Local Model Checking of Weighted CTL with Upper-Bound Constraints

Jonas Finnemann Jensen, Kim Guldstrand Larsen,
Jiří Srba, and Lars Kaerlund Oestergaard

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

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

1 Introduction

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

Within the area of model checking a number of state-machine based modeling formalisms has emerged, allowing for such quantitative aspects to be expressed. In particular, timed automata (TA) [1], and the extensions to weighted timed automata (WTA) [6,2] are popular and tool-supported formalisms that allow for such constraints to be modeled.

Interesting behavioural properties of TAs and WTAs may be expressed in natural weight-extended versions of classical temporal logics such as CTL for

branching-time and LTL for linear-time. Just as TCTL and MTL provide extensions of CTL and LTL with time-constrained modalities, WCTL and WMTL are extensions with weight-constrained modalities interpreted with respect to WTAs. Unfortunately, the addition of weight now turns out to come with a price: whereas the model-checking problems for TAs with respect to TCTL and MTL are decidable, it has been shown that model-checking WTAs with respect to WCTL is undecidable [9].

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

Our results are based on a novel symbolic extension of the dependency graph framework of Liu and Smolka [16] where they encode boolean equation systems and offer global and local algorithms for computing minimal and maximal fixed points in linear time. Whereas a direct encoding of our model checking problem into dependency graphs leads to a pseudo-polynomial algorithm[1], the novel symbolic dependency graphs allow for a polynomial encoding and a polynomial time fixed-point computation. Most importantly, the symbolic dependency graph encoding enables us to perform a symbolic local fixed-point evaluation. Experiments with the various approaches (direct versus symbolic encoding, global versus local algorithm) have been conducted on a large number of cases, demonstrating that the combined symbolic and local approach is the most efficient one. For model-checking problems with affirmative outcome, this combination is often one order or magnitude faster than the other approaches.

Related Work

Laroussinie, Markey and Oreiby [14] consider the problem of model checking durational concurrent game structures with respect to timed ATL properties, offering a PTIME result in the case of non-punctual constraints in the formula. Restricting the game structures to a single player gives a setting similar to ours, as timed ATL is essentially WCTL. However, in contrast to [14], we do allow transitions with zero weight in the model, making a fixed-point computation necessary. As a result, the corresponding CTL model checking (with no weight constraints) is a special instance of our approach, which is not the case for [14]. Most importantly, the work in [14] does not provide any local algorithm, which our experiments show is crucial for the performance. No implementation is provided in [14].

Buchholz and Kemper [10] propose a valued computation tree logic (CTL$) interpreted over a general set of weighted automata that includes CTL in the logic as a special case over the boolean semiring. For model checking CTL$ formulae they describe a matrix-based algorithm. Their logic is more expressive than the one proposed here, since they support negation and all the comparison

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

operators. In addition, they permit nested CTL formulae and can operate on max/plus semirings in $O(\min(log(t) \cdot mm, t \cdot nz))$ time, where t is the number of vector matrix products, mm is the complexity of multiplying two matrices of order n and nz is the number of non-zero elements in special matrix used for checking "until" formulae up to some bound t. However, they do not provide any on-the-fly technique for verification.

Another related work [8] shows that the model-checking problem with respect to WCTL is PSPACE-complete for one-clock WTAs and for TCTL (the only cost variable is the time elapsed).

Several approaches to on-the-fly/local algorithms for model checking the modal mu-calculus have been proposed. Andersen [3] describes a local algorithm for model checking the modal mu-calculus for alternation depth one running in $O(n \cdot log(n))$ (where n is the product of the size of the assertion and the labeled transition system). Liu and Smolka[16] improve on the complexity of this approach with a local algorithm running in $O(n)$ (where n is the size of the input graph) for evaluating alternation-free fixed points. This is also the algorithm that we apply for WCTL model checking and the one we extend for symbolic dependency graphs. Cassez et. al. [11] present another symbolic extension of the algorithm by Liu and Smolka; a zone-based forward, local algorithm for solving timed reachability games. Later Liu, Ramakrishnan and Smolka [15] also introduce a local algorithm for the evaluation of alternating fixed points with the complexity $O(n + (\frac{n+ad}{ad})^{ad})$, where ad is the alternation depth of the graph. We do not consider the evaluation of alternating fixed points in the weighted setting and this is left for the future work.

