Model Checking and Synthesis for Branching Multi-Weighted Logics

Jensen, J.S.; Kaufmann, Isabella; Larsen, Kim Guldstrand; Nielsen, S.M.; Srba, Jiri

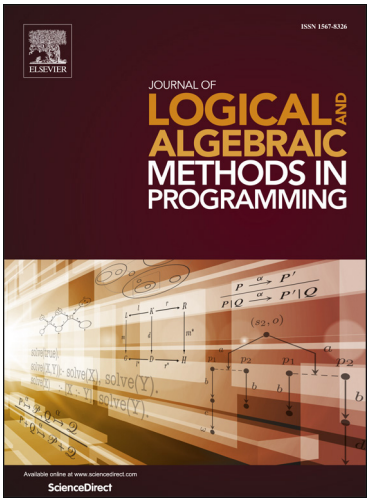
Published in:
Journal of Logic and Algebraic Programming

DOI (link to publication from Publisher):
10.1016/j.jlamp.2019.02.001

Publication date:
2019

Document Version
Accepted author manuscript, peer reviewed version

Accepted Manuscript



Model checking and synthesis for branching multi-weighted logics

L.S. Jensen, I. Kaufmann, K.G. Larsen, S.M. Nielsen, J. Srba

PII: S2352-2208(18)30033-6
DOI: https://doi.org/10.1016/j.jlamp.2019.02.001
Reference: JLAMP 441

To appear in: *Journal of Logical and Algebraic Methods in Programming*

Received date: 2 March 2018
Revised date: 30 December 2018
Accepted date: 3 February 2019

Please cite this article in press as: L.S. Jensen et al., Model checking and synthesis for branching multi-weighted logics, *J. Log. Algebraic Methods Program.* (2019), https://doi.org/10.1016/j.jlamp.2019.02.001

Model checking and synthesis for branching multi-weighted logics

L.S. Jensen[a], I. Kaufmann[a,*], K.G. Larsen[a], S.M. Nielsen[a], J. Srba[a,1]

[a]*Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, DK-9220 Aalborg East, Denmark*

Abstract

We investigate the open synthesis problem in a quantitative game theoretic setting where the system model is annotated with multiple nonnegative weights representing quantitative resources such as energy, discrete time or cost. We consider system specifications expressed in the branching time logic CTL extended with bounds on resources. As our first contribution, we show that the model checking problem for the full logic is undecidable with already three weights. By restricting the bounds to constant upper or lower-bounds on the individual weights, we demonstrate that the problem becomes decidable and that the model checking problem is PSPACE-complete. As a second contribution, we show that by imposing upper-bounds on the temporal operators and assuming that the cost converges over infinite runs, the synthesis problem is also decidable. Finally, we provide an on-the-fly algorithm for the synthesis problem on an unrestricted model for a reachability fragment of the logic and we prove EXPTIME-completeness of the synthesis problem.

Keywords: synthesis, model checking, quantitative, temporal logics, dependency graphs

1. Introduction

Complex systems are an integral part of everyday life and the correctness of these systems is an area of great interest. For several safety-critical application areas the cost of unexpected behaviour in the system can be very high, thus creating the demand for a more thorough verification. One way of achieving increased assurance of correctness is by using formal methods such as model checking [5, 19] and more recently synthesis [16].

While model checking can verify whether an implementation (or model thereof) satisfies a given specification, synthesis is the effort to automatically

*Corresponding author

Email addresses: (L.S. Jensen), Kaufmann@cs.aau.dk (I. Kaufmann), kgl@cs.aau.dk (K.G. Larsen), (S.M. Nielsen), srba@cs.aau.dk (J. Srba)

[1]Partially affiliated with FI MU, Brno.

generate a satisfying implementation directly from a specification. This implementation can either be thought of as an isolated system where we only consider the specification, or an open system where we have to interact with an environment [8].

In the setting of an open system, the synthesis problem consists in constructing a control program P that together with the environment E satisfies the given specification [18]. While the resulting system is deterministic, it creates a computation tree. The branching corresponds to external nondeterminism, caused by the uncontrollable environment. For a given linear time specification we must verify that it holds in all paths of the resulting computation tree. However, in order to express possibility properties, one needs to use branching temporal logics, which enable both universal and existential path quantification.

In this paper, we deal with the (open) synthesis problem in the setting where both the behaviour of the system contains quantitative aspects (e.g. energy, time, resource usage, etc.) and the specification may (or may not) contain hard constraints on these. To this end, we consider the synthesis problem in a branching setting, where specifications may simultaneously pose requirements both on possibilities and necessities. Such specifications are not expressible in a linear time view.

