cover-edge is, in fact, the smallest.

If we consider `Case min`, then after $A(u)$ has been assigned a value, every outgoing edge from u is explored, using the auxiliarly function `explore` in Listing 5. Note that *all* nodes v, where $u \to v$ must be explored since every node is initially assigned the maximum value (∞), and it is not possible to know which node results in the smallest assignment, thus u must depend on every such node v. The function `explore` adds the source node u to the dependency set of the target node v, hence $u \in D(v)$, which ensures that u is reprocessed if $A(v)$ changes. If v is not marked as explored, it is marked and added to the waiting list corresponding to its cover-level.

After having invoked `explore`(u, v) on every node v connected to u, the algorithm checks if every edge $u \to v$ is *finished*, according to the value returned by the boolean function `finished`(u, v). The function `finished`(u, v) returns tt if the fixed-point assignment of u does not depend on a further decrease of $A(v)$ for a node v at a greater cover-level. In the positive case, the node u is removed from the waiting list.

The function `finished` is also provided in Listing 5. We now examine its return cases. First of all, a node v, such that $u \overset{w}{\to} v$, is finished if both u and v are on the same cover-level. In this case $W_{c\ell(u)} = \emptyset$ ensures that neither u or v will be reprocessed. Secondly, given the cover-edge $u \xrightarrow{k \mapsto w_1; w_1} v$, a node v is finished if the value assigned to it is less than k, i.e. the cover-condition is

satisfied. Because assignments are strictly decreasing, it can never be falsified. Consequently, changes to v cannot affect u, hence, v is finished w.r.t u. In the third case, we have that node v is finished if it has been explored, or rather, it has been in the waiting list W_i at least once and the waiting list is now empty. In other words, there are no more nodes for the algorithm to process at the given cover-level i. If none of these cases are satisfied, the node v is regarded to be unfinished, so the source node u is not yet ready to be removed from the waiting list, since u is waiting for more information regarding the fixed-point assignment on a greater cover-level.

`Case max` is slightly different than `Case min`. After the node u is assigned a value (possible $-\infty$ if $u \not\to v$ for any node v), the algorithm checks if there is a node v, such that $u \to v$. If this is the case, some node that assigns the maximum value to $A(u)$ is picked and explored. It is sufficient to just consider the node v, as it gives rise to the largest assignment, hence the maximum value of $A(u)$ can only become smaller if the value of v decreases. Should the value of v change, then u will at some point be re-assigned and another 'largest' node may be explored. If the node v is finished, it is safe to remove u from queue, as we are guaranteed that the maximum value assigned to u remains the same. If u has no outgoing edges, it is simply removed from the queue because its value can never change. Note also that in this case $A(u) = -\infty$.

Finally, `Propagate` is the part of the algorithm responsible for propagating values in the graph whenever the assignment of the node u under consideration is changed. Observe that every node v that depends on the value of u is added to the waiting list corresponding to its respective cover-level. Now the `Main-loop` may perform another iteration. Once the top-most queue $W_{c\ell(v_0)}$ is empty, the maximum post-fixed point assignment of v_0 is returned.

Lemma 3 (Local Algorithm Invariants). *the following invariants holds for any node u, where $i = c\ell(u)$.*

A) $A^{max}(u) \leq A(u)$,
B) if $u \in V_{min}$ *and* $X(u) = tt$, *then either*
  1) $u \in W_i$, *or*
  2) $A(u) = \min\{wgt_i(u, v, A) \mid u \to v\}$ *and* $u \in D(v)$ *for all* v *s.t.* $u \to v$.
C) if $u \in V_{max}$ *and* $X(u) = tt$, *then either*
  1) $u \in W_i$,
  2) $A(u) = wgt_i(u, v, A)$ *and* $u \in D(v)$ *for some* v *where* $wgt_i(u, v, A) = x$ *and such that* $x \geq wgt_i(u, v', A)$ *for all* $u \to v'$, *or*
  3) $A(u) = -\infty$ *and* $\nexists v$ *s.t.* $u \to v$.
D) if $X(u) = tt$ *and* $W_i = \emptyset$ *then* $A(u) = A_i^{max}(u)$,
E) $wgt_i(u, v, A) \leq$ `weight(`u`, `v`)`, *and*
F) if `finished(`u`, `v`)` $= tt$ *then* $wgt_i(u, v, A) =$ `weight(`u`, `v`)`.

where wgt_i is the auxiliary function used in the functor (See Equation (2)).

Proof. To show that invariants (A) through (F) hold for every iteration of the `Main-loop`, we show that these invariants hold initially, and that they are preserved by `Main-loop`.

The invariants hold initially: From the section of Algorithm 4 labelled `Init` it is easy to see that invariant (A) holds, as $A^{max}(u) \leq \infty$ for all nodes v. Furthermore, we observe that invariants (B), (C) and (D) hold vacuously, for all nodes other than v_0, since we have $X(v) = \mathit{ff}$ for all v s.t. $v \neq v_0$. If $v_0 \in V_{min}$ then Invariant (B) holds by the first case, the same argument applies for Invariant (C) if $v_0 \in V_{max}$.

As `finished(`u`, `v`) =` ff for all nodes $u \to v$, we have that invariant (F) holds vacuously. It is also easy to observe that `weight(`u`, `v`) =` ∞ for all $u \to v$ that are weighted edges. And through simple case analysis it can be observed that `weight(`u`, `v`) =` $\max\{w_1, w_2\}$ if $u \xrightarrow{k \mapsto w1;w2} v$. Hence showing that invariant (E) holds. Thus, we conclude that invariants (A) through (F) hold initially.

The invariants are preserved: Assuming that invariants (A) through (F) hold before an iteration of the `Main-loop` we now show that they also hold after.

Invariant (A) To show that Invariant (A) holds after an iteration of the `Main-loop`, assuming invariants (A) through (D) holds before the iteration, we consider the following two cases.

`Case min` In this case we must show that $A^{max}(u) \leq \min\{$`weight(`u`, `v`)` $\mid u \to v\}$. From Invariant (E) we observe that $wgt_j(u, v, A) \leq$ `weight(`u`, `v`)`, which substituted into the functor from Equation (3), gives us $F_j(u, A) \leq \min\{$`weight(`u`, `v`)` $\mid u \to v\}$. Thus, it must be the case that $A^{max}(u) \leq A(u)$, since F_j is a monotonically decreasing function.

`Case max` This case is similar to the `Case min`, and $A^{max}(u) \leq A(u)$ is easily verified using Invariant (E).

Invariant (B) Again we consider the `Case min` and `Case max` from the `Main-loop` to prove that Invariant (B) is preserved.

`Case min` In this case we show that Invariant (B) is preserved for any node v. To do this we let u be the node picked in the `Main-loop` and consider the four following cases.

If $v = u$: In this case v is in W_j, so initially Invariant (B) must hold by the first case. If for all successors v', we have that `finished(`u`, `v'`) =` tt, then u will be removed from W_j. However, if `finished(`u`, `v'`) =` tt for all successors v', then it follows from Invariant (F) that $wgt_j(u, v', A) =$ `weight(`u`, `v'`, `A`)`. Which by substitution into the second case of Invariant (B) shows that this case holds.

If $v \to u$: In this case we have that if v is satisfied by the first case of Invariant (B), then this is preserved. However, if v is satisfied by the second case, and $A(u)$ is changed, then v may not satisfy the second case of Invariant (B) anymore. However, if $A(u)$ is changed and v was satisfied by the second case, then $v \in D(u)$ and v will be added to W_i, where $i = c\ell(v)$, in the `Propagate` section following `Case min`. Thus, if v satisfies Invariant (B) by the second case, then either this is preserved, or v will satisfy Invariant (B) by the first case.

If $u \to v$: In this case we have that `explore(u, v)` may change $X(v)$. However, if this happens `explore(u, v)` will also add v to W_j, in which case v would satisfy Invariant (B) by the first case. If $X(v)$ is unchanged, then Invariant (B) will be preserved trivally.

Otherwise: In this case neither $A(v)$ or $X(v)$ is changed, and v is not removed from W_j. Thus, Invariant (B) is preserved for v.

`Case max` In this case we again have to consider different nodes v and show that Invariant (B) is preserved for v. As before, we let u be the node picked in the `Main-loop` and consider the four following cases.

If $v = u$: In this case we have that $v \in V_{max}$, thus, Invariant (B) holds vacuously.

If $v \to u$: This case is the same as the $v \to u$ case for `Case min`, as Invariant (B) is preserved by the `Propagate` section.

If $u \to v$: This case is the same as the $u \to v$ case for `Case min`, as changes to $X(v)$ in `explore(u, v)` will also add v to W_j.

Otherwise: In this case neither $A(v)$ or $X(v)$ is changed, and v is not removed from W_j. Thus, Invariant (B) is preserved for v.

Invariant (C) Again we consider the `Case min` and `Case max` from the `Main-loop` to prove that Invariant (C) is preserved.

`Case min` In this case we show that Invariant (C) is preserved for any node v. Let u be the node picked in the `Main-loop`, we now consider the four following cases.

If $v = u$: In this case we have that $v \in V_{min}$, thus, Invariant (C) holds vacuously.

If $v \to u$: This case is the same as the $v \to u$ case for `Case min`, as Invariant (C) is preserved by the `Propagate` section.

If $u \to v$: This case is the same as the $u \to v$ case for `Case min`, as changes to $X(v)$ in `explore(u, v)` will also add v to W_j.

Otherwise: In this case neither $A(v)$ or $X(v)$ is changed, and v is not removed from W_j. Thus, Invariant (C) must be preserved for v.

`Case max` As before we now show that Invariant (C) is preserved for any node v. Let u be the node picked in the `Main-loop`, we now consider the four following cases.

If $v = u$: In this case v is in W_j, so initially Invariant (C) must hold by the first case. If for some successor v', where $v' = \arg\max_{u \to v'}$ `weight(u, v', A)`, we have that `finished(u, v')` $= tt$, then u may be removed from W_j. However, if `finished(u, v')` $= tt$, then it follows by Invariant (F) that $wgt_j(u, v', A) =$ `weight(u, v', A)`. Which by substitution into the second case of Invariant (C) shows that this case holds. If u has no successors, then $A(u) = -\infty$ and Invariant (C) holds by the third case, which allows us to remove u from W_j.

If $v \to u$: This case is the same as the $v \to u$ case for `Case min`, as Invariant (C) is preserved by the `Propagate` section.

If $u \to v$: This case is the same as the $u \to v$ case for `Case min`, as changes to $X(v)$ in `explore(`u`,` v`)` will also add v to W_j.

Otherwise: In this case neither $A(v)$ or $X(v)$ is changed, and v is not removed from W_j. Thus, Invariant (C) is preserved for v.

Invariant (D) To show that Invariant (D) is preserved, we show that it holds for any node u with cover-level $i = c\ell(u)$.

If $W_i \neq \emptyset$ or $X(u) \neq tt$, then Invariant (D) holds vacuously. This leaves us to show that if $W_i = \emptyset$ and $X(u) = tt$, then $A(u) = A^{max}(u)$. We already have from Invariant (A) that $A^{max}(u) \leq A(u)$, leaving us to show that $A(u)$ is also a post fixed-point assignment.

To show that $A(u)$ is a post fixed-point assignment. We must show that $A(u) = F_i(A)(u)$. From the definition of the functor (Equation (3)) we have following three cases to show.

i) if $c\ell(u) > i$ then $A(u) = A^{max}_{i+1}(u)$,
ii) if $u \in V_{min}$ then $A(u) = \min\{wgt_i(u, v, A) \mid u \to v\}$, and
iii) if $u \in V_{max}$ then $A(u) = \max\{wgt_i(u, v, A) \mid u \to v\}$.

Case (i): This case holds vacuously as Invariant (D) only mentions nodes u with cover-level $i = c\ell(u)$.

Case (ii): As we have $X(u) = tt$ and $W_i = \emptyset$, Invariant (B) must be satisfied by the second case, which says $A(u) = \min\{wgt_i(u, v, A) \mid u \to v\}$.

Case (iii): Again, we have that $X(u) = tt$ and $W_i = \emptyset$, which gives us that Invariant (C) must be satisfied by either the second or third case. If satisfied by the third case, then $A(u) = -\infty = \max(\emptyset)$, hence (iii) must hold. If Invariant (C) is satisfied by the second case then we have some v', s.t. $A(u) = wgt_i(u, v', A)$ and $v' = \arg\max_{u \to v} wgt_i(u, v, A)$. It is now easy to see that $A(u) = wgt_i(u, v', A) = \max\{wgt_i(u, v, A) \mid u \to v\}$. Thus, we have that (iii) must hold.

Invariant (E) and (F) Given Invariant (A) and (D), we have that Invariants (E) and (F) follow from straightforward case analysis, comparing `weight(`u`,` v`)` and `finished(`u`,` v`)` in Algorithm 5 with $wgt_j(u, v, A)$ in Equation (2).

Thus, we have proved that the invariants hold. □

Theorem 8 (Local Algorithm Correctness). *Given an MMG G and an initial node v_0, Algorithm 4 returns $A^{max}(v_0)$ if it terminates.*

Proof. Algorithm 4 terminates with $A(v_0)$ if $W_i = \emptyset$ where $i = c\ell(v_0)$. By Invariant (D) from Lemma 3 we have that $W_i = \emptyset$ and $X(v_0) = tt$ implies that $A(v_0) = A^{max}(v_0)$. Thus, the algorithm must return the maximum post fixed-point assignment of v_0, since $X(v_0)$ is set to tt in the `Init` section. □

Theorem 9 (Termination under Fairness Assumption). *Let H be the heuristic strategy for choosing u and j in the* `Main-loop` *of Algorithm 4, if H infinitely often chooses any j then Algorithm 4 always terminates.*

Proof. We prove Theorem 9 by observing that if u is picked from W_j, where j is the largest cover-level j, such that $W_j \neq \emptyset$, then there are two possible cases.

i) some v, s.t. $u \to v$ is added to W_i, where $i > j$, or
ii) u is removed from W_j.

Case (i): This case occurs when v is unexplored, $X(v) = \mathit{ff}$, and v has a cover-level higher than u, $c\ell(v) > c\ell(u)$. In this case we have that $X(v)$ is set to tt and as there is a finite number of nodes and cover-levels, this case cannot happen indefinitely.

Case (ii): If there is no v, s.t. $u \to v$, $X(v) = \mathit{ff}$ and $c\ell(v) > c\ell(u)$, as in (i), then `finished(`u`, `v'`)` $= tt$ for all successors v'. This follows from the fact that j is the largest cover-level for which $W_j \neq \emptyset$, so $W_i = \emptyset$ for all i considered in `finished(`u`, `v'`)`.

When `finished(`u`, `v'`)` $= tt$ for all successors v', then clearly, u is removed from W_j. And as $A(v)$ is decreased or $X(v)$ changed to tt for some node v, whenever, u is added to W_j, then this case cannot happen indefinitely.

Thus, as neither case (i) or (ii) can happen indefinitely, a heuristic strategy H that picks the largest j for which $W_j \neq \emptyset$ infinitely often must lead to a situation where the algorithm terminates. □

Remark 2. It is easy to see that Algorithm 4 does not terminate if the heuristic strategy always picks a node u from some W_j, such that $u \xrightarrow{w} v$ and $v \in W_i$, where $i = c\ell(v)$. Because `finished(`u`, `v`)` will not return tt as long as $W_i \neq \emptyset$, node u is never be removed from W_j. Thus, the algorithm can continue to pick the same u from W_j indefinitely. However, in practice using round-robin approach to pick j remedies this situation.

8 Model Checking with MMGs

In this section we introduce a reduction from the WCTL model checking problem over WKS to maximum post fixed-point computation on MMGs.

Definition 5 (Encoding of Formula Satisfiability). *Given a WKS $\mathcal{K}$, a state s and a WCTL formula ψ, we encode the model checking problem as a min-max graph. Every node, with the exception of intermediate nodes, is labelled with a satisfaction triple $\langle s, \nabla, \varphi \rangle$ consisting of a state $s \in S$, assertion $\nabla \in \{\models, \not\models\}$ and formula φ. The MMG is expanded from the initial node labelled $\langle s, \models, \psi \rangle$ using the rules illustrated in Figures 9 through 17. Min/max nodes are also distinguished in Figures 9 through 17. Nodes may be assigned any cover-level that satisifies the conditions outlined in Definition 4.*

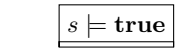
Remark 3. While Definition 5 does not specify any particular assignment of cover-levels for an MMG expanded from an initial node, labelled $\langle s, \models, \psi \rangle$, it is possible to assign cover-levels for any MMG constructed using Definition 5. Equation (13) is one such possible assignment of cover-levels. In Equation (13), a node $\langle s, \nabla, \varphi \rangle$ is assigned cover-level $\ell_\psi(\varphi)$, where ψ is the root formula as given in Definition 5. Intermediate nodes, i.e. unlabelled, are assigned the same cover-level as their parent.

The cover-level of a node, as defined with ℓ_ψ, is well-defined because the annotation of formula φ on a node is a sub-formula of the formula ψ from which the MMG was constructed. There are a few exceptions to this in Figures 12 and 15, but simple case analysis of ℓ_ψ in Equation (13) will reveal that these cases are taken into account.