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

2 Basic Definitions

Let $\mathbb{N}_0$ be the set of nonnegative integers. 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.

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

[2] A blocking WKS can be turned into a nonblocking one by introducing a new state with no atomic propositions, zero-weight self-loop and with zero-weight transitions from all blocking states into this newly introduced state.

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

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

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

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

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

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

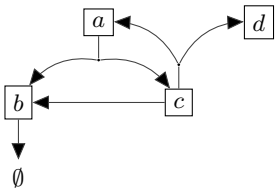
$s \models \mathbf{true}$	
$s \models a$	if $a \in L(s)$
$s \models \varphi_1 \wedge \varphi_2$	if $s \models \varphi_1$ and $s \models \varphi_2$
$s \models \varphi_1 \vee \varphi_2$	if $s \models \varphi_1$ or $s \models \varphi_2$
$s \models E\ \varphi_1\ U_{\leq k}\ \varphi_2$	if there exists a run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

3 Dependency Graph

In this section we present the dependency graph framework and a local algorithm for minimal fixed-point computation as originally introduced by Liu and Smolka [16]. This framework can be applied to model checking of the alternation-free modal mu-calculus, including the CTL logic. Later, in Section 4, we demonstrate how to extend the framework from CTL to WCTL.

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

Let $G = (V, E)$ be a dependency graph. For a hyper-edge $e = (v, T)$, we call v the source configuration and T the target (configuration) set of e. For a configuration v, the set of its successors is given by $succ(v) = \{(v, T) \in E\}$.



$a = b \wedge c$
$c = b \vee (a \wedge d)$
$b = \mathbf{true}$

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

Fig. 1. A dependency graph, function F, and four iterations of the global algorithm

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

By taking the standard component-wise ordering $\sqsubseteq$ on assignments, where $A \sqsubseteq A'$ if and only if $A(v) \leq A'(v)$ for all $v \in V$ (assuming that $0 < 1$), we get by Knaster-Tarski fixed-point theorem that there exists a unique minimum pre fixed-point assignment, denoted by A_{min}.

The minimum pre fixed-point assignment A_{min} of G can be computed by repeated applications of the monotonic function F from assignments to assignments, starting from A_0 where $A_0(v) = 0$ for all $v \in V$, and where

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

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

Example 1. Figure 1 shows a dependency graph, its corresponding function F given as a boolean equation system, and four iterations of the global algorithm (sufficient to compute the minimum pre fixed-point assignment). Configurations in the dependency graph are illustrated as labeled squares and hyper-edges are drawn as a span of lines to every configuration in the respective target set.

In model checking we are often only interested in the minimum pre-fixed point assignment $A_{min}(v)$ for a specific configuration $v \in V$. For this purpose, Liu and Smolka [16] suggest a local algorithm presented with minor modifications[3] in Algorithm 1. The algorithm maintains three data-structures throughout its execution: an assignment A, a dependency set D for every configuration and a set of hyper-edges W. The dependency set $D(v)$ for a configuration v maintains

[3] At line 12 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 empty here, leading to an incorrect propagation.

Algorithm 1: Liu-Smolka Local Algorithm

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

```
1  Let A(v) = ⊥ for all v ∈ V
2  A(v_0) = 0; D(v_0) = ∅
3  W = succ(v_0)
4  while W ≠ ∅ do
5  |  let e = (v, T) ∈ W
6  |  W = W \ {e}
7  |  if A(u) = 1 for all u ∈ T then
8  |  |  A(v) = 1; W = W ∪ D(v)
9  |  else if there is u ∈ T such that A(u) = 0 then
10 |  |  D(u) = D(u) ∪ {e}
11 |  else if there is u ∈ T such that A(u) = ⊥ then
12 |  |  A(u) = 0; D(u) = {e}; W = W ∪ succ(u)
13 return A(v_0)
```

a list of hyper-edges that were processed under the assumption that $A(v) = 0$. Whenever the value of $A(v)$ changes to 1, the hyper-edges from $D(v)$ must be reprocessed in order to propagate this change to the respective sources of the hyper-edges.