1.1. Motivating Example

To motivate the investigation of model checking and synthesis in a multi-weighted setting, we shall introduce a small example. Consider Figure 1 which models an investment scheme. In this model the investor begins in the initial state s_0. She can then move to the state s_1 from which the choices of making some investment or liquidating all assets are possible. If the investor wants to make a low risk investment she follows the transitions to the state s_2. The transition is annotated with the time in months it takes to evaluate such an investment. From this state the result of the investment is determined and the investor is back at s_1. A similar situation occurs when choosing to make a high risk investment. The last possibility is to liquidate, and thus ending this round of investments. In this model only one investment can be ongoing at any given time.

The transition between these states are depicted as arrows, each annotated with a vector. In this model the vector represent loss of funds, earnings and the passage of time in months. When the transition from one state to another is affected by something beyond our control, we depict it as a dashed arrow. In this example we model market influence in the transitions leading from a high or low risk investment, and show that this affect the loss or earnings of a particular investment.

Based on this model, we can now ask questions about the feasibility of certain objectives. An example can be: *Within a year, can we achieve earnings of at least 600 units, while never experiencing a loss of more than 30 units over a period of 6 months.* In general, the answer to this type of question depends on two things: the choices we make and the role the environment plays. The choices we make often reflect trade-offs. For example it is more time efficient

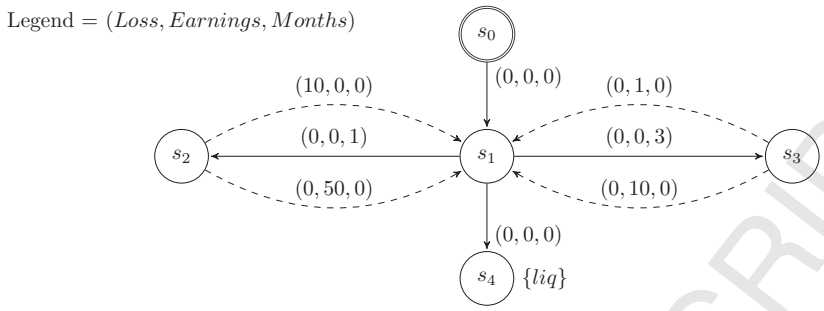


Figure 1: Model of investment scheme

and possibly more rewarding to go for a high risk investment, but it also carries the danger of suffering a loss of funds. The right choice to make then depends on the situation and the goal we wish to achieve. In this example the objective can only be met if we always go for a high risk investment and the environment never causes us to loose funds.

We will investigate problems of this nature. Specifically deciding whether an objective is realizable or not within given cost bounds, and in the situation where it is realizable, we want to determine a strategy to achieve the goal.

1.2. Our Contributions

We introduce a multi-weighted extension to Kripke structures and provide a multi-weighted extension of Computation Tree Logic (CTL). We initially prove that the model checking problem is undecidable on a finite structure with three weights. However, by restricting the form of the bounds we find a decidable subset and show that the complexity of model checking is PSPACE-complete. We then extend the multi-weighted formalism to a game theoretic setting and formally define the synthesis problem. We show that for an upper-bounded specification and a cost-convergent game graph, the synthesis problem is decidable. Finally we investigate the synthesis problem for a fragment of the logic which focuses on reachability. We provide an on-the-fly algorithm by reducing the synthesis problem to the problem of calculating the fixed-point assignment of a dependency graph and show that the synthesis problem for the reachability subset is EXPTIME-complete.

1.3. Related Work

In a recent paper [14] Larsen et al. studies a multi-weighted extension of the alternation-free μ-calculus and show that the satisfiability problem is decidable for the class of weighted transition systems extended with nonnegative reals on the transitions. However no complexity characterization is given. While we consider a subset of the logic presented in [14] (the weighted CTL subset) we provide tight complexity results for the model checking problem for this subset.

When considering model checking for quantitative formalisms, we find some related results for models with a single weight. A discrete time extension of CTL with the possibility to set bounds on clocks and with the possibility to reset clocks is studied in [13]. It is shown that in their case the reset operator causes the complexity to go from PTIME to PSPACE. In our work we consider a weighted extension of CTL with a syntax close to the one presented in [13], however while they consider a discrete timed extension of Kripke structures we consider a multi-weighed extension. In [9] an on-the-fly algorithm was suggested and implemented for a weighed extension of CTL without negation and with upper-bounds on the temporal operators (e.g. $EX_{\leq k}\varphi$). In contrast to their work we do not have bounds attached to the temporal operators, instead they can be expressed and used as propositions. Additionally we consider multiple weights.