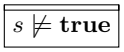
$$\ell_\psi(\varphi) = \begin{cases} 0 & \text{if } \psi = \varphi \\ 1 + \max\{\ell_{\psi_1}(\varphi), \ell_{\psi_2}(\varphi)\} & \text{if } \psi = \psi_1 \wedge \psi_2 \text{ or } \psi = \psi_1 \vee \psi_2 \\ 1 + \ell_{\psi'}(\varphi) & \text{if } \psi = Q X_{\bowtie k}\ \psi' \text{ or } \psi = \neg\psi' \\ 1 & \text{if } \psi = Q\ \varphi_1\ U_{\leq k}\ \varphi_2 \text{ and } \varphi = Q\ \varphi_1\ U_{<?}\ \varphi_2 \\ 1 & \text{if } \psi = Q\ \varphi_1\ W_{\geq k}\ \varphi_2 \text{ and } \varphi = Q\ \varphi_1\ W_{\geq ?}\ \varphi_2 \\ 2 + \max\{\ell_{\psi_1}(\varphi), \ell_{\psi_2}(\varphi)\} & \text{if } \psi = Q\ \varphi_1\ W_{\geq k}\ \varphi_2 \text{ or } \psi = Q\ \varphi_1\ U_{\leq k}\ \varphi_2 \\ -\infty & \text{otherwise} \end{cases} \tag{13}$$

Consider an MMG constructed using Definition 5, Figures 12 and 15 introduce formulas containing "?" as the bound, and Figure 12 uses strict upper bound $<$ instead of $\leq$. This slight abuse of notation is used to introduce *symbolic* nodes. Let φ be a formula in this extended form, i.e. with the abuse of notation introduced above, we then write $\varphi[k/?]$ whenever the symbol "?" is substituted for $k \in \mathbb{N}_0$.

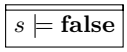
The encoding also introduces unlabelled *intermediate* nodes. Intuitively, they serve the purpose of minimizing/maximizing values between nodes labelled with a satisfaction triple, in order to correctly capture our semantics.



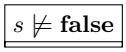
(a) Assertion that **true** holds in s is true



(b) Assertion that **true** does not hold in s is false



(c) Assertion that **false** holds in s is false



(d) Assertion that **false** does not hold in s is true

if $a \in L(s)$

(e) Assertion that proposition a holds in s is true

if $a \notin L(s)$

(f) Assertion that proposition a holds in s is false

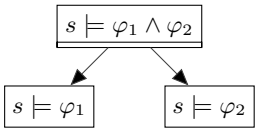
if $a \in L(s)$

(g) Assertion that proposition a does not hold in s is false

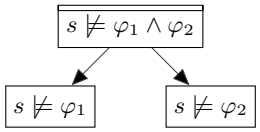
if $a \notin L(s)$

(h) Assertion that proposition a does not hold in s is true

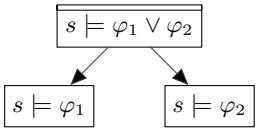
Fig. 9. Boolean encodings



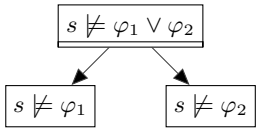
(a) Conjunction



(b) Conjunction w. negative assertion

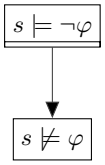


(c) Disjunction

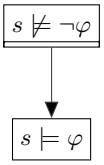


(d) Disjunction w. negative assertion

Fig. 10. Logical connectives

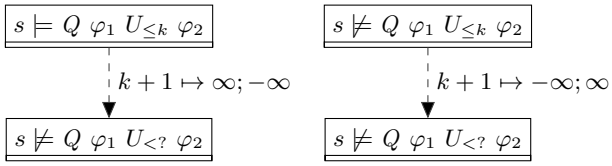


(a) Assertion that negated formula holds



(b) Assertion that negated formula does not hold

Fig. 11. Logical negation



(a) Existential until w. upper bound

(b) Assertion that existential until w. upper bound does not hold

Fig. 12. Existential and universal until path quantifiers with upper bound

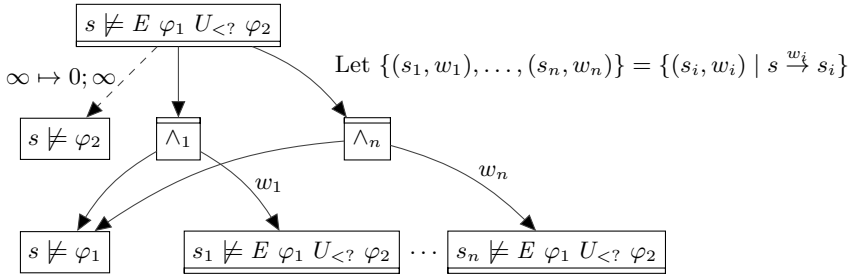


Fig. 13. Symbolic existential until w. upper bound

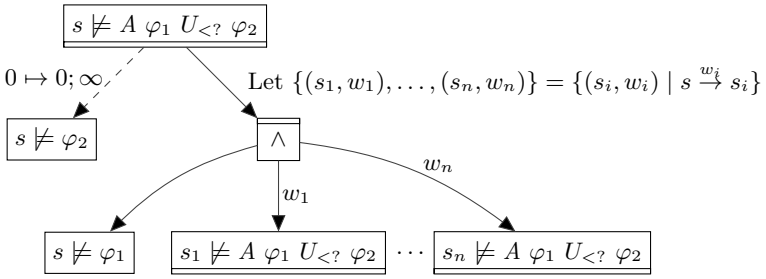
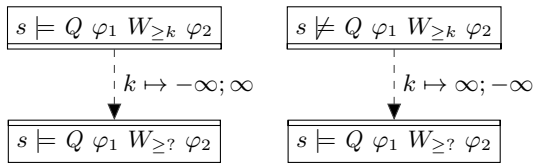


Fig. 14. Symbolic universal until w. upper bound



(a) Weak until w. lower bound

(b) Weak until w. lower bound and negative assertion

Fig. 15. Existential and universal weak until path quantifiers with lower bound

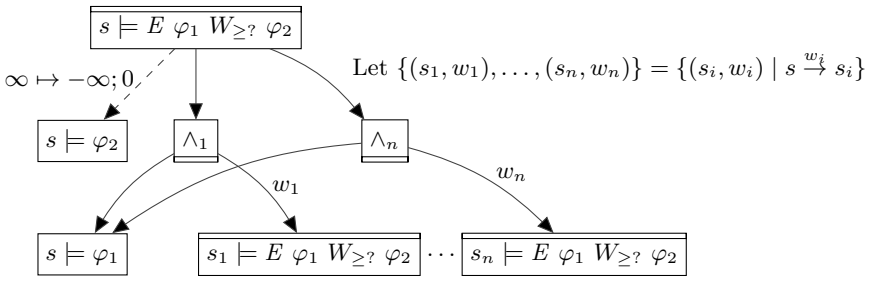


Fig. 16. Symbolic existential weak until w. upper bound

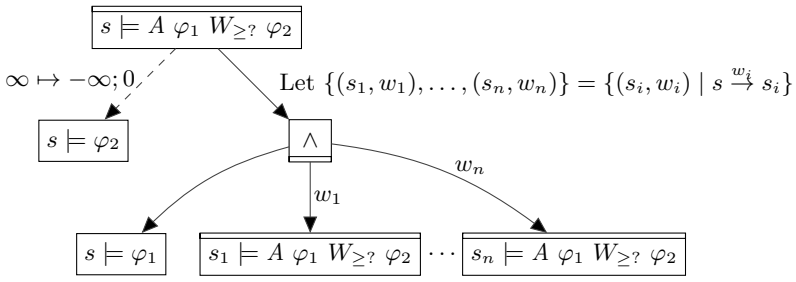
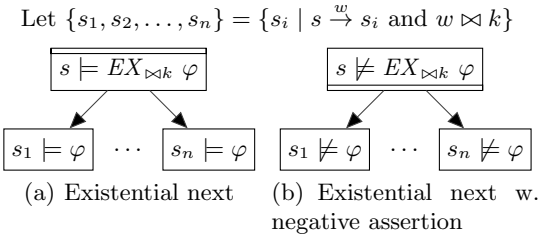


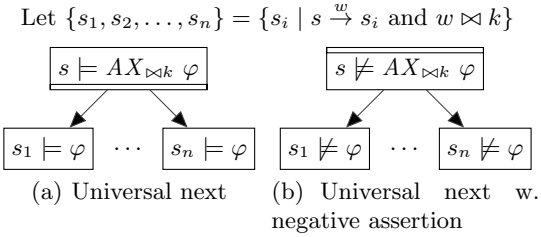
Fig. 17. Symbolic universal weak until w. upper bound



(a) Existential next

(b) Existential next w. negative assertion

Fig. 18. Existential next



(a) Universal next

(b) Universal next w. negative assertion

Fig. 19. Universal next

Definition 6 (Semantic Assignment). *Let G be an MMG constructed according to Definition 5 we define the semantic assignment $\mathcal{A}^{sem}(u)$ of a non-intermediate node $u = \langle s, \nabla, \varphi \rangle$ as*

$$\mathcal{A}^{sem}(\langle s, \nabla, \varphi \rangle) = \max\{k \in \mathbb{N}_0 \mid s \nabla \varphi[k/?]\}$$

where $\max(\emptyset) = -\infty$ *and* $\max(\mathbb{N}_0) = \infty$.

The semantic assignment of an intermediate node v is defined as $F_j(\mathcal{A}^{sem})(v)$, i.e. the min/max combination of the successors of v. This is well-defined as all successors of an intermediate node are non-intermediate (i.e. contain a satisfaction triple).

Remark 4. The semantic assignment of a non-symbolic node, i.e. a node where φ does not contain the symbol "?", is well-defined. Because φ does not contain any "?" symbols, then substituting "?" for k does affect φ, so $\varphi = \varphi[k/?]$. There are two cases for the semantic assignment, depending on the assertion ∇.

For a node with a positive assertion $v = \langle s, \models, \varphi \rangle$, then $\mathcal{A}^{sem}(v) = \infty$ if $s \models \varphi$ and $\mathcal{A}^{sem}(v) = -\infty$, otherwise. Conversely, for a node containing a negative assertion $v = \langle s, \not\models, \varphi \rangle$, then $\mathcal{A}^{sem}(v) = \infty$ if $s \not\models \varphi$ and otherwise $\mathcal{A}^{sem}(v) = \infty$.

Thus, $\{k \in \mathbb{N}_0 \mid s \nabla \varphi[k/?]\} = \{k \in \mathbb{N}_0 \mid s \nabla \varphi\}$, which is then either $\mathbb{N}_0$ or $\emptyset$, depending on whether or not the statement $s \models \varphi$ is true.

8.1 Example Encoding of Weak Until

Figure 21 illustrates the expanded MMG encoding of the formula $\varphi = A\ a\ W_{\geq 1}\ b$ with the intial state s of the WKS shown in Figure 20. The formula asks if for all paths, we always observe a's or after a non-zero transition, b is observed.

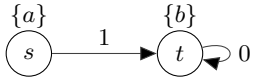


Fig. 20. Weighted Kripke structure with states $S = \{s, t\}$ and $\mathcal{AP} = \{a, b\}$

According to the semantics, we see that $s \models \varphi$. For any run $\sigma = s \xrightarrow{1} t \xrightarrow{0} t \ldots$, the property a is initially satisfied, as $\sigma(0) \models a$ and for any position $p > 0$, it holds that $\sigma(p) \models b$ and $W_\sigma(p) \geq 1$.

In Figure 20 the root node v_0 of the MMG is labelled with the state s and formula φ. Node v_0 is connected by a cover-edge to the symbolic node v_1 immediately below it, according to Figure 15(a). Informally, the cover-edge states: If $A(v_1)$ is strictly less than 1, the value offered is $-\infty$, in this case signifying that that $s \not\models \varphi$. On the other hand, if $A(v_1) \geq 1$, the formula φ is satisfied, so the value ∞ is offered.

The symbolic node v_1 is expanded according to Figure 17. It has a cover-edge to the max-node v_2 and an edge to the intermediate node v_3. The reason

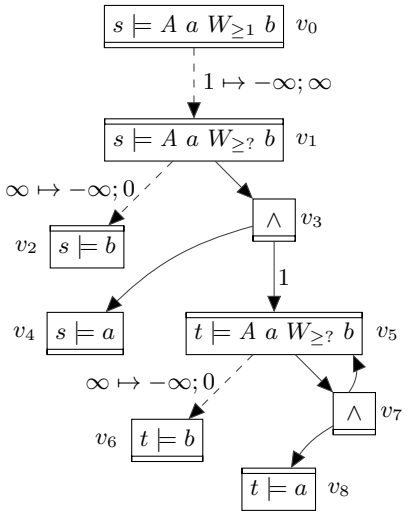


Fig. 21. Unfolding of the weak until formula for the WKS in Figure 20

for v_2 being a max-node is because $b \notin L(s)$, and as a consequence $s \not\models b$. The semantic assignment should therefore be $\mathcal{A}^{sem}(v_2) = -\infty$. This corresponds to the negative case, shown in Figure 9(f). When the fixed-point is computed, $A(v_2)$ thus yields the correct negative value $-\infty$, due to the fact that $\max(\emptyset) = -\infty$.

The cover-edge from v_1 to v_2 checks if the assignment of v_2 is less than ∞, in which the case is interpreted as $-\infty$. If the value is at least ∞, the value 0 is offered, which happens if b is observed in s.

The intermediate node v_3 minimizes the assignments of from v_4, corresponding to the sub-formula $s \models a$ and the symbolic node v_5 for successor state t.

For the state s, we have that $a \in L(s)$, so the statement $s \models a$ is true. The semantic assignment of v_4 must be $\mathcal{A}^{sem}(v_4) = \infty$. Node v_4 encodes this fact accordingly as a min-node, as illustrated in Figure 9(e). Because $\min(\emptyset) = \infty$, the value of $A(v_4)$ is consistent with our observations.

Opposite statements can be made for the successor nodes encoded for state t. Note that the successor nodes of v_5 are due to the recursive application of the rules in Figure 17.

Finally, we see that the value of $A^{max}(v_5)$ becomes 0, due to the assignments of the nodes below it. As mentioned before this value can be viewed as the weight for which the formula φ is satisfied in state t. Thus, the atomic proposition b is observed within a weight of 0. As the assignment of v_5 is propagated backwards, the weight (1) it takes to transition to t from s is added to this value.

Table 4 lists the cover-levels and maximum fixed-point assignment of the nodes in the expanded MMG.

Node u	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
$c\ell(u)$	0	1	2	1	2	1	2	1	2
$A^{max}(u)$	∞	1	$-\infty$	1	∞	0	∞	$-\infty$	$-\infty$

Table 4. Cover-levels and fixed-point assignments for the MMG in Figure 21

Theorem 10 (Encoding Correctness). *Let G be an MMG constructed according to Definition 5, A^{max} be the maximum post fixed-point assignment of G and $\mathcal{A}^{sem}$ be the semantic assignment of G, then it holds that*

$$\mathcal{A}^{sem} = A^{max}$$

Proof. Let $\mathcal{A}^{sem}_j(v) = \mathcal{A}^{sem}(v)$ for all $v \in CL_j$ be the projection of $\mathcal{A}^{sem}$ to nodes with cover-level greater than or equal to j. We now prove Theorem 10 by showing $\mathcal{A}^{sem}_j = A^{max}_j$ for every cover-level j. This is equivalent to Theorem 10, since $\mathcal{A}^{sem}_0 = \mathcal{A}^{sem}$ and $A^{max}_0 = A^{max}$. We show $\mathcal{A}^{sem}_j = A^{max}_j$ by induction in j starting from the maximum cover-level m.

Base Case ($j = m$): In this case we show that $\mathcal{A}^{sem}_m = A^{max}_m$. This amounts to showing that $\mathcal{A}^{sem}_m(u) = A^{max}_m(u)$ for any node $u = \langle s, \nabla, \varphi\rangle$ where $c\ell(u) = m$, which we shall do by structural induction in φ.

- **Case $\varphi =$ true:** For nodes u on the form $u = \langle s, \models, \mathbf{true}\rangle$ we have from Figure 9(a) that u is a min-node with no successors. Thus, $A^{max}_m(u) = \infty$, which clearly proves the case $\mathcal{A}^{sem}_j(u) = \infty$. Also observing that nodes on the form $\langle s, \not\models, \mathbf{true}\rangle$ are max-nodes proves the rest of this case.
- **Case $\varphi =$ false:** This is follows by similar arguments as those given for the previous case, observe that min/max modality is reversed in Figures 9(c) and 9(d).
- **Case $\varphi = a$:** For nodes u on the form $u = \langle s, \nabla, a\rangle$ we have that $A^{max}_m(u) = \mathcal{A}^{sem}_m(u)$ follows by the observation that u is encoded as $\langle s, \nabla, \mathbf{true}\rangle$ if $a \in L(s)$ and otherwise u is encoded as $\langle s, \nabla, \mathbf{false}\rangle$.
- **Case $\varphi = \varphi_1 \vee \varphi_2$:** For nodes u on the form $u = \langle s, \models, \varphi_1 \vee \varphi_2\rangle$ we have from Figure 10(c) that u is a max-node with successors $v_1 = \langle s, \models, \varphi_1\rangle$ and $v_2 = \langle s, \models, \varphi_2\rangle$. By structural induction we have that $A^{max}_m(v_1) = \mathcal{A}^{sem}_m(v_1)$ and likewise for v_2. Thus, as $\mathcal{A}^{sem}_m(v_1) = \infty$ implies $s \models \varphi_1$, we have that $A^{max}_m(u) = \infty$, if and only if $s \models \varphi_1$ or $s \models \varphi_2$, which in turn is the same as $s \models \varphi_1 \vee \varphi_2$, hence, proving the case.
- **Case $\varphi = \varphi_1 \wedge \varphi_2$:** This case follows by arguments similar to those given for the previous case.
- **Case $\varphi = \neg\varphi'$:** For nodes u on the form $u = \langle s, \models, \neg\varphi'\rangle$ we have from Figure 11(a) that $A^{max}_m(u) = A^{max}_m(v)$, where $v = \langle s, \not\models, \varphi'\rangle$. By structural induction we have that $A^{max}_m(v) = \mathcal{A}^{sem}_m(v)$, thus, $A^{max}_m(u) = \infty$ if and only if $s \not\models \varphi'$. For nodes u on the form $u = \langle s, \not\models, \neg\varphi'\rangle$ we have that $A^{max}_m(u) = \mathcal{A}^{sem}_m(v)$ follows by similar arguments.
- **Case $\varphi = EX_{\bowtie k}\ \varphi'$:** For nodes u on the form $u = \langle s, \models, EX_{\bowtie k}\ \varphi'\rangle$ we have from Figure 18(a) that u is a max-node with successor on the form $v_i = \langle s_i, \models, \varphi'\rangle$ where $s \xrightarrow{w_i} s_i$ and $w_i \bowtie k$.