Theorem 1 (Correctness of Local Algorithm [16]). *Given a dependency graph $G = (V, E)$ and a configuration $v_0 \in V$, Algorithm 1 computes the minimum pre-fixed point assignment $A_{min}(v_0)$ for the configuration v_0.*

As argued in [16], both the local and global model checking algorithms run in linear time.

4 Model Checking with Dependency Graphs

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

Given a WKS $\mathcal{K}$, a state s of $\mathcal{K}$, and a WCTL formula φ, we construct a dependency graph where every configuration is a pair of a state and a formula. Starting from the initial pair $\langle s, \varphi \rangle$, the dependency graph is constructed according to the rules given in Figure 2.

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

Proof. By structural induction on the formula φ. □

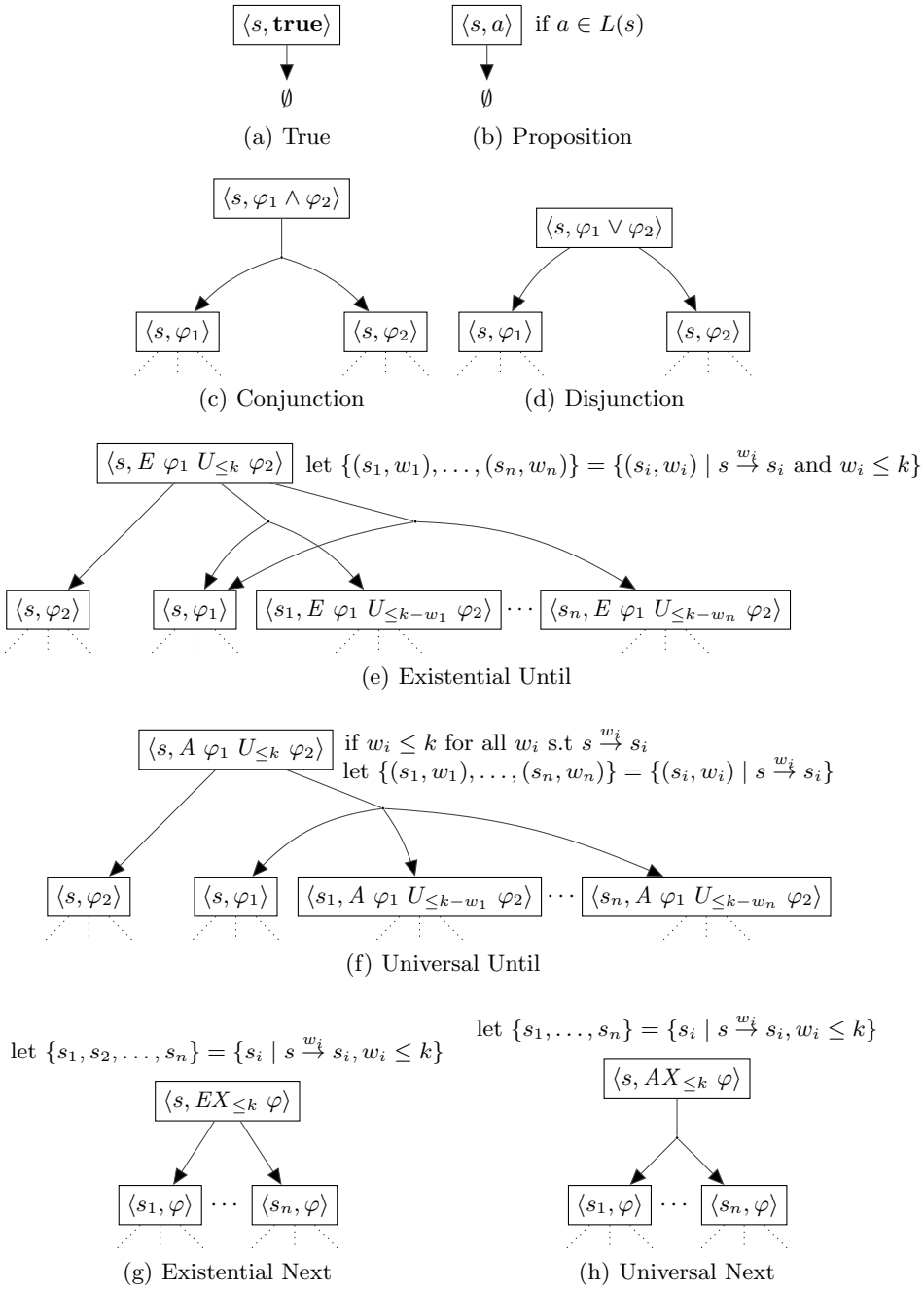


Fig. 2. Dependency graph encoding of state-formula pairs.

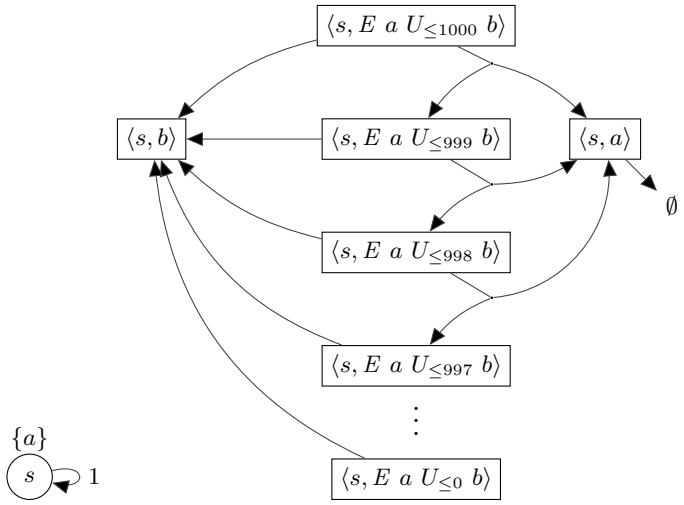


Fig. 3. A WKS and its dependency graph for the formula $E\ a\ U_{\leq 1000}\ b$

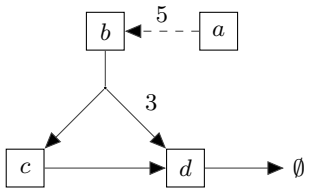
Clearly, to profit from the local algorithm by Liu and Smolka [16] presented in the previous section, we construct the dependency graph on-the-fly whenever successor configurations are requested by the algorithm. Such an exploration gives us often more efficient local model checking algorithm compared to the global one (see Section 7).

However, the drawback of this approach is that we may need to construct exponentially large dependency graphs. This is demonstrated in Figure 3 where a single-state WKS on the left gives rise to a large dependency graph on the right where its size depends on the bound in the formula. Hence this method gives us only a pseudo-polynomial algorithm for model checking WCTL.

5 Symbolic Dependency Graph

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

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



(a) A symbolic dependency graph

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

(b) Minimum pre fixed-point computation

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

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

Figure 4(a) shows an example of a SDG. Hyper-edges are denoted by solid lines and hyper-edge branches have weight 0 unless they are annotated with another weight. Cover-edges are drawn as dashed lines annotated with a cover-condition. We shall now describe a global algorithm for the computation of the minimum pre fixed-point. The main difference is that symbolic dependency graphs operate over the complete lattice $\mathbb{N}_0 \cup \{\infty\}$, contrary to standard dependency graphs that use only boolean values.