For the model checking problem of temporal logics in multi-weighted settings, effort has been made to identify the largest possible decidable logic fragments. In [2] temporal specifications on quantitative Kripke structures with both negative and nonnegative weights are considered. They show that the model checking problem for a fragment of CTL without the EG and EU operator but with bounds on the cost is decidable and 2NEXPTIME-hard. It is easy to see that allowing the EG and EU operators while having both negative and positive weights on transitions would result in undecidability. We limit ourselves to nonnegative weights and are thus able to represent the full CTL while still providing a decidability result. Furthermore we show that the complexity for the model checking problem is then PSPACE-complete.

In a multi-weighted setting, formalisms like energy games are often considered. While originally presented in a real-time setting for a single clock [3] it was later considered in a discrete multi-weighted setting [10, 12, 7, 4]. While we only work with nonnegative weights, we consider more expressive branching time winning conditions where we are able to specify multiple lower-and upper-bounds on the accumulated cost at any point of the computation. Furthermore we consider reset operators which can locally reset the cost of a number of specified weights for a subformula.

For the problem of synthesis for multi-weighted formalisms less work has been done. In [1] Almagor et al. look at LTL model checking and synthesis where they measure the quantitative aspects. In contrast our approach is based on CTL and focus on branching time properties.

2. Preliminaries

We present the n-Weighted Kripke Structure (n-WKS), and we write $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_\infty = \mathbb{N}_0 \cup \infty$.

Definition 2.1 (n-Weighted Kripke Structure) An n-Weighted Kripke Structure (n-WKS) is a tuple $K = (S, s_0, \mathcal{AP}, L, T)$ where:

- S is a set of states,
- $s_0 \in S$ is the initial state,

- $\mathcal{AP}$ is a finite set of atomic propositions,
- $L : S \to \mathcal{P}(\mathcal{AP})$ is a labelling function, and
- $T \subseteq S \times \mathbb{N}_0^n \times S$ is a transition relation, with a weight vector of n dimensions.

When $(s, \bar{c}, s') \in T$, where $s, s' \in S$ and $\bar{c} \in \mathbb{N}_0^n$ is a vector, then we write $s \xrightarrow{\bar{c}} s'$. When s' is reachable from s, by at least one transition, we write $s \to^+ s'$ and when s has no outgoing transitions we write $s \not\to$.

An n-WKS $K = (S, s_0, \mathcal{AP}, L, T)$ is *finite* whenever S is a finite set of states and T is a finite transition relation.

Let $\overline{w} \in \mathbb{N}_0^n$ then we denote the ith component of $\overline{w}$ by $\overline{w}[i]$, where $1 \leq i \leq n$. To set the ith component of $\overline{w}$ to a specific value $k \in \mathbb{N}_0$ we write $\overline{w}[i \to k]$ and to set multiple components $I \subseteq \{0, \ldots, n\}$ to a specific value $k \in \mathbb{N}_0$ we write $\overline{w}[I \to k]$.

Definition 2.2 (Ordering on Vectors) Let $\overline{w} = (\overline{w}[1], \ldots, \overline{w}[n]) \in \mathbb{N}_0^n$ and $\overline{w}' = (\overline{w}'[1], \ldots, \overline{w}'[n]) \in \mathbb{N}_0^n$. We write $\overline{w} \leq \overline{w}'$ iff $\overline{w}[i] \leq \overline{w}'[i]$ for all $1 \leq i \leq n$.

We define a run ρ in the n-WKS K to be an infinite or finite sequence of states and transitions:

$$\rho = s_1 \xrightarrow{\bar{c}_1} s_2 \xrightarrow{\bar{c}_2} s_3 \xrightarrow{\bar{c}_3} \ldots$$