  We now consider the following cases.

  i) $s_i \models \varphi'$ for some s_i where $s \xrightarrow{w_i} s_i$ and $w_i \bowtie k$

  ii) $s_i \not\models \varphi'$ for some s_i where $s \xrightarrow{w_i} s_i$ and $w_i \bowtie k$

  Case (i): In this case we have that there is some $v_i = \langle s_i, \models, \varphi'\rangle$ for which $\mathcal{A}^{sem}_m(v_i) = \infty$, by structural induction it follows that $A^{max}_m(v_i) = \infty$. With this fact and because u is a max-node, we have that $A^{max}_m(u) = \infty$, proving the case as $s \models EX_{\bowtie k}\ \varphi'$ and, hence, $\mathcal{A}^{sem}_m(u) = \infty$.

Case (ii): In this case we clearly have that $s \not\models EX_{\bowtie k}\ \varphi'$, hence, $\mathcal{A}_m^{sem}(u) = -\infty$. We also observe that for all successors v_i, we have that $\mathcal{A}_m^{sem}(v_i) = -\infty$. By structural induction it follows that $A_m^{max}(v_i) = -\infty$. Thus, it must be the case that $A_m^{max}(u) = -\infty$ as A_m^{max} is a post fixed-point assignment.

For nodes u with on the form $u = \langle s, \not\models, EX_{\bowtie k}\ \varphi' \rangle$ we have that $A_m^{max}(u) = \mathcal{A}_m^{sem}(v)$ follows by similar arguments.

Case $\varphi = AX_{\bowtie k}\ \varphi'$: This case follows by arguments similar to those given for the previous case.

As it is easy to see from Figures 13, 14, 16 and 17 that symbolic and intermediate nodes always have a successor that has a cover-edge, no symbolic or intermediate node can have the maximal cover-level m. Thus, we have that $\mathcal{A}_m^{sem} = A_m^{max}$.

Inductive Step ($j < m$): In this case we show that $\mathcal{A}_j^{sem} = A_j^{max}$, assuming that $\mathcal{A}_i^{sem} = A_i^{max}$ for all $i > j$. We do this in two steps by showing that the following statements hold.

A) $\mathcal{A}_j^{sem} \sqsubseteq F_j(\mathcal{A}_j^{sem})$ and

B) $A^{max} \sqsubseteq \mathcal{A}_j^{sem}$

We first show (A) that $\mathcal{A}_j^{sem}$ is a post fixed-point assignment of G. We then proceed to show (B) that $\mathcal{A}_j^{sem}$ is greater than or equal to A^{max}. From these two statements it follows that $\mathcal{A}_j^{sem}$ is the maximal post fixed-point assignment, thus, $\mathcal{A}_j^{sem} = A_j^{max}$.

We now show statement (A), by showing $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ for all nodes u with cover-level $c\ell(u) \geq j$. Since all successors of intermediate nodes are non-intermediate, the case for intermediate nodes is a technicality that follows by definition, once we have showed the case for all non-intermediate nodes. For nodes u with a cover-level i strictly greater than j, i.e. $c\ell(u) = i > j$, this follows by induction in the cover-level, as we know that $\mathcal{A}_i^{sem}$ is the maximal post fixed-point assignment on cover-level $i > j$. This leaves us to consider nodes $u = \langle s, \nabla, \varphi \rangle$ with cover-level $c\ell(u) = j$.

If u is $\langle s, \nabla, \mathbf{true} \rangle$, $\langle s, \nabla, \mathbf{false} \rangle$ or $\langle s, \nabla, a \rangle$ then $\mathcal{A}_j^{sem}(u) = A_j^{max}(u)$ follows from the same arguments as in the base case. Thus, it must be the case that $\mathcal{A}_j^{sem}(u) = F_j(\mathcal{A}_j^{sem})(u)$, since u has no successors.

If $u = \langle s, \models, \varphi_1 \wedge \varphi_2 \rangle$ then we have two cases to consider (i) $s \models \varphi_1 \wedge \varphi_2$, hence, $\mathcal{A}_j^{sem}(u) = \infty$, and (ii) $s \not\models \varphi_1 \wedge \varphi_2$, hence, $\mathcal{A}_j^{sem}(u) = -\infty$.

If (i) is the case, then we observe from Figure 10(a) that u is a min-node with successors $v_1 = \langle s, \models, \varphi_1 \rangle$ and $v_2 = \langle s, \models, \varphi_2 \rangle$. Since $s \models \varphi_1 \wedge \varphi_2$ in case (i), it follows from the semantics that $s \models \varphi_1$ and $s \models \varphi_2$. Thus, it must be the case that $\mathcal{A}_j^{sem}(v_1) = \infty$ and $\mathcal{A}_j^{sem}(v_2) = \infty$, which clearly gives us $F_j(\mathcal{A}_j^{sem})(u) = \infty$. This proves that $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ for case (i).

If (ii) is the case, then clearly $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ holds since $F_j(\mathcal{A}_j^{sem})(u)$ cannot obtain a value less than $-\infty$.

If $u = \langle s, \not\models, \varphi_1 \wedge \varphi_2 \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ follows from arguments similar to those given for the previous case, with the observation that either $\mathcal{A}_j^{sem}(v_1) = \infty$ or $\mathcal{A}_j^{sem}(v_2) = \infty$ is enough for $F_j(\mathcal{A}_j^{sem})(u) = \infty$ to be the case.

If $u = \langle s, \nabla, \varphi_1 \vee \varphi_2 \rangle$ where ∇ is either $\models$ or $\not\models$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ follows from arguments similar to those given for nodes on the form $\langle s, \nabla, \varphi_1 \wedge \varphi_2 \rangle$.

If $u = \langle s, \models, EX_{\bowtie k}\ \varphi \rangle$ then we consider the following cases.

i) $s_i \models \varphi$ for some s_i where $s \xrightarrow{w_i} s_i$ and $w_i \bowtie k$

ii) $s_i \not\models \varphi$ for some s_i where $s \xrightarrow{w_i} s_i$ and $w_i \bowtie k$

Case (i): In this case we have from Figure 18(a) that $v_i = \langle s_i, \models, \varphi \rangle$ is a successor of u. Furthermore, it follows from $s_i \models \varphi$ that $\mathcal{A}_j^{sem}(v_i) = \infty$, which in turn shows that $F_j(\mathcal{A}_j^{sem})(u) = \infty$. Thus, clearly $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ holds.

Case (ii): In this case it is easy to see that $s \not\models EX_{\bowtie k}\ \varphi$, thus, it must be the case that $\mathcal{A}_j^{sem}(u) = -\infty$ by which $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ holds trivially.

If $u = \langle s, \not\models, EX_{\bowtie k}\ \varphi \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ follows by arguments similar to those given for the previous case.

If $u = \langle s, \nabla, AX_{\bowtie k}\ \varphi \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ follows from arguments similar to those given for nodes on the form $\langle s, \nabla, EX_{\bowtie k}\ \varphi \rangle$.

If $u = \langle s, \models, Q\ \varphi_1\ U_{<k}\ \varphi_2 \rangle$ then we observe that with u being a non-symbolic node we have $\mathcal{A}_j^{sem}(u)$ is either $-\infty$ or ∞. If $\mathcal{A}_j^{sem}(u) = -\infty$ then clearly $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$, as $F_j(\mathcal{A}_j^{sem})(u)$ cannot be less than $-\infty$.
If $\mathcal{A}_j^{sem}(u) = \infty$ then we have that $s \models Q\ \varphi_1\ U_{<k}\ \varphi_2$. From Figure 12(a) we have that $F_j(\mathcal{A}_j^{sem})(u) < \infty$ if and only if $\mathcal{A}_j^{sem}(\langle s, \not\models, Q\ \varphi_1\ U_{<?}\ \varphi_2 \rangle) \geq k$. However, as $s \models Q\ \varphi_1\ U_{<k}\ \varphi_2$ it must be the case that $\mathcal{A}_j^{sem}(\langle s, \not\models, Q\ \varphi_1\ U_{<?}\ \varphi_2 \rangle) < k$. Thus, $F_j(\mathcal{A}_j^{sem})(u) = \infty$, proving that $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ holds in this case.

If $u = \langle s, \not\models, Q\ \varphi_1\ U_{<k}\ \varphi_2 \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ follows from arguments similar to those given for the previous case.

If $u = \langle s, \nabla, Q\ \varphi_1\ W_{\geq k}\ \varphi_2 \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ follows from arguments similar to those given for nodes on the form $\langle s, \nabla, Q\ \varphi_1\ U_{<k}\ \varphi_2 \rangle$.

If $u = \langle s, \not\models, E\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$ then we recall from the semantics that $s \models E\ \varphi_1\ U_{<k}\ \varphi_2$ if and only if there exists a run σ and position $p \geq 0$ satisifying conditions 14, 15 and 16 for some $c < k$.

$$\sigma(p) \models \varphi_2 \tag{14}$$

$$\sigma(q) \models \varphi_1 \quad , \text{ for all } q < p \tag{15}$$

$$W_\sigma(p) \leq c \tag{16}$$

We now observe that $s \not\models E\ \varphi_1\ U_{<0}\ \varphi_2$ always holds, as there is no run σ and position p such that $W_p(\sigma) < 0$. Thus, we have that $\mathcal{A}_j^{sem}(u) \geq 0$, leaving us to consider the following cases.

i) $\mathcal{A}_j^{sem}(u) = \infty$, and

ii) $\mathcal{A}_j^{sem}(u) = k$ for some $k \in \mathbb{N}_0$

Before considering Cases (i) and (ii) we observe from Figure 13 that u is a min-node with successors $v = \langle s, \not\models, \varphi_2 \rangle$ and intermediate max-nodes combining $v' = \langle s, \not\models, \varphi_1 \rangle$ and $v_i = \langle s_i, \not\models, E\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$, where $s \xrightarrow{w_i} s_i$.

Case (i): If $\mathcal{A}_j^{sem}(u) = \infty$ then clearly $s \not\models \varphi_2$, leaving us to consider the two following cases.

1) $s \not\models \varphi_1$, or
2) $s_i \not\models E\ \varphi_1\ U_{<k}\ \varphi_2$ for any $s \to s_i$ and $k \in \mathbb{N}_0$.

Case (1): From $s \not\models \varphi_1$ we have that $\mathcal{A}_j^{sem}(v') = \infty$. It is now easy to see from Figure 13 that all the intermediate max-nodes after u become ∞. Furthermore, as $s \not\models \varphi_2$ we get that $\mathcal{A}_j^{sem}(v) = \infty$. And with the cover-edge from v returning ∞, whenever, $\mathcal{A}_j^{sem}(v) = \infty$, it is easy to see that $wgt_j(u, u', \mathcal{A}_j^{sem}) = \infty$ for all successors u' of u. Thus, it must be the case that $F_j(\mathcal{A}_j^{sem})(u) = \infty$.

Case (2): This case follows by the same arguments as Case (1), with the observation that intermediate max-nodes becomes ∞ because for all successors v_i we have that $\mathcal{A}_j^{sem}(v_i) = \infty$. Recall that $v_i = \langle s_i, \not\models, E\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$, where $s \xrightarrow{w_i} s_i$, thus, $\mathcal{A}_j^{sem}(v_i) = \infty$ follows from $s_i \not\models E\ \varphi_1\ U_{<k}\ \varphi_2$ for any $s \to s_i$ and $k \in \mathbb{N}_0$.

Case (ii): In this case we show that $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ when $\mathcal{A}_j^{sem}(u) = k$ for some $k \in \mathbb{N}_0$. We prove this by contradiction, assume that $F_j(\mathcal{A}_j^{sem})(u) < k$.

From the definition of F_j (Equation (3)) and Figure 13 it also follows that there are two cases which can give rise to $F_j(\mathcal{A}_j^{sem})(u) < k$.

1) $\mathcal{A}_j^{sem}(v) = -\infty$ giving us $F_j(\mathcal{A}_j^{sem})(u) = 0 < k$, or
2) $\mathcal{A}_j^{sem}(v') = -\infty$, for some $\mathcal{A}_j^{sem}(v_i) = k'$ and $F_j(\mathcal{A}_j^{sem})(u) = k' + w_i < k$.

If (1) is the case, then clearly $\sigma = s \ldots$ and position $p = 0$ is a run and position satisfying conditions 14, 15 and 16 for $c = 0$. Hence, $k = 0$ is the largest k for which $s \not\models E\ \varphi_1\ U_{<k}\ \varphi_2$. Thus, $F_j(\mathcal{A}_j^{sem})(u) = 0$ cannot be strictly smaller than k.

If (2) is the case, then clearly it follows from $\mathcal{A}_j^{sem}(v_i) = k'$ that k' is the largest k' for which $s_i \not\models E\ \varphi_1\ U_{<k'}\ \varphi_2$ holds. So we have that $s_i \models E\ \varphi_1\ U_{<k'+1}\ \varphi_2$ must hold. From this we have that there is a run σ and a position p satisfying conditions 14, 15 and 16 for $c < k' + 1$.

We now consider the extension $\sigma' = s \xrightarrow{w_i} s_i \ldots$ of σ and position $p' = p + 1$, and observe that σ' is a run and p' is a position satisfying conditions 14, 15 and 16 for $c < k' + 1 + w_i$.

- Condition 14 holds because $\sigma'(p') = \sigma(p)$ and $\sigma(p) \models \varphi_2$,
- Condition 15 holds as $\sigma(0) = s$, and $s \models \varphi_1$ follows from $\mathcal{A}_j^{sem}(v') = -\infty$, and for all $q < p$ we have that $\sigma'(q+1) = \sigma(q)$ and $\sigma(q) \models \varphi_1$.
- Condition 16 holds since $W'_\sigma(p') = W_\sigma(p) + w_i$ and $W_\sigma(p) < k' + 1$.

It now follows from the semantics that $s \models E\ \varphi_1\ U_{<x}\ \varphi_2$ holds for all $x > k' + w_i$. Thus, the largest y for which $s \not\models E\ \varphi_1\ U_{<y}\ \varphi_2$ holds must be $y \leq k' + w_i$. From the definition of the semantic

assignment we get that $\mathcal{A}_j^{sem}(u) \leq k' + w_i$, and consequently, $\mathcal{A}_j^{sem}(u) = k > k' + w_i$ cannot be the case.

If $u = \langle s, \not\models, A\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ can be shown with arguments similar to those given for nodes on the form $\langle s, \not\models, E\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$.

If $u = \langle s, \models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$ then we recall from the semantics that $s \models E\ \varphi_1\ W_{\geq k}\ \varphi_2$ if and only if there exists a run σ such that $\sigma(p) \models \varphi_1$ for all p or there is a position $p \geq 0$ where conditions 17, 18 and 19 holds for $k' \geq k$.

$$\sigma(p) \models \varphi_2 \tag{17}$$

$$\sigma(q) \models \varphi_1 \quad , \text{ for all } q < p \tag{18}$$

$$W_\sigma(p) \geq k' \tag{19}$$

We also observe from Figure 16 that u is a max-node with successors $v = \langle s, \models, \varphi_2 \rangle$ and intermediate min-nodes combining $v' = \langle s, \models, \varphi_1 \rangle$ and $v_i = \langle s_i, \models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$ for each s_i where $s \xrightarrow{w_i} s_i$.

We now consider the following cases for $\mathcal{A}_j^{sem}(u)$.

i) $\mathcal{A}_j^{sem}(u) = -\infty$,
ii) $\mathcal{A}_j^{sem}(u) = k$ for some $k \in \mathbb{N}_0$, and
iii) $\mathcal{A}_j^{sem}(u) = \infty$.

If (i) is the case then clearly $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ holds. This is because $F_j(\mathcal{A}_j^{sem})(u)$ cannot be less than $\mathcal{A}_j^{sem}(u) = -\infty$.

If (ii) is the case then $s \models E\ \varphi_1\ W_{\geq k}\ \varphi_2$ follows from $\mathcal{A}_j^{sem}(u) = k$. By the semantics we get that there is a run σ such that $\sigma \models \varphi_1\ W_{\geq k}\ \varphi_2$. We now consider the different cases by which $\sigma \models \varphi_1\ W_{\geq k}\ \varphi_2$ may hold.