An assignment $A : V \to \mathbb{N}_0 \cup \{\infty\}$ in an SDG $G = (V, H, C)$ is a mapping from configurations to values. We denote the set of all assignments by $Assign$. A *pre fixed-point assignment* is an assignment $A \in Assign$ such that $A = F(A)$ where $F : Assign \to Assign$ is defined as

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

If we consider the partial order $\sqsubseteq$ over assignments of a symbolic dependency graph G such that $A \sqsubseteq A'$ if and only if $A(v) \geq A'(v)$ for all $v \in V$, then the function F is clearly monotonic on the complete lattice of all assignments ordered by $\sqsubseteq$. It follows by Knaster-Tarski fixed-point theorem that there exists a unique minimum pre fixed-point assignment of G, denoted A_{min}.

Notice that we write $A \sqsubseteq A'$ if for all configurations v we have $A(v) \geq A'(v)$ in the opposite order. Hence, $A_0(v) = \infty$ for all $v \in V$ is the smallest element in the lattice.

As the lattice is finite and there are no infinite decreasing sequences of weights (nonnegative integers), the minimum pre fixed-point assignment A_{min} of G can be computed by a finite number of applications of the function F on the smallest assignment A_0, where all configurations have the initial value ∞. So there exists an $m \in \mathbb{N}_0$ such that $F^m(A_0) = F^{m+1}(A_0)$, implying that $F^m(A_0) = A_{min}$ is the minimum pre fixed-point assignment of G. Figure 4(b) shows a computation of the minimum pre fixed-point assignment on our example.

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

Theorem 3. *The computation of the minimum post fixed-point assignment for an SDG $G = (V, H, C)$ by repeated application of the function F takes time $O(|V| \cdot |C| \cdot (|H| + |C|))$.*

We now propose a local algorithm for minimum pre fixed-point computation on symbolic dependency graphs, motivated by the fact that in model checking we are often interested in the value for a single given configuration only, hence we might be able (depending on the formula we want to verify) to explore only a part of the reachable state space.

Given a symbolic dependency graph $G = (V, H, C)$, Algorithm 2 computes the minimum pre fixed-point assignment $A_{min}(v_0)$ of a configuration $v_0 \in V$. The algorithm is an adaptation of Algorithm 1. We use the same data-structures as in Algorithm 1. However, the assignment $A(v)$ for each configuration v now ranges over $\mathbb{N}_0 \cup \{\bot, \infty\}$ where $\bot$ once again indicates that the value is unknown at the moment.

Table 1 lists the values of the assignment A, the set W (implemented as queue) and the dependency set D during the execution of Algorithm 2 on the SDG Figure 4(a). Each row displays the values before the i'th iteration of the while-loop. The value of the dependency set $D(a)$ for a is not shown in the table because it remains empty.

In order to prove the correctness of Algorithm 2, we extend the loop invariant for the local algorithm on dependency graphs [16] with weights.

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

Algorithm 2: Symbolic Local Algorithm

Input: A SDG $G = (V, H, C)$ and a configuration $v_0 \in V$
Output: Minimum pre fixed-point assignment $A_{min}(v_0)$ for v_0

```
1  Let A(v) = ⊥ for all v ∈ V
2  A(v_0) = ∞; W = succ(v_0)
3  while W ≠ ∅ do
4      Pick e ∈ W
5      W = W \ {e}
6      if e = (v, T) is a hyper-edge then
7          if ∃(w, u) ∈ T where A(u) = ∞ then
8              D(u) = D(u) ∪ {e}
9          else if ∃(w, u) ∈ T where A(u) = ⊥ then
10             A(u) = ∞; D(u) = {e}; W = W ∪ succ(u)
11         else
12             a = max{A(u) + w | (w, u) ∈ T}
13             if a < A(v) then
14                 A(v) = a; W = W ∪ D(v)
15             let (w, u) = arg max_{(w,u)∈T} A(u) + w
16             if A(u) > 0 then
17                 D(u) = D(u) ∪ {e}
18     else if e = (v, k, u) is a cover-edge then
19         if A(u) = ⊥ then
20             A(u) = ∞; D(u) = {e}; W = W ∪ succ(u)
21         else if A(u) ≤ k < ∞ or A(u) < k == ∞ then
22             A(v) = 0
23             if A(v) was changed then
24                 W = W ∪ D(v)
25         else
26             D(u) = D(u) ∪ {e}
27 return A(v_0)
```

a) $e \in W$,
b) $e \in D(u)$ *and* $A(u) > k$, *or*
c) $A(v) = 0$.