1) $\sigma(p) \models \varphi_1$ for all positions p,
2) position $p = 0$ satisfies conditions 17, 18 and 19 for $k' \geq k$ and $k = 0$, and
3) some $p > 0$ satisfies conditions 17, 18 and 19 for $k' \geq k$.

If (1) is the case then clearly $s \models \varphi_1$ as $\sigma(0) = s$. From $s \models \varphi_1$ we get that $\mathcal{A}_j^{sem}(v') = \infty$. We now observe that σ may be written as $\sigma = s \xrightarrow{w} s_i \ldots$ for some s_i where $s \xrightarrow{w} s_i$. It then follows that the suffix $\sigma' = s_i \ldots$ of σ proves that $s_i \models \varphi_1\ W_{\geq k''}\ \varphi_2$ for any $k'' \in \mathbb{N}_0$. Thus, we have that $\mathcal{A}_j^{sem}(v_i) = \infty$ for the corresponding node v_i, combining this with $\mathcal{A}_j^{sem}(v') = \infty$ from previous and we get that $F_j(\mathcal{A}_j^{sem})(u) = \infty$. Hence, proving case (1).

If (2) is the case clearly $s \models \varphi_2$, which gives us that $\mathcal{A}_j^{sem}(v) = \infty$. It is then easily observed that $wgt_j(u, v, \mathcal{A}_j^{sem}) = 0$, hence, as u is a max-node, $F_j(\mathcal{A}_j^{sem})(u) \geq 0$. Thus, as $k = 0$ in this case, it follows that $F_j(\mathcal{A}_j^{sem})(u) \geq k$, which proves $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ for case (2).

If (3) is the case then $s \models \varphi_1$ as $k > 0$ and from this we get that $\mathcal{A}_j^{sem}(v') = \infty$. As in case (1) we observe that σ may be written as $\sigma = s \xrightarrow{w} s_i \ldots$ for some s_i where $s \xrightarrow{w} s_i$. It then follows that

the suffix $\sigma' = s_i \ldots$ of σ proves that $s_i \models \varphi_1 \; W_{\geq k-w_i} \; \varphi_2$. Thus, we have that $\mathcal{A}_j^{sem}(v_i) \geq k - w_i$ for the corresponding node v_i, combining this with $\mathcal{A}_j^{sem}(v') = \infty$ from previous and the fact that the edge from intermediate min-node to v_i has weight w_i, we get that $F_j(\mathcal{A}_j^{sem})(u) \geq k$.

Thus, we conclude that $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ for case (ii).

If (iii) is the case it follows from $\mathcal{A}_j^{sem}(u) = \infty$ that $s \models E \; \varphi_1 \; W_{\geq k''} \; \varphi_2$ holds for all $k'' \in \mathbb{N}_0$. By the semantics this is only possible if there is a run σ such that $\sigma(p) \models \varphi_1$ for all positions p. As before we now write σ as $\sigma = s \xrightarrow{w} s_i \ldots$ and observe that the suffix $\sigma' = s_i \ldots$ of σ proves that $\mathcal{A}_j^{sem}(v_i) = \infty$ for some v_i. In addition, we have $s \models \varphi_1$, as $\sigma(0) = s$ which gives us $\mathcal{A}_j^{sem}(v') = \infty$. Thus, even though v_i and v' are combined with an intermediate min-node we get $F_j(\mathcal{A}_j^{sem})(u) = \infty$ as $\mathcal{A}_j^{sem}(v_i) = \infty$ and $\mathcal{A}_j^{sem}(v') = \infty$.

If $u = \langle s, \models, A \; \varphi_1 \; W_{\geq ?} \; \varphi_2 \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ can be shown with arguments similar to those for nodes on the form $\langle s, \not\models, E \; \varphi_1 \; W_{\geq ?} \; \varphi_2 \rangle$.

If $u = \langle s, \models, \neg\varphi \rangle$ then we observe that u is a non-symbolic node, hence, $\mathcal{A}_j^{sem}(u)$ is either $-\infty$ or ∞. If $\mathcal{A}_j^{sem}(u) = -\infty$, then clearly $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ must hold.

If $\mathcal{A}_j^{sem}(u) = \infty$ then we have that $s \models \neg\varphi$ which implies that $s \not\models \varphi$. We then observe from Figure 11(a) that u is a min-node with the successor $v = \langle s, \not\models, \varphi \rangle$, which tells us that $F_j(\mathcal{A}_j^{sem})(u) = \mathcal{A}_j^{sem}(v)$. However, as $s \not\models \varphi$ we have that $\mathcal{A}_j^{sem}(v) = \infty$, thus, it follows that $F_j(\mathcal{A}_j^{sem})(u) = \infty$ which proves that $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ holds in this case.

If $u = \langle s, \not\models, \neg\varphi \rangle$ then $\mathcal{A}_j^{sem}(u) \leq F_j(\mathcal{A}_j^{sem})(u)$ follows from arguments similar to those given for the previous case.

Having shown (A) that $\mathcal{A}_j^{sem}$ is a post fixed-point assignment, we now show (B) that $\mathcal{A}_j^{sem}$ is also maximal, i.e. $A_j^{max} \sqsubseteq \mathcal{A}_j^{sem}$. This amounts to demonstrating that $A_j^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ for every node u. For nodes u with a cover-level $c\ell(u) = i$ strictly greater than j, i.e. $i > j$, we have that this follows by induction in the cover-level. For intermediate nodes v, $A_j^{max}(v) \leq \mathcal{A}_j^{sem}(v)$ is a technicality that follows trivially, once we have showed $A_j^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ for all non-intermediate nodes u. This leaves us to show $A_j^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ for nodes $u = \langle s, \nabla, \varphi \rangle$ with cover-level $c\ell(u) = j$. We do this by structural induction in φ as follows.

If φ **is true, false,** a**,** $\varphi_1 \vee \varphi_2$**,** $\varphi_1 \wedge \varphi_2$**,** $\neg\varphi'$ **or** $QX_{\bowtie} \; \varphi'$ then we have for nodes u on the form $u = \langle s, \nabla, \varphi \rangle$ that $A^{max}(v) = \mathcal{A}_j^{sem}(v)$ follows by arguments similar to those given in the base case with $j = m$. Thus, $A_j^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ must be the case for any such node u.

If $\varphi = E \; \varphi_1 \; U_{\leq k} \; \varphi_2$ then for any node u on the form $u = \langle s, \models, E \; \varphi_1 \; U_{\leq k} \; \varphi_2 \rangle$ we observe from Figure 12(a) that u has a cover-edge to $v = \langle s, \not\models, E \; \varphi_1 \; U_{<?} \; \varphi_2 \rangle$ such that $A_j^{max}(u) = \infty$ if $A_j^{max}(v) \leq k + 1$, and otherwise $A_j^{max}(u) = -\infty$.

Clearly, if $A_j^{max}(u) = \infty$, then $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ holds. If $A_j^{max}(u) = \infty$ then we have that $A_j^{max}(v) \leq k + 1$ which by induction in the cover-

level gives us that $\mathcal{A}_j^{sem}(v) \leq k+1$. From $\mathcal{A}_j^{sem}(v) \leq k+1$ we get that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$, hence, $\mathcal{A}_j^{sem}(u) = \infty$, proving the case.

For nodes u on the form $u = \langle s, \not\models, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$ we have from Figure 12(b) that output of the cover-edge is reversed, thus, $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ follows by similar arguments as before.

If $\varphi = A\ \varphi_1\ U_{\leq k}\ \varphi_2$ then for any node u on the form $u = \langle s, \nabla, A\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$ we have that $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ follows by arguments similar to those given for nodes with formulas on the form $E\ \varphi_1\ U_{\leq k}\ \varphi_2$.

If $\varphi = E\ \varphi_1\ U_{<?}\ \varphi_2$ then we show $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ by contradiction for nodes u on the form $u = \langle s, \not\models, E\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$.

Assume $\mathcal{A}_j^{sem}(u) < A^{max}(u)$, clearly this cannot be the case if $\mathcal{A}_j^{sem}(u) = \infty$, moreover, we recall that $\mathcal{A}_j^{sem}(u) \geq 0$ as $s \not\models E\ \varphi_1\ U_{<0}\ \varphi_2$ always holds. This leaves us to show that $\mathcal{A}_j^{sem}(u) < A^{max}(u)$ cannot be the case when $\mathcal{A}_j^{sem}(u) = k \in \mathbb{N}_0$.

We now recall that $\mathcal{A}_j^{sem}(u) = k$ is the largest k for which $s \not\models E\ \varphi_1\ U_{<k}\ \varphi_2$ holds. Thus, it follows that $s \models E\ \varphi_1\ U_{<k+1}\ \varphi_2$, from the semantics we have that this implies the existence of a run σ and a position p satisfying conditions 14, 15 and 16 for some $c < k+1$.

The existence of a run and position satisfying the conditions from the semantics, also implies the existence of a run σ' and minimal position ρ, s.t. ρ is the smallest position for which there exists a run σ' where σ and ρ satisfies conditions 14, 15 and 16 for some $c < k+1$.

We now proceed to show that $\mathcal{A}_j^{sem}(u) < A^{max}(u)$, where $\mathcal{A}_j^{sem}(u) = k \in \mathbb{N}_0$, cannot hold for any node u on the form $\langle s, \not\models, E\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$ by induction in the smallest position ρ for which there is a run σ satisfying conditions 14, 15 and 16 for some $c < k+1$.

Base Case ($\rho = 0$): In this case we have from condition 14 that $s \models \varphi_2$ as $\sigma(\rho) = s$. Clearly, this implies that $\mathcal{A}_j^{sem}(u) = 0$ as any run $\sigma' = s \ldots$ satisfies conditions 14, 15 and 16 for $c = 0$.

By induction in the cover-level we also have from $s \models \varphi_2$ that $A_j^{max}(v) = -\infty$ for $v = \langle s, \not\models, \varphi_2 \rangle$. We now see from Figure 13 that u is a min-node with cover-edge to v, such that $A_j^{max}(v) = -\infty$ makes $A^{max}(u) \leq 0$ as A^{max} is a post-fixed point assignment. Thus, $\mathcal{A}_j^{sem}(u) < A^{max}(u)$ cannot hold in this case.

Inductive Step ($\rho > 0$): In this case we may write σ as $\sigma = s \xrightarrow{w_i} s_i \ldots$ for some successor s_i. Since $\rho > 0$ we have from condition 15 that $s \models \varphi_1$, hence, it follows by structural induction that $\mathcal{A}_j^{sem}(v') = -\infty$ where $v' = \langle s, \not\models, \varphi_1 \rangle$. Furthermore, it is easy to see that the suffix $\sigma' = s_i \ldots$ of σ and position $\rho' = \rho - 1$ satisfies conditions 14, 15 and 16 for $c < k + 1 - w_i$. From the semantics it easy to see that $\mathcal{A}_j^{sem}(v_i) = k - w_i$, as $\mathcal{A}_j^{sem}(v_i) < k + w_i$ contradicts the existence of σ' and ρ', and $\mathcal{A}_j^{sem}(v_i) > k + w_i$ contradicts the fact that $\mathcal{A}_j^{sem}(u) = k$.

From Figure 13 that u is a min-node with an intermediate max-node combining v' and v_i. As $\rho' < \rho$ it follows by induction in the minimal position that $\mathcal{A}_j^{sem}(v_i) < A^{max}(v_i)$ cannot hold, hence, $A^{max}(v_i) \leq$

$\mathcal{A}_j^{sem}(v_i) = k + w_i$. As A^{max} is a post fixed-point assignment, it follows from Figure 13 that $A^{max}(u) \leq k$, as the intermediate max-node will be assigned k. Thus, $\mathcal{A}_j^{sem}(u) < A^{max}(u)$ cannot hold in this case either.

If $\varphi = A\ \varphi_1\ U_{<?}\ \varphi_2$ then $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$, for nodes u on the form $u = \langle s, \not\models, A\ \varphi_1\ U_{<?}\ \varphi_2 \rangle$, follows from similar arguments as those given for the previous case.

If $\varphi = Q\ \varphi_1\ W_{\geq k}\ \varphi_2$ then for any node u on the form $u = \langle s, \nabla, Q\ \varphi_1\ W_{\geq k}\ \varphi_2 \rangle$ we have that $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ follows by arguments similar to those given for nodes with formulas on the form $Q\ \varphi_1\ U_{\leq k}\ \varphi_2$.

If $\varphi = E\ \varphi_1\ W_{\geq ?}\ \varphi_2$ then we show $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$ for nodes $u = \langle s, \not\models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$ by contradiction. Assume $\mathcal{A}_j^{sem}(u) < A^{max}(u)$, clearly this cannot be the case if $\mathcal{A}_j^{sem}(u) = \infty$. This leaves us to show that $\mathcal{A}_j^{sem}(u) < A^{max}(u)$ cannot hold in the following cases.

i) $\mathcal{A}_j^{sem}(u) = -\infty$, and
ii) $\mathcal{A}_j^{sem}(u) = k$.

Before we show Cases (i) and (ii) we observe from Figure 16 that u is a max-node with successors $v = \langle s, \models, \varphi_2 \rangle$ and an intermediate min-node combining $v' = \langle s, \models, \varphi_1 \rangle$ and $v_i = \langle s_i, \models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$ for each s_i where $s \xrightarrow{w_i} s_i$.

Case (i): We observe from the semantics that $\mathcal{A}_j^{sem}(u) = \infty$ is the case if and only if there is a run $\sigma = s \ldots$ such that $\sigma(p) \models \varphi_1$ for every position p. Thus, we know from $\mathcal{A}_j^{sem}(u) = -\infty$ that there is no such run σ. This leads us to conclude that for any run $\sigma = s \ldots$ there must be a position p for which $\sigma(p) \not\models \varphi_1$ holds.

Further, it follows from $\mathcal{A}_j^{sem}(u) = -\infty$ that there is no $k \in \mathbb{N}_0$ for which $s \models E\ \varphi_1\ W_{\geq k}\ \varphi_2$ holds. From the semantics this implies that for any run σ there is no position p for which it holds that $\sigma(p) \models \varphi_2$ and $\sigma(p') \models \varphi_1$ for all $p' < p$.

Combining this with the previous observation, that for any $\sigma = s \ldots$ there must be a position p for which $\sigma(p) \not\models \varphi_1$, we get that for any run $\sigma = s \ldots$ there must be a position p for which $\sigma(p) \not\models \varphi_1$ and $\sigma(p') \not\models \varphi_2$ for all $p' \leq p$. This in turn, implies the existences of a smallest ρ such that for any run σ there is a position $p \leq \rho$ such that $\sigma(p) \not\models \varphi_1$ and $\sigma(p') \not\models \varphi_2$ for all $p' \leq p$.

We now show for any node u on the form $u = \langle s, \not\models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$ that $\mathcal{A}_j^{sem}(u) = -\infty$ implies $A_j^{max}(u) = -\infty$. We do this by induction in the smallest ρ for which any run $\sigma = s \ldots$, there is a position $p \leq \rho$ such that $\sigma(p) \not\models \varphi_1$ and $\sigma(p') \not\models \varphi_2$ for all $p' \leq p$.

Base Case ($\rho = 0$): In this case we have that $s \not\models \varphi_1$ and $s \not\models \varphi_2$ which by structural induction and induction in cover-level, respectively, implies that $\mathcal{A}_j^{sem}(v') = A_j^{max}(v') = -\infty$ and $\mathcal{A}_j^{sem}(v) = A_j^{max}(v) = -\infty$. From here it is easy to see that with A_j^{max} being a post fixed-point assignment, it must be the case that $A_j^{max}(u) = -\infty$.

Inductive Step ($\rho > 0$): Again, we have that $s \not\models \varphi_2$ implies $\mathcal{A}_j^{sem}(v) = A_j^{max}(v) = -\infty$ by induction in the cover-level. Furthermore, we have that $s \models \varphi_1$ as otherwise $\rho = 0$ would have been the smallest ρ for which any run $\sigma = s \ldots$ there is a position $p \leq \rho$ such that $\sigma(p) \not\models \varphi_1$ and $\sigma(p') \not\models \varphi_2$ for all $p' \leq p$.

Hence, it follows from $\mathcal{A}_j^{sem}(u) = -\infty$ that for all successors s_i of s, ie. $s \xrightarrow{w_i} s_i$, we have $s \not\models E\ \varphi_1\ U_{\geq k}\ \varphi_2$ for all $k \in \mathbb{N}_0$. This implies $\mathcal{A}_j^{sem}(v_i) = -\infty$, and that the ρ' which proves this must be strictly smaller than ρ. Thus, by induction in ρ' we get that $A_j^{max}(v_i) = \mathcal{A}_j^{sem}(v_i) = -\infty$.

Which with A_j^{max} being a post fixed-point assignment shows that $A_j^{max}(u) = -\infty$ must be the case.

Having shown that $\mathcal{A}_j^{sem}(u) = -\infty$ implies $A_j^{max}(u) = -\infty$ for any node u on the form $u = \langle s, \not\models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$, it is easy to see that $\mathcal{A}_j^{sem}(u) < A^{max}(u)$ cannot hold in case (i).