These loop-invariants allow us to conclude the correctness of the local algorithm.

Theorem 4. *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 returned value* $A(v_0)$ *is the minimum pre fixed-point assignment of* v_0.

We note that the termination argument is not completely straightforward as there is not a guarantee that it terminates within a polynomial number of

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 SDG from Figure 4(a)

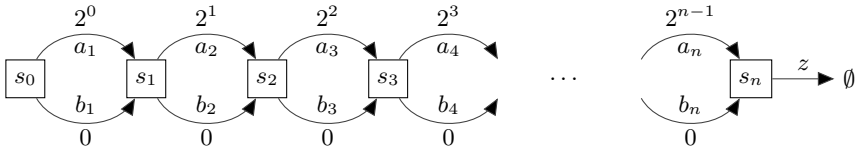


Fig. 5. A SDG where the local algorithm can take exponential running time

steps as depicted on the SDG in Figure 5 where for technical convenience, we named the hyper-edges $a_1, \ldots, a_n$, $b_1, \ldots, b_n$ and z. Consider now an execution of Algorithm 2 starting from the configuration s_0. Let us pick the edges from W at line 4 according to the strategy:

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

Then the initial assignment of $A(s_0) = \infty$ is gradually improved to $2^n - 1$, $2^n - 2$, $2^n - 3$, $\ldots 1$, 0. Hence, in the worst case, the local algorithm can perform exponentially many steps before it terminates, whereas the global algorithm always terminates in polynomial time. However, as we will see in Section 7, the local algorithm is in practice performing significantly better despite its high (theoretical) complexity.

6 Model Checking with Symbolic Dependency Graphs

We are now ready to present an encoding of a WKS and a WCTL formula as a symbolic dependency graph and hence decide the model checking problem via the computation of the minimum pre fixed-point assignment.

Given a WKS $\mathcal{K}$, a state s of $\mathcal{K}$ and a WCTL formula φ, we construct the corresponding symbolic dependency graph as before with the exception that the existential and universal "until" operators are encoded by the rules given in Figure 6.