Case (ii): Again we have that that $\mathcal{A}_j^{sem}(u) < \infty$, thus, there is no run $\sigma = s \ldots$ such that $\sigma(p) \models \varphi_1$ holds for all positions p. Which, once more leads us to conclude that for any run $\sigma = s \ldots$ there must be a position p for which $\sigma(p) \not\models \varphi_1$ holds.

Furthermore, it follows from $\mathcal{A}_j^{sem}(u) = k$ that there is no $k' > k$ for which $s \models E\ \varphi_1\ W_{\geq k'}\ \varphi_2$ holds. From the semantics this implies that for any run σ there is no position p for which it holds that $W_\sigma(p) > k$, $\sigma(p) \models \varphi_2$ and $\sigma(p') \models \varphi_1$ for all $p' < p$.

Combining this with the previous observation, that for any $\sigma = s \ldots$ there must be a position p for which $\sigma(p) \not\models \varphi_1$, we get that for any run $\sigma = s \ldots$ there must be a position p such that $\sigma(p) \not\models \varphi_1$ and either $W_\sigma(p) < k$ or $\sigma(p') \not\models \varphi_2$ for all $p' \leq p$. This in turn, implies the existence of a smallest ρ such that for any run σ there is a position $p \leq \rho$ satisfying conditions (20) and either (21) or (22) for $k' = k$.

$$\sigma(p) \not\models \varphi_1 \tag{20}$$

$$\sigma(p') \not\models \varphi_2 \quad \text{for all } p' \leq p \tag{21}$$

$$W_\sigma(p) < k' \tag{22}$$

We now show for any node u on the form $u = \langle s, \not\models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$ that $\mathcal{A}_j^{sem}(u) = k$ implies $A_j^{max}(u) \leq k$ by induction in the smallest ρ for which any run $\sigma = s \ldots$ there is a position $p \leq \rho$ satisfying conditions (20) and either (21) or (22) for $k' = k$.

Base Case ($\rho = 0$): In this case we have that $s \not\models \varphi_2$ which implies $\mathcal{A}_j^{sem}(v') = -\infty$, and by structural induction we get that $A_j^{max}(v') = -\infty$. From this is follows that all the intermediate min-nodes after u get $-\infty$ in A_j^{max}. As u has a cover-edge from $v = \langle s, \models, \varphi_2 \rangle$ which does not allow $wgt_j(u, v, A_j^{max}) > 0$, and $A_j^{max}(u)$ cannot get a value higher than $-\infty$ from the interme-

diate min-nodes it can be observed that $A_j^{max}(u) \leq 0$, hence, proving the case.

Inductive Step ($\rho > 0$): We have that $s \models \varphi_1$ as otherwise $\rho = 0$ would have been the smallest ρ for which any run $\sigma = s \ldots$ there is a position $p \leq \rho$ satisfying conditions (20) and either (21) or (22) for $k' = k$.

We now observe that for all successors s_i of s, i.e. $s \xrightarrow{w_i} s_i$, we have any run $\sigma' = s_i \ldots$ is a suffix of some run $\sigma s \xrightarrow{w_i} s_i \ldots$. Hence, there must be a $\rho' < \rho$ such that for any run $\sigma' = s_i \ldots$ there is a position $p \leq \rho'$ satisfying conditions (20) and either (21) or (22) for $k' = k - w_i$.

From the existence of ρ' we have $\mathcal{A}_j^{sem}(v_i) \leq k - w_i$ for any node $v_i = \langle s_i, \models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$. As $\rho' < \rho$ it follows by induction in ρ' that $A^{max}(v_i) \leq \mathcal{A}_j^{sem}(v_i) \leq k - w_i$. Since A_j^{max} is a post fixed-point assignment, it is now easy to see from Figure 16 that $A_j^{max}(u) \leq \mathcal{A}_j^{sem}(v_i) + w_i$. Thus, $A_j^{max}(u) \leq k$ which proves this case.

Having showed that $\mathcal{A}_j^{sem}(u) = k$ implies $A_j^{max}(u) \leq k$ for any node u on the form $u = \langle s, \not\models, E\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$, it is easy to see that $\mathcal{A}_j^{sem}(u) < A^{max}(u)$ cannot holds in case (ii).

If $\varphi = A\ \varphi_1\ W_{\geq ?}\ \varphi_2$ then $A^{max}(u) \leq \mathcal{A}_j^{sem}(u)$, for nodes u on the form $u = \langle s, \not\models, A\ \varphi_1\ W_{\geq ?}\ \varphi_2 \rangle$, follows from similar arguments as those given for the previous case.

Thus, we have shown that $\mathcal{A}^{sem} = A^{max}$. □

Corollary 1 (Encoding Correctness). *Let $\mathcal{K}$ be a WKS, s be a state of $\mathcal{K}$, φ be a WCTL formula and G be an MMG expanded from $\langle s, \models, \varphi \rangle$ following Definition 5. Then $s \models \varphi$ if and only if $A^{max}(\langle s, \models, \varphi \rangle) = \infty$.*

Proof. Correctness follows trivially from Theorem 10. □

We note that for a WKS $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ and a formula φ, the size of the components of the constructed MMG $G = (V_{min}, V_{max}, E, T, c\ell)$ is $|V_{min} \cup V_{max}| = O(|S| \cdot |\varphi|)$ and $|E| = O(|\rightarrow| \cdot |\varphi|)$. Due to Theorem 7 and Corollary 1 we are able to state the following theorem, declaring that global model checking of full WCTL takes polynomial time.

Theorem 11 (Complexity of WCTL Model Checking). *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 $O(|S| \cdot |\rightarrow| \cdot |\varphi|^2)$.*

9 Weighted Calculus of Communicating Systems

In this section we present a high-level process algebra for WKS. The motivation for this is to have compact way of modelling weighted processes through the usage of parallel composition. This calculus is essentially a weighted variant of Milner's Calculus of Communicating Processes (CSS) [24].

Definition 7 (WCCS Process). *A WCCS processes over a set of actions $\alpha \in Act$, atomic propositions $x \in \mathcal{AP}$, and process names $M \in \mathcal{M}$, is given by the following grammar.*

$$P ::= M \mid \langle \alpha, w \rangle.P \mid P \parallel Q \mid P + Q \mid P \setminus L \mid P[f] \mid x : P \mid \mathbf{0}$$

where $w \in \mathbb{N}_0$ and $f : \underline{Act} \to Act \cup \mathcal{AP} \to \mathcal{AP}$ is a relabelling function satisfying $f(\tau) = \tau$ and $f(\bar{\alpha}) = \overline{f(\alpha)}$.

9.1 Semantics of WCCS

Let $\uplus$ denote the union operation over multisets. A WCCS process can be translated into a WKS using the semantics and the rules given in the following definition.

$$\text{Action } \frac{}{\langle \alpha, w \rangle.P \xrightarrow{\alpha, w} P}$$

$$\text{Choice } \frac{P \xrightarrow{\alpha, w} P'}{P + Q \xrightarrow{\alpha, w} P'} \qquad \text{Restrict } \frac{P \xrightarrow{\alpha, w} P' \quad \alpha, \bar{\alpha} \notin L}{P \setminus L \xrightarrow{\alpha, w} P' \setminus L}$$

$$\text{Par1 } \frac{P \xrightarrow{\alpha, w} P' \quad Q \xrightarrow{\bar{\alpha}, w'} Q'}{P \parallel Q \xrightarrow{\tau, w + w'} P' \parallel Q'} \qquad \text{Label } \frac{P \xrightarrow{\alpha, w} P'}{x : P \xrightarrow{\alpha, w} P'}$$

$$\text{Par2 } \frac{P \xrightarrow{\alpha, w} P'}{P \parallel Q \xrightarrow{\alpha, w} P' \parallel Q} \qquad \text{Rename } \frac{P \xrightarrow{\alpha, w} P'}{P[f] \xrightarrow{f(\alpha), w} P'[f]}$$

Fig. 22. Structural Operational Semantics of WCCS

Definition 8 (Translation). *Given a WCCS process P we translate it into a WKS $\mathcal{K} = (Proc, \mathcal{AP}, L, \to')$, where $(P, w, P') \in \to'$ if $P \xrightarrow{\alpha, w} P'$, for some $\alpha \in Act$, using rules in Figure 22, and L is defined as follows.*

$$\begin{aligned} L(x : P) &= \{x\} & L(P[f]) &= \{f(x) \mid x \in L(P)\} \\ L(P \setminus L) &= L(P) & L(M) &= L(P), \text{ where } M = P \\ L(P \parallel Q) &= L(P) \uplus L(Q) & L(\mathbf{0}) &= \emptyset \\ L(P + Q) &= L(P) \uplus L(Q) & L(\langle \alpha, w \rangle.P) &= \emptyset \end{aligned}$$

Remark 5. Note that the labelling function of atomic propositions uses multisets in the translation. This enables us to count and compare the number of atomic propositions using arithmetic expressions in the logic.

Example 6 (Translation of WCCS Process).

The following WCCS definition models a ring-based leader election protocol consisting of three processes in the network. For convenience we write $\langle\overline{m_{i\to j}}\rangle$ to denote that a message is sent to process P_i with rank j. A process changes its state whenever receives a rank higher than observed so far. If a process receives its own rank, it can declare itself the winner of the election. The highest ranking process eventually becomes the leader.

$$P_1 := \langle\overline{m_{3\to 1}, 1}\rangle.P_1 + \langle m_{1\to 1}\rangle.\mathit{leader} : \mathbf{0} + \langle m_{1\to 2}\rangle.P_{1,2} + \langle m_{1\to 3}\rangle.P_{1,3}$$
$$P_{1,2} := \langle\overline{m_{3\to 2}, 1}\rangle.P_{1,2} + \langle m_{1\to 1}\rangle.\mathit{leader} : \mathbf{0} + \langle m_{1\to 2}\rangle.P_{1,2} + \langle m_{1\to 3}\rangle.P_{1,3}$$
$$P_{1,3} := \langle\overline{m_{3\to 3}, 1}\rangle.P_{1,3} + \langle m_{1\to 1}\rangle.\mathit{leader} : \mathbf{0} + \langle m_{1\to 2}\rangle.P_{1,3} + \langle m_{1\to 3}\rangle.P_{1,3}$$

$$P_2 := \langle\overline{m_{1\to 2}, 1}\rangle.P_2 + \langle m_{2\to 1}\rangle.P_2 + \langle m_{2\to 2}\rangle.\mathit{leader} : \mathbf{0} + \langle m_{2\to 3}\rangle.P_{2,3}$$
$$P_{2,3} := \langle\overline{m_{1\to 3}, 1}\rangle.P_{2,3} + \langle m_{2\to 2}\rangle.\mathit{leader} : \mathbf{0} + \langle m_{2\to 1}\rangle.P_{2,3} + \langle m_{2\to 3}\rangle.P_{2,3}$$

$$P_3 := \langle\overline{m_{2\to 3}, 1}\rangle.P_3 + \langle m_{3\to 1}\rangle.P_3 + \langle m_{3\to 2}\rangle.P_3 + \langle m_{3\to 3}\rangle.\mathit{leader} : \mathbf{0}$$

$$\mathit{Ring} := (P_1 \parallel P_2 \parallel P_3)\backslash\{m_{i\to j} \mid 1 \le i, j \le 3\}$$

The model is then translated to the WKS shown in Figure 23. State s_0 corresponds to the process named *Ring*. Whenever a process sends a message there is a cost of one associated it. State s_5 is the final state, where P_3 wins the election and becomes the leader.

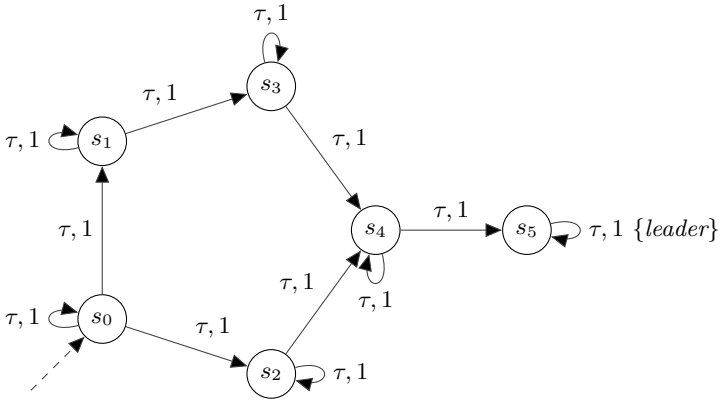


Fig. 23. A WKS for the leader election example with 3 processes

10 Experiments

In order to evaluate the performance of the three different approaches to WCTL model checking described in this paper, we have implemented a tool. A web-based front-end written in CoffeeScript (which compiles to JavaScript) is available at

`http://wktool.jonasfj.dk`

The tool permits the definition of systems using WCCS, the extended CCS-like [23] syntax presented in Section 9. A screenshot of the tool is shown in Figure 24. This allows us to easily define scalable models for use in the benchmarks. We carry out experiments on the following well-known models.

- Leader Election [10],
- Alternating Bit Protocol [5], and
- Task Graph Scheduling problems for two processors [15].

In the first two models the weight represents cost associated with the transmission of messages, while the weights in the task scheduling models represents the clock ticks of the processors.

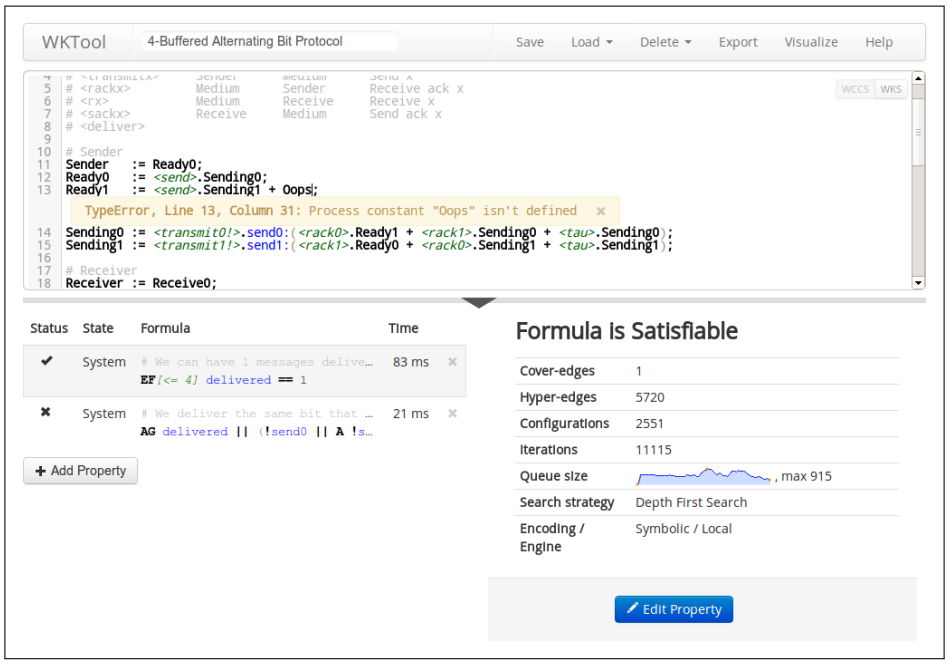


Fig. 24. A screenshot of WKTool as hosted on `http://wktool.jonasfj.dk`.

Experiments were carried out on laptop (Intel Core i7) running Ubuntu 12.04. Node.js (v. ≥ 0.8), an efficient JavaScript runtime environment, was used to run

the program from a command-line interface. The heap of the runtime environments garbage collector was limited to 1 GB memory, aborting verification if more memory was needed.

The experiments are structured as follows. First we provide results for a comparison between the three different encodings presented in this paper. That is, we analyze the performance of DG (dependency graph), SDG (symbolic dependency graph) and MMG (min-max graph) encodings as we scale the bound of a formula. The, we compare the local and global SDG algorithms. Afterwards, the performance of the local and global MMG algorithms is studied. For these experiments we will only be using $\text{WCTL}_{\leq}$ formulas so that we are able to compare the results. Finally, we evaluate the local and global algorithms for MMGs using the full WCTL logic, to determine if local search still provides a speed-up for mixed-modality formulas.

10.1 DG, SDG and MMG Compared

Tables 5 and 6 show the results for a comparison between direct (DG), symbolic and min-max algorithms. Execution times are in seconds and entries with "OOM" state that verification runs out of memory. For both problems the models are set to a fixed size, while the bound k in the formulas is scaled. For the leader election protocol fixed to eight processes two formulas are used. The first formula is satisfiable and asks whether a leader can be elected within k message exchanges, while the second formula is unsatisfiable, asking if there can be more than one leader within k message exchanges.

Leader Election w. 8 Processes							
	Direct		Symbolic		Min-Max		
k	Global	Local	Global	Local	Global	Local	
200	3.90	0.23	0.26	0.02	0.31	0.02	Satisifed
400	8.53	0.24	0.26	0.02	0.31	0.02	
600	OOM	0.25	0.26	0.02	0.31	0.02	
800	OOM	0.25	0.26	0.02	0.31	0.02	
1000	OOM	0.26	0.26	0.02	0.31	0.02	
200	7.64	8.11	0.25	0.25	0.27	0.30	Unsatisfied
400	16.87	20.03	0.26	0.26	0.27	0.30	
600	OOM	OOM	0.26	0.26	0.27	0.30	
800	OOM	OOM	0.26	0.26	0.27	0.30	
1000	OOM	OOM	0.25	0.26	0.27	0.30	

Satisfied: $EF_{\leq k}$ leader
Unsatisfied: $EF_{\leq k}$ leader > 1

Table 5. Scaling of bounds in formula for leader election (time in seconds)

For the alternating bit protocol with a four-entry communciation buffer, a satisfiable and an unsatisfiable formula is used. The first formula asks if a message can be delivered within k communication steps and the second formula asks if it is possible for the sender and receiver to get out of sync within k communciation steps.