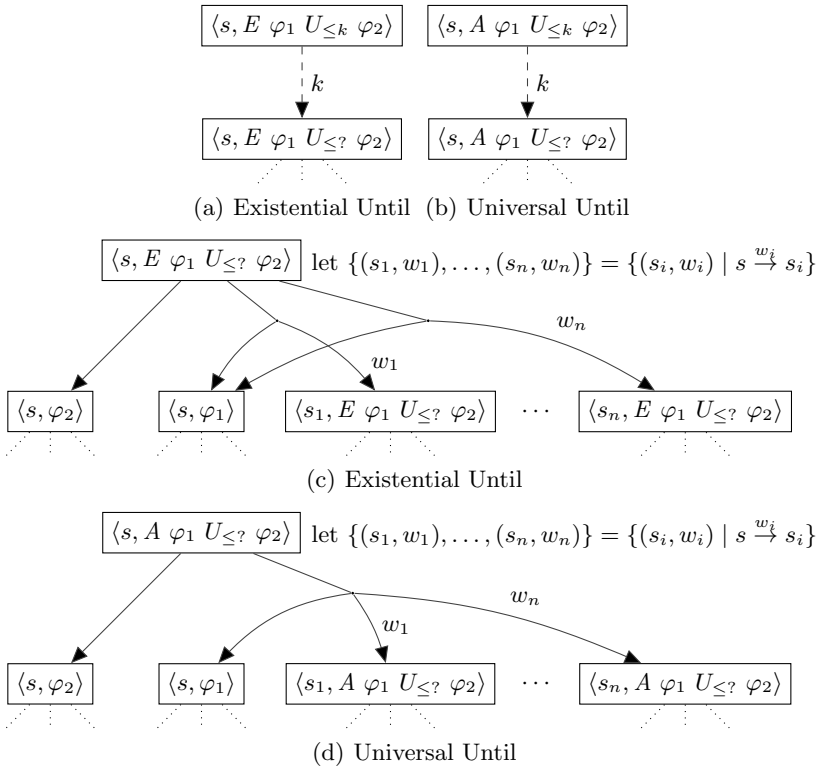


Fig. 6. SDG encoding of existential and universal 'until' formulas

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

Proof. By structural induction on φ. □

In Figure 7 we depict the symbolic dependency graph encoding of $E\ a\ U_{\leq 1000}\ b$ for the configuration s in the single-state WKS from Figure 3. This clearly illustrates the succinctness of SDG compared to standard dependency graphs. The minimum pre fixed-point assignment of this symbolic dependency graph is now reached in two iterations of the function F defined in Equation (1).

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

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

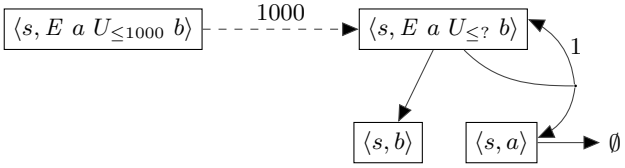


Fig. 7. SDG for the formula $s \models E\ a\ U_{\leq 1000}\ b$ and the WKS from Figure 3

As we already explained, the local model checking approach in Algorithm 2 may exhibit exponential running time. Nevertheless, the experiments in the section to follow show that this unlikely to happen in practice.

7 Experiments

In order to compare the performance of the algorithms for model checking WCTL, we developed a prototype tool implementation. There is a web-based front-end written in CoffeeScript available at

`http://wktool-spin2013.jonasfj.dk`

and the tool is entirely browser-based, requiring no installation. The model checking algorithms run with limited memory resources but the tool allows a fair comparison of the performance for the different algorithms. All experiments were conducted on a standard laptop (Intel Core i7) running Ubuntu Linux.

In order to experiment with larger, scalable models consisting of parallel components, we extend the process algebra CCS [17] with weight prefixing as well as proposition annotations and carry out experiments with weighted models of Leader Election [12], Alternating Bit Protocol [5], and Task Graph Scheduling problems for two processors [13]. The weight (communication cost) is associated with sending messages in the first two models while in the task graph scheduling the weight represents clock ticks of the processors.

7.1 Dependency Graphs vs. Symbolic Dependency Graphs

In Table 2 we compare the direct (standard dependency graph) algorithms with the symbolic ones. The execution times are in seconds and OOM indicates that verification runs out of memory. For a fixed size of the problems, we scale the bound k in the WCTL formulae. In the leader election protocol with eight processes, we verified a satisfiable formula $E\ \mathbf{true}\ U_{\leq k}\ \mathit{leader}$, asking if a leader can be determined within k message exchanges, and an unsatisfiable formula $E\ \mathbf{true}\ U_{\leq k}\ \mathit{leader} > 1$, asking if there can be more than one leader selected within k message exchanges. For the alternating bit protocol with a communication buffer of size four, we verified a satisfied formula $E\ \mathbf{true}\ U_{\leq k}\ \mathit{delivered} = 1$, asking if a message can be delivered within k communication steps, and an unsatisfied formula $E\ \mathbf{true}\ U_{\leq k}\ (s_0 \wedge d_1) \vee (s_1 \wedge d_0)$, asking whether the sender and receiver can get out of synchrony withing the first k communication steps.