Alternating Bit Protocol w. Buffer Size 4							
	Direct		Symbolic		Min-Max		
k	Global	Local	Global	Local	Global	Local	
100	3.96	0.05	0.23	0.03	0.26	0.02	Satisfied
200	8.48	0.06	0.23	0.03	0.26	0.02	
300	OOM	0.08	0.21	0.03	0.26	0.02	
400	OOM	0.11	0.23	0.03	0.26	0.02	
500	OOM	0.13	0.23	0.03	0.26	0.02	
100	3.72	4.00	0.27	0.23	0.28	0.31	Unsatisfied
200	7.51	10.07	0.27	0.23	0.28	0.30	
300	OOM	16.26	0.26	0.23	0.27	0.31	
400	OOM	OOM	0.28	0.22	0.28	0.31	
500	OOM	OOM	0.27	0.23	0.28	0.31	

Satisfied: $EF_{\leq k}$ delivered $= 1$
Unsatisfied: $EF_{\leq k}$ $(\text{send}_0 \wedge \text{deliver}_1) \vee (\text{send}_1 \wedge \text{deliver}_0)$

Table 6. Scaling of bounds in formula for alternating bit protocol (time in seconds)

For the satisfiable formulas, the global direct algorithm quickly runs out of memory as the bound in the formula is increased. It is apparent that the local direct algorithm (using DFS) is advantageous over the global direct algorithm for the satisfiable formulas, as it performs similarly to the global symbolic and min-max algorithms. Still, the global direct algorithm runs out of memory for the unsatisfiable formulas when the bound exceeds a certain threshold. It is clear that the local symbolic and min-max algorithms are the best performing. Furthermore, their execution times are closely correlated as the bound k is scaled.

Similar observations have been made for other models during testing, where the symbolic and min-max encodings repeatedly outperformed the direct encoding. For this reasion, we shall direct our attention to just the local/global symbolic and min-max algorithms for the remaining experiments.

10.2 Local vs. Global Model Checking on SDG

Now we examine the performance of the local and global SDG algorithms in more detail. Once again, we use the leader election and alternating bit protocol models, shown in Table 7. Only this time, the instances are scaled, i.e. number of processes and buffer size, respectively, instead of the bound in the formula. The formulas are the same as in the previous experiments.

Leader Election			
	$k = 200$		
n	Global	Local	
7	0.08	0.01	Satisifed
8	0.30	0.02	
9	1.12	0.02	
10	5.14	0.03	
11	23.07	0.03	
12	OOM	0.04	
7	0.07	0.07	Unsatisfied
8	0.26	0.26	
9	1.01	1.02	
10	4.94	4.93	
11	24.67	22.81	
12	OOM	OOM	

Alternating Bit Protocol							
	$k = 10$		$k = 20$		$k = \infty$		
n	Global	Local	Global	Local	Global	Local	
5	0.34	0.10	0.34	0.07	0.34	0.04	Satisifed
6	0.76	0.19	0.77	0.10	0.77	0.05	
7	1.92	0.34	1.93	0.14	1.83	0.05	
8	4.55	0.78	4.59	0.71	4.65	0.09	
9	13.24	9.99	12.39	1.71	12.35	0.18	
10	OOM	OOM	OOM	5.72	OOM	0.23	
4	0.26	0.22	0.28	0.24	0.32	0.28	Unsatisfied
5	0.51	0.40	0.54	0.43	0.65	0.47	
6	1.21	0.91	1.29	0.97	1.42	1.09	
7	2.98	2.42	3.11	2.31	3.17	2.81	
8	6.41	4.79	6.68	4.97	7.55	5.52	
9	OOM	OOM	OOM	OOM	OOM	OOM	

Table 7. Scaling the model size to compare global and local algorithms for symbolic dependency graphs

For the leader election protocol we observe that the local algorithm is significantly faster than the global algorithm for satisfiable formulas. Though for the unsatisfiable formulas the execution times are roughly the same.

For the alternating bit protocol the bound k in the formula is also scaled to 10, 20 and ∞. The execution times for the unsatisfiable formulas are more or less the same regardless of the bound. On the other hand, for the satisfiable formula the bound $k = 10$ is narrow in the sense that there are only a few runs that satisfy the property. When the bound is increased, the solution space becomes larger, and the local algorithm is therefore able to find a witness much faster. This is particularly apparent for $k = \infty$, as there is essentially no constraint on the accumulated weight of a candidate run s.t. a message is finally delivered.

The SDG algorithms were also tested on a set of task graph scheduling problems [4]. Given a collection of parallel tasks, the multiprocessor scheduling problem asks, whether there is a non-preemptive schedule such that all tasks can complete given their constraints and processing times on a fixed number of homogenous processors [15].

The benchmark consists of 180 automatically generated models from the standard task set for two processors. Each task graph is scaled by the number of parallel tasks n included in the schedulability analysis. We present results for the first three task graphs in Table 8. Two nested formulas are used for this test; one satisfiable and the other unsatisfiable. The satisfiable formula asks if within 90

	T0		T1		T2		
n	Global	Local	Global	Local	Global	Local	
3	3.14	0.01	0.15	0.08	0.19	0.01	Satisifed
4	4.70	1.11	0.16	0.08	0.88	0.19	
5	6.05	0.03	2.62	0.01	6.96	0.02	
6	OOM	OOM	5.18	0.92	OOM	1.32	
7	OOM	0.02	OOM	0.01	OOM	0.01	
8	OOM	0.03	OOM	OOM	OOM	2.49	
9	OOM	OOM	OOM	OOM	OOM	1.80	
10	OOM	0.03	OOM	OOM	OOM	OOM	
2	0.22	0.19	0.05	0.05	0.07	0.01	Unsatisfied
3	2.89	2.54	0.14	0.13	0.20	0.01	
4	5.50	2.16	0.15	0.14	0.90	0.19	
5	7.37	4.97	2.26	1.69	7.10	0.02	
6	OOM	OOM	4.64	4.10	OOM	1.34	
7	OOM	OOM	OOM	OOM	OOM	8.04	

Satisfied: $EF_{\leq 90}\ (t_{n-2}^{\text{ready}} \wedge AF_{\leq 80}\ \text{done} = n)$
Unsatisfied: $EF_{\leq 10}\ (t_{n-2}^{\text{ready}} \wedge AF_{\leq 5}\ \text{done} = n)$

Table 8. Scaling task graphs by number of tasks to compare global and local algorithm for symbolic dependency graphs

clock ticks, task t_{n-2} can be released and the entire schedule always terminates within 80 clock ticks. The formula becomes unsatisfiable for all instances when the bounds are reduced to 10 and 5, respectively. Once more, we see that local outperforms global for the positive formulas, in terms of execution time and the problem size for which available memory is exhausted.

10.3 Local vs. Global Model Checking on MMG

In this section we repeat the experiments from the previous section, but now for the local and global MMG algorithms.

Results for the leader election and alternating bit protocol problems are listed in Table 10. Again we see that the local and global algorithms exhibit the same characteristics as for the SDG algorithms. Namely, local clearly outperforms global for positive instances. While both algorithms take about as long to verify the negative instances.

The execution times for the three different task graphs are shown in Table 9. For the task graphs the local and global MMG algorithms scale approximately to the same problem size as for SDG before running out of memory.

We now provide a more comprehensive benchmark to compare the SDG and MMG algorithms. Towards this end we tested the SDG and MMG algorithms on all of the 180 task graphs for values $k = 30$, 60 and 90 on the formula $EF_{\leq k}\ \text{done} = n$, as shown in Table 11. The asks if the entire task graph can be scheduled within k clock ticks. The number of complete verification tasks was measured (i.e. those that did not run out of memory), including the total accumulated time it took to verify the cases where both the local and global algorithm were able to finish.

	T0		T1		T2		
n	Global	Local	Global	Local	Global	Local	
2	0.35	0.17	0.08	0.05	0.09	0.05	Satisifed
3	5.70	2.30	0.19	0.11	0.26	0.13	
4	11.02	4.17	0.21	0.12	1.51	0.61	
5	15.15	4.39	6.25	2.01	15.42	4.54	
6	OOM	OOM	16.84	3.48	OOM	10.79	
7	OOM	OOM	OOM	OOM	OOM	OOM	
2	0.34	0.29	0.07	0.06	0.09	0.06	Unsatisfied
3	5.48	4.04	0.18	0.16	0.26	0.13	
4	13.33	7.59	0.26	0.18	1.56	0.75	
5	17.65	8.12	5.12	3.63	16.30	5.71	
6	OOM	OOM	14.58	5.82	OOM	14.04	
7	OOM	OOM	OOM	OOM	OOM	OOM	

Satisfied: $EF_{\leq 90}\ (t_{n-2}^{\text{ready}} \wedge AF_{\leq 80}\ \text{done} = n)$
Unsatisfied: $EF_{\leq 10}\ (t_{n-2}^{\text{ready}} \wedge AF_{\leq 5}\ \text{done} = n)$

Table 9. Scaling task graphs by number of tasks to compare global and local algorithms for min-max graphs

Leader Election			
	$k = 200$		
n	Global	Local	
7	0.10	0.01	Satisfied
8	0.32	0.02	
9	1.39	0.03	
10	6.38	0.04	
11	31.03	0.04	
12	OOM	0.05	
7	0.08	0.12	Unsatisfied
8	0.28	0.31	
9	1.22	1.37	
10	5.98	6.40	
11	27.72	33.53	
12	OOM	OOM	

Alternating Bit Protocol							
	$k = 10$		$k = 20$		$k = \infty$		
n	Global	Local	Global	Local	Global	Local	
5	0.67	0.03	0.54	0.03	0.54	0.02	Satisfied
6	1.53	0.02	1.30	0.02	1.31	0.02	
7	3.65	0.03	3.15	0.03	3.01	0.03	
8	8.64	0.04	7.43	0.04	7.24	0.04	
9	OOM	OOM	OOM	0.04	OOM	0.04	
10	OOM	OOM	OOM	0.05	OOM	0.05	
4	0.27	0.31	0.28	0.31	0.28	0.30	Unsatisfied
5	0.54	0.57	0.55	0.57	0.54	0.57	
6	1.35	1.26	1.35	1.27	1.36	1.27	
7	3.05	3.58	2.92	3.54	2.93	3.64	
8	7.51	7.43	7.37	7.41	7.47	7.40	
9	OOM	OOM	OOM	OOM	OOM	OOM	

Table 10. Scaling the model size to compare global and local algorithms for min-max graphs

The results from this table support the claim that the local algorithm benefits from situations where the number of solutions (i.e. schedules) grows due to a relaxation in the bound k. Furthermore, we see that the MMG algorithms are slightly slower than those for SDG. However, this only looks to be a constant factor slowdown, ≈ 2 for the global algorithm and far less for the local algorithm. A plausible explanation for this is that the MMG encoding needs roughly twice as many configurations to encode the formulas due to the intermediate nodes, which also have a memory footprint.

	Symbolic					
180 task graphs for	$k = 30$		$k = 60$		$k = 90$	
Algorithm	Global	Local	Global	Local	Global	Local
Number of finished tasks	32	85	32	158	32	178
Accumulated time (seconds)	50.9	13.2	45.9	2.3	45.5	0.45
	Min-Max					
180 task graphs for	$k = 30$		$k = 60$		$k = 90$	
Algorithm	Global	Local	Global	Local	Global	Local
Number of finished tasks	32	83	32	158	32	178
Accumulated time (seconds)	119	20	115	3.4	115	0.67

Table 11. Summary of task graph benchmarks (180 cases in total)

In conclusion of the experiments thus far, we argue that the MMG algorithms can handle the models just as well as the SDG algorithms. So, despite the fact that we are dealing with two different encodings, local verification is still advantageous for MMGs and the extra overhead of the MMG encoding is only by a constant factor. With that being said, it is important to establish whether the local MMG algorithm is still benefitial for mixed-modality formulas. In the section to follow we test the MMG algorithms on the full WCTL logic in order to evaulate their performance on such formulas.

10.4 Full WCTL Local vs. Global on MMG

To evaluate the performance of local vs. global for mixed-modality formulas, we once again ran experiments on the three task graph models (T0,T1 and T2). The problem size is scaled as before and both a positive and negative nested formula is used.

The positive one asks if it is possible for task t_{n-2} to be released within 10 clock ticks and at the same time the entire schedule cannot always complete within 5 ticks. The negative formula asks if it is always possible for task t_{n-2} to be released within 10 ticks and there is also no way for a schedule to complete within 5 ticks. Table 12 lists the results. Note the usage of negation in the formulas, so they are no longer simple reachability properties.

	T0		T1		T2		
n	Global	Local	Global	Local	Global	Local	
2	0.41	0.16	0.08	0.05	0.10	0.06	Satisifed
3	6.19	2.39	0.21	0.11	0.28	0.13	
4	9.91	4.24	0.27	0.12	1.84	0.62	
5	16.09	5.01	6.90	2.99	15.20	6.33	
6	OOM	OOM	14.04	7.72	OOM	8.22	
7	OOM	OOM	OOM	OOM	OOM	OOM	
2	0.40	0.23	0.08	0.05	0.09	0.07	Unsatisfied
3	8.38	3.91	0.25	0.14	0.25	0.17	
4	8.37	6.03	0.32	0.17	1.36	0.87	
5	11.56	5.55	7.89	3.37	11.24	8.87	
6	OOM	OOM	14.84	6.02	OOM	OOM	
7	OOM	OOM	OOM	OOM	OOM	OOM	

Satisfied: $EF_{\leq 10}\ (t_{n-2}^{\text{ready}} \wedge \neg AF_{\leq 5}\ \text{done} = n)$
Unsatisfied: $AF_{\leq 10}\ (t_{n-2}^{\text{ready}} \wedge \neg EF_{\leq 5}\ \text{done} = n)$

Table 12. Scaling task graphs by number of tasks to compare global and local algorithms for min-max graphs

The local algorithm is roughly twice as fast as the global algorithm in most of the positive and negative cases. Despite it beeing the case that the formulas cause an extensive exploration, the local algorithm is able to exploit the structure of the underlying MMG. The local algorithm performs a partial exploration of the successors of max-nodes. A max node is only put back into the waiting list if the node with the largest assignment is updated. Hence, nested formulas with mixed modalities may still benefit from the local algorithm.

11 Conclusion

We have described three different approaches to WCTL model checking and discussed their limitations. Our previous efforts for the verification of the fragment $\text{WCTL}_{\leq}$, i.e. negation-free and restricted to upper-bound constraints, has been covered in the form of a reduction to fixed-point computation on dependency graphs and the proposed symbolic extension. These techniques permit local model checking, but are limited to alternation-free fixed-points, and thus in their expressive power with respect to the logic.

We expanded on these ideas and proposed min-max graphs as a technique to verify the full weighted CTL logic. Opposite to these techniques for WCTL model checking, min-max graphs support alternating fixed-points and hence a more expressive logic, e.g. invariant properties and the definition of weak until with lower-bound constraints. We described a global and local algorithm for the computation of fixed-points in order to solve model checking problems for the logic.

The algorithms were implemented in an online tool that makes WCTL model checking on WCCS models readily available. Through experiments with our implementation, we have demonstrated the advantages of local model checking for all three approaches. The principal conclusion is that the local approach can provide an order of magnitude speed-up when the bounds and logical formula permit a large number of possible witnesses of the satisfiability of the property.

11.1 Future Work

For future work it is of interest to study whether it is possible to adapt the technique to permit lower-bound constraints on the until modality. We believe that the use of the Floyd-Warshall all-pairs shortest path algoritmh for $\{-1, 0, 1\}$-weighted structures, presented in [18], can be extended to facilitate $\mathbb{Z}$-weighted structures. The approach is, however, inherently global, yet it would be worthwhile to further investigate this idea.

Both the local algorithm for symbolic dependency graphs and the local algorithm for min-max graphs presented here, rely on unspecified heuristics. For our experiments a depth-first search was used. Still, it would be interesting to study more elaborate heuristics. Our preliminary findings on this aspect suggest that the heuristic choices have a significant impact on performance.

Another interesting topic of future work would be the investigation of partial min-node exploration. The local algorithm for min-max graphs presented here always explores every successor of a min-node. This could potentially be avoided by back-propagating a special value for when the fixed-point assignment of a node is believed to be ∞, such that another successor of a min-node may be chosen for exploration. Obviously, this would be another heuristic, though this could facilitate further exploitation of the min-max graph structure.

In this work we present min-max graphs as a framework for solving weighted model checking problems. The reduction of other problems, e.g. games, to min-

max graphs is certainly also interesting. Along the same lines we can explore further abstraction and generalization of this approach to local model checking.

12 Bibliographical Remarks

The results presented in this paper are a continuation of our pre-specialization project [14]. For completeness, some definitions and results, that were also presented in [14], have been included in Sections 2, 3 and 4, though these are not excerpts. Proof of Theorem 1, as attached in appendix A.1, is an excerpt from [14] with minor changes. Proof of Theorem 4, presented in appendix A.2, was not a part of [14], but was also submitted as an appendix to the initial draft of [13]. The experiments presented in Section 10 were not part of [14], but the models and formulas were used for the experiments in [13].

References

1. Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.
2. Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In Benedetto and Sangiovanni-Vincentelli [7], pages 49–62.
3. Henrik Reif Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3 – 30, 1994.
4. Kasahara Laboratory at Waseda University. Standard task graph set. http://www.kasahara.elec.waseda.ac.jp/schedule/.
5. K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.
6. Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In Benedetto and Sangiovanni-Vincentelli [7], pages 147–161.
7. Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *Lecture Notes in Computer Science*. Springer, 2001.
8. Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. Model-checking for weighted timed automata. In Yassine Lakhnech and Sergio Yovine, editors, *FORMATS/FTRTFT*, volume 3253 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2004.
9. Peter Buchholz and Peter Kemper. Model checking for a class of weighted automata. *Discrete Event Dynamic Systems*, 20:103–137, 2010.
10. E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. of ACM*, 22(5):281–283, 1979.
11. Edmund M Clarke and E Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*. Springer, 1982.

12. Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.
13. Jonas Finnemann Jensen, Kim Guldstrand Larsen, Jiří Srba, and Lars Kaerlund Oestergaard. Local model checking of weighted ctl with upper-bound constraints. In Ezio Bartocci and C.R. Ramakrishnan, editors, *Model Checking Software*, volume 7976 of *Lecture Notes in Computer Science*, pages 178–195. Springer Berlin Heidelberg, 2013.
14. Jonas Finnemann Jensen and Lars Kærlund Østergaard. Local model checking of weighted ctl. http://www.lets.dk/downloads/sdg.pdf, 2012.
15. Y.-K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381 – 422, 1999.
16. F. Laroussinie, Ph. Schnoebelen, and M. Turuani. On the expressivity and complexity of quantitative branching-time temporal logics, 2001.
17. François Laroussinie, Nicolas Markey, and Ghassan Oreiby. Model-checking timed atl for durational concurrent game structures. In Eugene Asarin and Patricia Bouyer, editors, *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2006.
18. François Laroussinie, Antoine Meyer, and Eudes Petonnet. Counting ctl. *Logical Methods in Computer Science*, 9(1), 2012.
19. Kim G. Larsen, Uli Fahrenberg, and Claus R. Thrane. A quantitative characterization of weighted kripke structures in temporal logic. In *MEMICS*, 2009.
20. Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell, 1997.
21. Xinxin Liu, C.R. Ramakrishnan, and ScottA. Smolka. Fully local and efficient evaluation of alternating fixed points. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 5–19. Springer Berlin Heidelberg, 1998.
22. Xinxin Liu and Scott A. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In *ICALP*, pages 53–66, 1998.
23. R. Milner. A calculus of communicating systems. *LNCS*, 92, 1980.
24. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
25. Sergio Yovine. Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.

A Appendix

A.1 Proofs Related to Dependency Graph

This section is a slightly modified excerpt from our pre-specialization project [14].

Proof of Theorem 1

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

Proof. We prove Theorem 1 by structural induction on φ.

(I) For $\varphi = \mathbf{true}$ we show that for all $s \in S$ we have $A^{min}(\langle s, \mathbf{true} \rangle) = 1$ if and only if $s \models \mathbf{true}$. But as $s \models \mathbf{true}$ always holds, it is sufficient to show that $A^{min}(\langle s, \mathbf{true} \rangle) = 1$ for any pre fixed-point assignment A of G. In Figure 2(a) we add a hyper-edge from the configuration $\langle s, \mathbf{true} \rangle$, to the empty target set. Thus, we have that $A(v) = 1$ for any pre fixed-point assignment A of G, because all vertices in the empty set satisfy any property vacuously.

(II) For $\varphi = a$ we prove that $A^{min}(\langle s, a \rangle) = 1$ 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 2(b), there is a hyper-edge from the configuration $\langle s, a \rangle$ to the empty target set. As in (I) this means that $A^{min}(\langle s, a \rangle) = 1$, 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 2(b), we can conclude that there is no hyper-edge from the configuration $\langle s, a \rangle$ when $a \notin L(s)$. Thus, we have $A^{min}(\langle s, a \rangle) = 0$ because A^{min} is the minimum pre fixed-point assignment.

(III) For $\varphi = \varphi_1 \wedge \varphi_2$ we show that $A^{min}(\langle s, \varphi_1 \wedge \varphi_2 \rangle) = 1$ if and only if $s \models \varphi_1 \wedge \varphi_2$ for all $s \in S$. By Figure 2(c), 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 $A^{min}(\langle s, \varphi_1 \wedge \varphi_2 \rangle) = 1$ if and only if $A^{min}(\langle s, \varphi_1 \rangle) = 1$ and $A^{min}(\langle s, \varphi_2 \rangle) = 1$. By the induction hypothesis this is equivalent to $s \models \varphi_1$ 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 $A^{min}(\langle s, \varphi_1 \vee \varphi_2 \rangle) = 1$ if and only if $s \models \varphi_1 \vee \varphi_2$ for all $s \in S$. By Figure 2(d), 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 $A^{min}(\langle s, \varphi_1 \vee \varphi_2 \rangle) = 1$ if and only if $A^{min}(\langle s, \varphi_1 \rangle) = 1$ or $A^{min}(\langle s, \varphi_2 \rangle) = 1$. 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 $A^{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 1$ if and only if $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ 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.

$$\sigma(p) \models \varphi_2 \tag{23}$$

$$\sigma(j) \models \varphi_1 \text{ , for all } j < p \tag{24}$$

$$W_\sigma(p) \leq k' \tag{25}$$

$\Rightarrow$: Assume that $A^{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$, 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 the value 1, as $Z(v)$, formally we write the auxiliary function Z as follows.

$$Z(v) = \begin{cases} i & \text{if } F^i(A_0)(v) \neq F^{i-1}(A_0)(v) \\ \infty & \text{otherwise} \end{cases} \tag{26}$$

For any configuration v it holds that $Z(v) < \infty$ if and only if $A^{min}(v) = 1$, as a pre 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 $A^{min}(v) = 1$, 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. In Figure 2(e) we observe 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 $A^{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ implies the existence of a run σ and a position p satisifying conditions 23, 24 and 25 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 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 $A^{min}(\langle s, \varphi_2\rangle) = 1$. By structural induction, $A^{min}(\langle s, \varphi_2\rangle) = 1$ gives us $s \models \varphi_2$. Thus, any run $\sigma = s \ldots$ and position $p = 0$ satisfy conditions 23, 24 and 25 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 $A^{min}(\langle s, \varphi_1\rangle) = 1$ and $A^{min}(\langle s', E\ \varphi_1\ U_{\leq k-w}\ \varphi_2\rangle) = 1$. By structural induction it follows from $A^{min}(\langle s, \varphi_1\rangle) = 1$ 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 23, 24 and 25 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 23 holds because $\sigma'(p') = \sigma(p)$ and $\sigma(p) \models \varphi_2$.
- Condition 24 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 25 holds due to the fact that $W_\sigma(p) \leq k - w$ implies $W_{\sigma'}(p') \leq k$, because $W_{\sigma'}(p') - W_\sigma(p) = w$.

We have now shown that $A^{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$ implies that there exists a run σ starting from s and a position p satisfying conditions 23, 24 and 25 for $k' \leq k$. Thus, given the semantics it follows that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$.

$\Leftarrow$: Assume that $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$, we now show that this implies $A^{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 1$. From the semantics it follows that there is a run σ and position p satisfying conditions 23, 24 and 25 for $k' \leq k$ Let $s = s_0$, then we can write σ as follows.

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

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

Base Case ($i = p$): By condition 23 of the semantics, $s_p \models \varphi_2$, which by structural induction on φ implies $A^{min}(\langle s_p, \varphi_2\rangle) = 1$. In Figure 2(e), 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, $A^{min}(\langle s_p, \varphi_2\rangle) = 1$ implies $A^{min}(\langle s_p, E\ \varphi_1\ U_{\leq k-W_\sigma(i)}\ \varphi_2\rangle) = 1$, which proves our base case.

Inductive Step ($i < p$): By condition 24 of the semantics, $s_i \models \varphi_1$, which by structural induction on φ implies $A^{min}(\langle s_i, \varphi_1\rangle) = 1$. By induction on i, we know that $A^{min}(\langle s_{i+1}, E\ \varphi_1\ U_{\leq k-W_\sigma(i+1)}\ \varphi_2\rangle) = 1$ holds. In Figure 2(e), 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 $A^{min}(v) = 1$ for all configurations v of the target-set of the hyper-edge e, then it must follow that $A^{min}(\langle s_i, E\ \varphi_1\ U_{\leq k-W_\sigma(i)}\ \varphi_2\rangle) = 1$ for all $i \leq p$.

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

$$\sigma(p) \models \varphi_2 \tag{27}$$

$$\sigma(j) \models \varphi_1 \text{ , for all } j < p \tag{28}$$

$$W_\sigma(p) \leq k' \tag{29}$$

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

We denote the iteration in which a configuration v was first assigned 1, as $Z(v)$, formally we write the auxiliary function Z as in Equation 26, shown in the previous case.

For any configuration v it holds that $Z(v) < \infty$ if and only if $A^{min}(v) = 1$, as a pre fixed-point assignment must be reached in a finite number of iterations. Considering $Z(v)$ for a configuration $v = \langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle$, where $A^{min}(v) = 1$, 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. In Figure 2(f) we see that there are at most two 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) $\mathrm{Z(v)} - 1 = \max \begin{cases} Z(\langle s, \varphi_1 \rangle) \\ Z(\langle s', A\ \varphi_1\ U_{\leq k-w}\ \varphi_2 \rangle) \quad \text{for all } s' \text{ , s.t. } s \xrightarrow{w} s' \end{cases}$

For any configuration $v = \langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$, we now show by induction on $Z(v)$ that $A^{min}(v) = 1$ implies that for any run $\sigma = s \ldots$, there is a position p satisfying conditions 27, 28 and 29 for $k' \leq k$. We observe that $Z(v)$ is always greater than 1, seeing that v does not have a trivial hyper-edge $(v, \emptyset)$, and only configurations with trivial hyper-edges are assigned the value 1 in F^1.

Base Case ($Z(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 2$): It must be the case that (A) holds, as it is not possible for any configuration on the form $u = \langle s', A\ \varphi_1\ U_{\leq k-w}\ \varphi_2 \rangle$ to have $Z(u) = 1$. From case (A), we have that $Z(\langle s, \varphi_2 \rangle) = 1$ which implies that $A^{min}(\langle s, \varphi_2 \rangle) = 1$. Hence, by structural induction it follows that $s \models \varphi_2$. For any run $\sigma = s \ldots$ we have that $p = 0$ is a position that satisfies conditions 27, 28 and 29 for $k' \leq k$.

Inductive Step ($Z(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) > 2$): Once more, we consider cases (A) and (B). If case (A) holds then for any run $\sigma = s \ldots$ we have position $p = 0$ that satisifes the conditions as before. If (B) is the case, we have that $A^{min}(\langle s, \varphi_1 \rangle) = 1$ and for all s_i s.t. $s \xrightarrow{w_i} s_i$, it holds that $A^{min}(\langle s_i, A\ \varphi_1\ U_{\leq k-w_i}\ \varphi_2 \rangle) = 1$, which by induction on $Z(\langle s_i, A\ \varphi_1\ U_{\leq k-w_i}\ \varphi_2 \rangle)$ implies that $s_i \models \langle s_i, A\ \varphi_1\ U_{\leq k-w_i}\ \varphi_2 \rangle$. By structural induction it follows from $A^{min}(\langle s, \varphi_1 \rangle) = 1$ that $s \models \varphi_1$.

Considering any run σ starting from s, we see that this run must be on the form $\sigma = s \xrightarrow{w_i} s_i \ldots$ for some s_i, s.t. $s \xrightarrow{w_i} s_i$. For any postfix $\sigma' = s_i \ldots$ of σ, there exists a position p' satisfying conditions 27, 28 and 29 for $k' \leq k - w_i$, as $s_i \models A\ \varphi_1\ U_{\leq k-w_i}\ \varphi_2$. Thus, given σ we have that $p = p' + 1$ is a position satisfying conditions 27, 28 and 29 for $k' \leq k$.

- Condition 27 holds because $\sigma(p) = \sigma'(p')$ and $\sigma'(p') \models \varphi_2$.
- Condition 28 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 29 holds due to the fact that $W'_\sigma(p') \leq k - w_i$ implies $W_\sigma(p) \leq k$, because $W_\sigma(p) - W'_\sigma(p') = w_i$.

We have now shown that $A^{min}(\langle s, A\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle) = 1$ implies that for any run σ starting from s, there is a position p satisfying conditions 27, 28 and 29 for $k' \leq k$. Thus, it follows from the semantics that $s \models A\ \varphi_1\ U_{\leq k}\ \varphi_2$.

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

Considering the formula $\varphi = A\ \varphi_1\ U_{\leq k}\ \varphi_2$ and state s, if $s \models \varphi$ then it follows from the semantics that for any run σ starting from s, there is a position p that satisfies conditions 27, 28 and 29 for $k' \leq k$. Given $\sigma = s \ldots$, the existence of p also implies the existence of some smallest

p'. By $\rho(s, \varphi)$, we denote maximum such smallest p' for any run starting from s.

$$\rho(s, A\ \varphi_1\ U_{\leq k}\ \varphi_2) = \max \left\{ \begin{array}{c} \text{smallest } p \text{ satisfying} \\ 27,\ 28 \text{ and } 29 \text{ for } k' \leq k \end{array} \;\middle|\; \text{for all } \sigma = s \ldots \right\}$$

Considering the state s and formula $\varphi = A\ \varphi_1\ U_{\leq k}\ \varphi_2$, we now show that $s \models \varphi$ implies $A^{min}(\langle s, \varphi \rangle) = 1$ by induction on $\rho(s, \varphi)$.

Base Case ($\rho(s, \varphi) = 0$): In this case we have that for any run $\sigma = s \ldots$, the position $p = 0$ satisifies conditions 27, 28 and 29 for $k' \leq k$. Condition 27 implies that $s \models \varphi_2$ which by structural induction implies $A^{min}(\langle s, \varphi_2 \rangle) = 1$. In Figure 2(f) we see that $\langle s, \varphi \rangle$ has a hyper-edge to $\langle s, \varphi_2 \rangle$. Thus, it must hold that $A^{min}(\langle s, \varphi \rangle) = 1$.

Inductive Step ($\rho(s, \varphi) > 0$): In this case we have that for any run $\sigma = s \ldots$, there is a position $p \leq \rho(s, \varphi)$ which satisifies conditions 27, 28 and 29 for $k' \leq k$. We also know that $p > 0$, because if p were 0 for some run $\sigma = s \ldots$, then this would imply $s \models \varphi_2$, in which case the smallest p would be 0 for any run $\sigma = s \ldots$. Thus, as $\rho(s, \varphi) > 0$ this cannot be the case and $p > 0$, which from condition 28 implies that $s \models \varphi_1$ and by structural induction we have that $A^{min}(\langle s, \varphi_1 \rangle) = 1$.

In Figure 2(f) we see that $\langle s, \varphi \rangle$ has a hyper-edge to the target-set containing $\langle s, \varphi_1 \rangle$ and $\langle s_i, A\ \varphi_1\ U_{\leq k - w_i}\ \varphi_2 \rangle$ for all s_i s.t. $s \xrightarrow{w_i} s_i$. Thus, to show that $A^{min}(\langle s, \varphi \rangle) = 1$ we need only show that $A^{min}(\langle s_i, A\ \varphi_1\ U_{\leq k - w_i}\ \varphi_2 \rangle) = 1$ for all s_i s.t. $s \xrightarrow{w_i} s_i$.

Consider some s_i s.t. $s \xrightarrow{w_i} s_i$, then any run $\sigma' = s_i \ldots$ starting from s_i must be a postfix of some run $\sigma = s \xrightarrow{w_i} s_i \ldots$ starting from s. We know that given σ, there exists a position $p \leq \rho(s, \varphi)$ satisifying conditions 27, 28 and 29 for $k' \leq k$. Now considering σ' we have that position $p' = p - 1$ also satisifies these conditions for $k' \leq k - w_i$.

- Condition 27 holds because $\sigma'(p') = \sigma(p)$ and $\sigma(p) \models \varphi_2$.
- Condition 28 holds since $\sigma'(j-1) = \sigma(j)$ and $\sigma(j) \models \varphi_1$ for all $j < p$.
- Condition 29 holds due to the fact that $W_\sigma(p) \leq k$ implies $W_{\sigma'}(p') \leq k - w_i$, because $W_\sigma(p) - W'_\sigma(p') = w_i$.

As the p' constructed is strictly smaller than p, we have that $\rho(s_i, A\ \varphi_1\ U_{\leq k - w_i}\ \varphi_2) < \rho(s, \varphi)$. Thus, by induction it follows from $s_i \models A\ \varphi_1\ U_{\leq k - w_i}\ \varphi_2$ that $A^{min}(s_i, A\ \varphi_1\ U_{\leq k - w_i}\ \varphi_2) = 1$. As all configurations in a hyper-edge for $\langle s, \varphi \rangle$ are assigned the value 1, it must hold that $A^{min}(\langle s, \varphi \rangle) = 1$.