Leader Election					
	Direct		Symbolic		
k	Global	Local	Global	Local	
200	3.88	0.23	0.26	0.02	Satisfied
400	8.33	0.25	0.26	0.02	
600	OOM	0.24	0.26	0.02	
800	OOM	0.25	0.26	0.02	
1000	OOM	0.26	0.27	0.02	
200	7.76	8.58	0.26	0.26	Unsatisfied
400	17.05	20.23	0.26	0.26	
600	OOM	OOM	0.26	0.26	
800	OOM	OOM	0.26	0.26	
1000	OOM	OOM	0.26	0.26	

Alternating Bit Protocol					
	Direct		Symbolic		
k	Global	Local	Global	Local	
100	3.87	0.05	0.23	0.03	Satisfied
200	8.32	0.06	0.23	0.03	
300	OOM	0.10	0.28	0.04	
400	OOM	0.11	0.23	0.03	
500	OOM	0.13	0.23	0.03	
100	3.39	3.75	0.27	0.23	Unsatisfied
200	6.98	8.62	0.30	0.25	
300	OOM	15.37	0.28	0.24	
400	OOM	OOM	0.27	0.24	
500	OOM	OOM	0.27	0.22	

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

For the satisfied formula, the direct global algorithm (global fixed-point computation on dependency graphs) runs out of memory as the bound k in the formulae is scaled. The advantage of Liu and Smolka [16] local algorithm is obvious as on positive instances it performs (using DFS search strategy) about as well as the global symbolic algorithm. The local symbolic algorithm clearly performs best. We observed a similar behaviour also for other examples we tested and the symbolic algorithms were regularly performing better than the ones using the direct translation of WCTL formulae into dependency graphs. Hence we shall now focus on a more detailed comparison of the local vs. global symbolic algorithms.

7.2 Local vs. Global Model Checking on SDG

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

We also tested the algorithms on a larger benchmark of task graph scheduling problems [4]. The task graph scheduling problem asks about schedulability of a number of parallel tasks with given precedence constraints and processing

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

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

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

times that are executed on a fixed number of homogeneous processors [13]. We automatically generate models for two processors from the benchmark containing in total 180 models and scaled them by the number of initial tasks that we include from each case into schedulability analysis.

The first three task graphs (T0, T1 and T2) are presented in Table 4. We model check nested formulae and the satisfiable one is $E\ \mathbf{true}\ U_{\leq 90}\ (t_{n-2}^{ready} \wedge A\ \mathbf{true}\ U_{\leq 80}\ \mathit{done})$ asking whether there is within 500 clock ticks a configuration where the task t_{n-2} can be scheduled such that then we have a guarantee that the whole schedule terminates within 500 ticks. When the upper-bounds are decreased to 5 and 10 the formula becomes unsatisfiable for all task graphs in the benchmark.

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

8 Conclusion

We suggested a symbolic extension of dependency graphs in order to verify negation-free weighted CTL properties where temporal operators are annotated with upper-bound constraints on the accumulated weight. Then we introduced global and local algorithms for the computation of fixed-points in order to answer

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

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

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)	50.4	12.9	47.6	2.30	47.32	0.44

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

the model checking problems for the logic. The algorithms were implemented and experimented with, coming to the conclusion that the local symbol algorithm is the preferred one, providing order of magnitude speedup in the cases where the bounds in the logical formula allow for a larger number of possible witnesses of satisfiability of the formula.

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

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. Patricia Bouyer, Kim Guldstrand Larsen, and Nicolas Markey. Model checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4(2), 2008.
9. 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.
10. Peter Buchholz and Peter Kemper. Model checking for a class of weighted automata. *Discrete Event Dynamic Systems*, 20:103–137, 2010.
11. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *IN CONCUR 05, LNCS 3653*, pages 66–80. Springer, 2005.
12. 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.
13. 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.
14. 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.
15. 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.
16. Xinxin Liu and Scott A. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In *ICALP*, pages 53–66, 1998.
17. R. Milner. A calculus of communicating systems. *LNCS*, 92, 1980.