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

$\Rightarrow$: Assume that $A^{min}(\langle s, EX_{\leq k}\ \varphi \rangle) = 1$, then it holds that $s \models EX_{\leq k}\ \varphi$. In Figure 2(g), the configuration $\langle s, EX_k\ \varphi \rangle$ has a hyper-edge for every $s_i \in \{ s_i \mid s \xrightarrow{w_i} s_i \text{ and } w_i \leq k \}$. Clearly, $A^{min}(\langle s, EX_{\leq k}\ \varphi \rangle) = 1$ if and only if $A^{min}(\langle s_i, \varphi \rangle) = 1$ 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 $A^{min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$. From the semantics, it must be the case that there exists an s_i, such that $s \xrightarrow{w_i} s_i$, with $w_i \leq k$, it holds that $s_i \models \varphi$. By the induction hypothesis, this implies that $A^{min}(\langle s_i, \varphi\rangle) = 1$ for any such s_i. Since A^{min} is a pre fixed-point assignment, a hyper-edge in Figure 2(g) ensures that $A^{min}(\langle s, EX_{\leq k}\ \varphi\rangle) = 1$.

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

$\Rightarrow$: Assume that $A^{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$, then it holds that $s \models AX_{\leq k}\ \varphi$. In Figure 2(h), the configuration $\langle s, AX_{\leq k}\ \varphi\rangle$ has a single hyper-edge with a target set on the form $\{\langle s_1, \varphi\rangle, \ldots, \langle s_n, \varphi\rangle\}$, for every s_i, such that $s \xrightarrow{w_i} s_i$ and $w_i \leq k$. It is clear that $A^{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$ if and only if $A^{min}(\langle s_i, \varphi\rangle) = 1$ 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 $A^{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$. By the semantics it must be that case that $s_i \models \varphi$, for all s_i such that $s \xrightarrow{w_i} s_i$, where $w_i \leq k$. By the induction hypothesis this implies that $A^{min}(\langle s_i, \varphi\rangle) = 1$ for all such s_i. As A^{min} is a pre fixed-point assignment, the hyper-edge in Figure 2(h) ensures that $A^{min}(\langle s, AX_{\leq k}\ \varphi\rangle) = 1$.

□

A.2 Proofs Related to Symbolic Dependency Graph

This section is a slightly modified excerpt from our pre-specialization project [14].

We begin with a technical lemma.

Lemma 4. *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.*

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

It holds that $F^i(c_i) = A^{min}(c_i)$.

Proof. To prove that $A^{min}(c_i) = F^i(c_i)$, we show that Equation (37) holds. It then trivially follows that $F^i(c_i)$ is the minimum pre fixed-point assignment of c_i, because no future smallest assignment in any iteration $j > i$ becomes less than $F^i(c_i)$.

To show that Equation (37) 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 (30).

$$F^i(c_{i+1}) > F^{i+1}(c_{i+1}) \tag{30}$$
$$F^{i+1}(c_{i+1}) = \max\{w' + F^i(u') \mid (w', u') \in T\} \tag{31}$$
$$F^{i-1}(u) > F^i(u) \tag{32}$$
$$F^i(u) \geq F^i(c_i) \tag{33}$$

This implies that there exists a hyper-edge $(c_{i+1}, T) \in H$ such that Equation (31) holds. Because the value $F^{i+1}(c_{i+1})$ 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 (32).

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 (33), because no other assignment made in the i'th iteration is smaller than $F^i(c_i)$.

$$\max\{w' + F^i(u') \mid (w', u') \in T\} \geq w + F^i(u) \tag{34}$$
$$F^{i+1}(c_{i+1}) \geq w + F^i(u) \tag{35}$$
$$F^{i+1}(c_{i+1}) \geq w + F^i(c_i) \tag{36}$$
$$F^{i+1}(c_{i+1}) \geq F^i(c_i) \tag{37}$$

As the hyper-edge branch (w, u) for which the value of u changed is in T, we observe that $w + F^i(u)$ must be less than equal to the right hand side of Equation (31) giving us Equation (34). Substituting this back into Equation (31) and we get Equation (35). We now recall the lower-bound on $F^i(u)$ from Equation (33) in order to write Equation (36). Thus, we get Equation (37) as w must be non-negative. □

Proof of Theorem 2
Computing the minimum pre fixed-point assignment of $G = (V, H, C)$ by repeated application of the functor F takes $O(|V| \cdot |C| \cdot (|H| + |C|))$ time.

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. Note that from the construction we have that there are always more configurations than cover-edges. After we establish that the algorithm terminates after no more than $|V| \cdot |C|$ iterations, the claim is proved.

If we consider a symbolic dependency graph without cover-edges $G = (V, H, \emptyset)$, we have that the minimum pre fixed-point assignment is reached within $|V|$ iterations. This follows from Lemma 4 that states that after each iteration, there is at least one configuration that reaches its minimum pre fixed point assignment.

Assume now that the symbolic dependency graph contains cover-edges. It is clear that once the value of a source configuration for a cover-edge is updated, it takes the value 0 and 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 and then 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. □

Proof of Theorem 4
Let $\mathcal{K} = (S, \mathcal{AP}, L, \rightarrow)$ be a WKS, $s \in S$ a state, 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 $A^{min}(\langle s, \varphi \rangle) = 0$.

Proof. We prove Theorem 4 by observing that there is two kinds of configurations in the symbolic dependency graph rooted with $\langle s, \varphi \rangle$. We have that configurations on 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. We shall refer to these configurations as *symbolic configurations*, and all other configurations as *concrete configurations*.

Notice that the bound for symbolic configurations is "?", while $\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2 \rangle$ is a concrete configuration. With the introduction of concrete and symbolic configurations, we now present two invariants for the symbolic encoding.

i) Concrete configurations $\langle s, \varphi \rangle$ can only obtain the values 0 or ∞, where $A^{min}(\langle s, \varphi \rangle) = 0$ if and only if $s \models \varphi$.
ii) For a symbolic configuration $v = \langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$ it holds that $A^{min}(v) = k$ if and only if $s \models E\ \varphi_1\ U_{\leq k'}\ \varphi_2$ for any $k' \geq k$.
(A similiar invariant applies to configurations for the universal until-formula).

It is easy to see that Theorem 4 follows trivially from Invariant (i). Thus, we need only show that these invariants hold by structural induction on φ.

(I) For $\varphi = \mathbf{true}$ we show that Invariant (i) holds for all configurations $\langle s, \mathbf{true} \rangle$. Because $s \models \mathbf{true}$ always holds we need only show that

$A^{min}(\langle s, \mathbf{true}\rangle) = 0$. In Figure 2(a) there is a hyper-edge from the configuration $\langle s, \mathbf{true}\rangle$ to the empty target set. Hence, we have that $A(v) = 0$ for any pre fixed-point assignment A of G.

(II) For $\varphi = a$ we prove Invariant (i), i.e. $A^{min}(\langle s, a\rangle) = 0$ 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 2(b), there is a hyper-edge from the configuration $\langle s, a\rangle$ to the empty target set. Like in the previous case this means that $A^{min}(\langle s, a\rangle) = 0$, which leaves us to consider the case when $a \notin L(s)$. In this case it is clear that $s \not\models a$ and by the side-condition in Figure 2(b), we can conclude that there is no hyper-edge from the configuration $\langle s, a\rangle$ when $a \notin L(s)$. Thus, we have $A^{min}(\langle s, a\rangle) = \infty$ since A^{min} is the minimum pre fixed-point assignment.

(III) For $\varphi = \varphi_1 \wedge \varphi_2$ we show that Invariant (i) holds. First we show that $A^{min}(\langle s, \varphi_1 \wedge \varphi_2\rangle)$ is either ∞ or 0, and $A^{min}(\langle s, \varphi_1 \wedge \varphi_2\rangle) = 0$ if and only if $s \models \varphi_1 \wedge \varphi_2$. Since sub-configurations $\langle s, \varphi_1\rangle$ and $\langle s, \varphi_1\rangle$ are concrete (Figure 2(c)) it follows by structural induction that their assignments only evaluate to either 0 or ∞. Furthermore, we have $A^{min}(\langle s, \varphi_1\rangle) = 0$ and $A^{min}(\langle s, \varphi_2\rangle) = 0$, if and only if $s \models \varphi_1$ and $s \models \varphi_2$, which following the semantics implies $s \models \varphi_1 \wedge \varphi_2$.

(IV) For $\varphi = \varphi_1 \vee \varphi_2$ Invariant (i) can be shown with arguments similar to those used previously for conjunction.

(V) For $\varphi = E\ \varphi_1\ U_{\leq k}\ \varphi_2$ we show Invariant (i), i.e. $A^{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 0$ if and only if $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$ for all $s \in S$. From Figure 7(a) we see that any 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 $v = \langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle$. By structural induction we have from Invariant (ii) that $A^{min}(v) \leq k$ if and only if $s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$. Thus, we have shown Invariant (i), as cover-edges can only assign the value 0.

(VI) For $\varphi = E\ \varphi_1\ U_{\leq ?}\ \varphi_2$ we show Invariant (ii), i.e. that $A^{min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) = k$ if and only if $s \models E\ \varphi_1\ U_{\leq k'}\ \varphi_2$ for any $k' \geq k$.
Recall the semantics for the satisfaction of the formula $E\ \varphi_1\ U_{\leq k}\ \varphi_2$, requires that for some $k' \leq k$, there exists a run σ and a position $p \geq 0$ satisfying the following conditions.

$$\sigma(p) \models \varphi_2 \tag{38}$$

$$\sigma(j) \models \varphi_1 \text{ , for all } j < p \tag{39}$$

$$W_\sigma(p) \leq k' \tag{40}$$

$\Rightarrow$: Assume that $A^{min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) = k$, we now show that this implies the existence of a run σ and position p satisfying conditions 38, 39 and 40 for $k' \leq k$. By the semantics this obviously implies $s \models E\ \varphi_1\ U_{\leq k'}\ \varphi_2$ for any $k' \geq k$.
We denote the iteration in which a configuration v was first assigned the value k, as $Z_k(v)$. Formally we write the auxiliary function Z_k as follows.

$$Z_k(v) = \begin{cases} i & \text{if } F^i(v) \leq k \text{ and } F^{i-1}(v) > k \\ \infty & \text{otherwise} \end{cases} \tag{41}$$

For any configuration v it holds that $Z_k(v) < \infty$ if and only if $A^{min}(v) \leq k$, as a fixed-point must be reached in a finite number of iterations. Considering $Z_k(v)$ for a configuration $v = \langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$, where $A^{min}(v) \leq k$, we see that in iteration $Z_k(v) - 1$, the assignment of some configuration in the target-set for a hyper-edge to v must have been changed to k. From Figure 2(e) 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) $Z_k(\langle s, \varphi_2 \rangle) = Z_k(v) - 1$, or
B) $\max\{Z_k(\langle s, \varphi_1 \rangle), Z_{k-w}(\langle s', E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle)\} = Z_k(v) - 1$, for some s', s.t. $s \xrightarrow{w} s'$.

We now show that $A^{min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle) = k$ implies the existence of a run σ and a position p satisifying conditions 38, 39 and 40 for $k' \leq k$, by induction on $Z_k(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle)$.

First we observe that $Z_k(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle)$ is always greater than 1, as only configurations v having trivial hyper-edges $(v, \emptyset)$ are assigned 0 in the first iteration of F.

Base Case ($Z_k(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle) = 2$): In this case we know that case (A) must hold, seeing that no configuration $u = \langle s', E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle$ can have $Z_{k-w}(u) = 1$. From case (A), we have that $Z_k(\langle s, \varphi_2 \rangle) = 1$ and as this is a concrete configuration, it holds that $A^{min}(\langle s, \varphi_2 \rangle) = 0$ by Invariant i. From here it also follows that $A^{min}(\langle s, \varphi_2 \rangle) = 0$ implies $s \models \varphi_2$. Thus, any run $\sigma = s \ldots$ and position $p = 0$ satisfy conditions 38, 39 and 40 for $k' = 0$, hence, it also holds for $k \geq k'$.

Inductive Step ($Z_k(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle) > 2$): 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 $Z_k(\langle s, \varphi_1 \rangle) \leq \infty$ which implies $A^{min}(\langle s, \varphi_1 \rangle) = 0$ as $\langle s, \varphi_1 \rangle$ is a concrete configuration. Futhermore, it follows from Invariant (ii) by structural induction that $s \models \varphi_1$.

Because $Z_{k-w}(\langle s', E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle) < Z_k(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle)$ it follows by induction that there is a run $\sigma = s' \ldots$ and a position p that satisfy conditions 38, 39 and 40 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 38 holds because $\sigma'(p') = \sigma(p)$ and $\sigma(p) \models \varphi_2$.
- Condition 39 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 40 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 $A^{min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2 \rangle) \leq k$. From the semantics it follows that there is a run σ and position p satisfying conditions 38, 39 and 40 for $k' \leq k$ Let $s = s_0$, then we can write σ as follows.

$$\sigma = s_0 \xrightarrow{w_1} s_1 \ldots s_{p-1} \xrightarrow{w_p} s_p \ldots$$

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

Base Case ($i = p$): By condition 38 of the semantics, $s_p \models \varphi_2$, which by structural induction on φ implies $A^{min}(\langle s_p, \varphi_2\rangle) = 0$ because $\langle s_p, \varphi_2\rangle$ is a concrete configuration. In Figure 7(c), we observe that there is a hyper-edge from $\langle s_p, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle$ to $\langle s_p, \varphi_2\rangle$, thus, $A^{min}(\langle s_p, \varphi_2\rangle) = 0$ implies $A^{min}(\langle s_p, E\ \varphi_1\ U_{\leq 0}\ \varphi_2\rangle) = 0$, which proves our base case.

Inductive Step ($i < p$): By condition 39 of the semantics, $s_i \models \varphi_1$, which by structural induction on φ implies $A^{min}(\langle s_i, \varphi_1\rangle) = 0$. By induction on i, we know that $A^{min}(\langle s_{i+1}, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) \leq k - W_\sigma(i+1)$ holds.

In Figure 7(c), we observe that there is a hyper-edge e from $\langle s_i, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle$ to the target-set $\langle s_i, \varphi_1\rangle$ and $\langle s_{i+1}, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle$. We also notice that e has the transition weight between s_i and s_{i+1}, w_{i+1}, on the hyper-edge branch to $\langle s_{i+1}, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle$. Thus, from the semantics of hyper-edges it follows that $A^{min}(\langle s_i, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) \leq k - W_\sigma(i+1) + w_{i+1}$. But as $W_\sigma(i) + w_{i+1} = W_\sigma(i)$ we have that $A^{min}(\langle s_i, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) \leq k - W_\sigma(i)$, which proves our inductive case.

(VII) For $\varphi = A\ \varphi_1\ U_{\leq k}\ \varphi_2$ we have that $A^{min}(\langle s, E\ \varphi_1\ U_{\leq k}\ \varphi_2\rangle) = 0$ 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 $\varphi = E\ \varphi_1\ U_{\leq k}\ \varphi_2$.

(VIII) For $\varphi = A\ \varphi_1\ U_{\leq ?}\ \varphi_2$ it can be shown that $A^{min}(\langle s, E\ \varphi_1\ U_{\leq ?}\ \varphi_2\rangle) = k$ if and only if $s \models A\ \varphi_1\ U_{\leq k'}\ \varphi_2$ for all $k' \geq k$. The proof strategy is an adaptation of the approach for ordinary dependency graphs. In particular it is similar to the proof strategy applied for $E\ \varphi_1\ U_{\leq ?}\ \varphi_2$, which was adapted from the proof for ordinary dependency graphs.

(IX) For $\varphi = EX_{\leq k}\ \varphi$ we observe in Figure 2(g) that all successor configurations are concrete. It is straightforward to adapt the proof strategy used for ordinary dependency graphs for this case.

(X) For $\varphi = AX_{\leq k}\ \varphi$, shown in Figure 2(h), all successor configurations are again conrete. Once again, it is easy to adapt the proof strategy for this case.

□

A.3 Resumé

I denne afhandling introduceres tre teknikker til verifikation af vægtet CTL (WCTL). Dertil afdækkes begrænsinger af de henholdsvise tilgange til problemet. Tidligere løsningsforslag til verifikation af fragmentet $\text{WCTL}_{\leq}$, uden negation, samt begrænset til øvre grænser på modaliteterne gengives i form af en reduktion til fikspunktberegning af afhængighedsgrafer og deres symbolske udvidelse. Disse teknikker muliggør lokal model checking, men er begrænset til alterneringsfrie fikspunkter, hvilket indskrænker udtrykskraften i forhold til logikken.

Vi udvider disse ideer i form min-max grafer som en teknik til verikation af den fuldstændige vægtet CTL logik. I modsætning til de førnævnte teknikker, understøttes alternerende fikspunkter ved denne tilgang. Derved kan der formuleres en mere udtryksrig logik, hvor invariante egenskaber, samt den svage 'until' modalitet med nedre grænser kan benyttes. Vi beskriver endvidere både en global og lokal algoritme til fikspunktsberegning, med henblik på at verificere logiske udsagn med min-max grafer.

Algoritmerne er implementeret i et online værktøj, som gør WCTL model verifikation af vægtet CCS modeller let tilgængeligt. Gennem eksperimenter med vores implementering demonstrerer vi fordelene ved lokal fikspunktsberegning til verifikation af egenskaber med de tre løsningsforslag. På denne baggrund konkluderes det at en lokal tilgang kan give anledning til en dramatisk forbedring af verifikationstiden, når både grænser og de logiske egenskaber tillader mange mulige vidner om sandheden af udsagnet.