where $s_i \xrightarrow{\bar{c}_i} s_{i+1}$ for all $i \geq 1$. Given a position $i \in \mathbb{N}$ along ρ, let $\rho(i) = s_i$, and $\textsc{Last}(\rho)$ be the state at last position along ρ, if ρ is finite. We also define the concatenation operator $\circ$, s.t. if $(\rho = s_1 \xrightarrow{\bar{c}_1} s_2 \xrightarrow{\bar{c}_2} \ldots \xrightarrow{\bar{c}_{n-1}} s_n)$ then $\rho \circ (s_n \xrightarrow{\bar{c}_n} s_{n+1}) = (s_1 \xrightarrow{\bar{c}_1} s_2 \xrightarrow{\bar{c}_2} s_3 \ldots s_n \xrightarrow{\bar{c}_n} s_{n+1})$. We denote the set of all runs ρ in the n-WKS K of the form $(\rho = s_1 \xrightarrow{\bar{c}_1} s_2 \xrightarrow{\bar{c}_2} \ldots)$ as Π_K. Furthermore we denote the set of all finite runs ρ in the n-WKS K of the form $(\rho = s_1 \xrightarrow{\bar{c}_1} \ldots \xrightarrow{\bar{c}_{n-1}} s_n)$ as Π_K^{fin}. Lastly, we define Π_K^{Max} as the set of all runs ρ s.t. ρ is infinite or $\textsc{Last}(\rho)$ is in a deadlock i.e. $\textsc{Last}(\rho) \not\to$.

Definition 2.3 (Cost) Let $K = (S, s_0, \mathcal{AP}, L, T)$ be an n-WKS and $(\rho = s_1 \xrightarrow{\bar{c}_1} s_2 \xrightarrow{\bar{c}_2} \ldots)$ be a run in K. The cost of ρ, at position $i \in \mathbb{N}$, is then defined as:

$$cost_\rho(i) = \begin{cases} 0^n & \text{if } i = 1 \\ \sum_{j=1}^{i-1} \bar{c}_j & \text{otherwise.} \end{cases}$$

If ρ is finite, we denote $cost(\rho)$ as the cost of the last position along ρ.

To gain familiarity with the notation, Example 1 presents the formal definition of the informal example presented in Section 1.1.

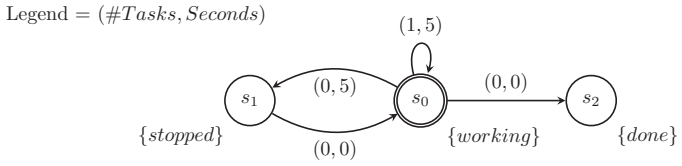


Figure 2: n-WKS illustrating working on a set of tasks

Example 1 (Notation around the *n*-WKS)
Consider the n-WKS presented in Figure 2, which illustrates a process completing a set of tasks. The loop transition $s_0 \xrightarrow{(1,5)} s_0$ *represents the completion of a single task, while the transition* $s_0 \xrightarrow{(0,5)} s_1$ *represents stopping the process.*

An example of a finite run $\rho \in \Pi_K^{fin}$ *could be the following*

$$\rho = (s_0 \xrightarrow{(1,5)} s_0 \xrightarrow{(0,5)} s_1 \xrightarrow{(0,0)} s_0 \xrightarrow{(1,5)} s_0 \xrightarrow{(0,0)} s_2)$$

where $cost_\rho(2) = (1, 10)$ *and* $cost_\rho(5) = cost(\rho) = (2, 15)$*. From the total cost of the run we see that two tasks have been completed and that 15 seconds have gone by.*

2.1. Logic

Based on this weighted extension of Kripke structures we define an extension of CTL which includes bounds specified as properties.

Definition 2.4 (Weighted Computation Tree Logic) We define a weighted extension of CTL, Weighted Computation Tree Logic (WCTL), in relation to an n-WKS $K = (S, s_0, \mathcal{AP}, L, T)$ s.t.

$$\varphi ::= \textsc{true} \mid \textsc{false} \mid a \mid e_1 \bowtie e_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid \varphi_1 \Leftrightarrow \varphi_2 \mid$$
$$AX\varphi \mid EX\varphi \mid AG\varphi \mid EG\varphi \mid AF\varphi \mid EF\varphi \mid E\varphi_1 U \varphi_2 \mid A\varphi_1 U \varphi_2 \mid \textit{reset } \#i \text{ in } \varphi$$
$$e ::= c \mid \#i \mid e_1 \oplus e_2$$

where $a \in \mathcal{AP}$, $\bowtie \in \{<, \leq, =, \geq, >\}$, $\oplus \in \{+, -, \cdot\}$, $c \in \mathbb{N}_0$, and $1 \leq i \leq n$ is a component index of a vector.

Intuitively the notation $\#i$ then refers to the cost of the i-th component. Given a WCTL formula φ, we define the set of all subformulae in φ as $Sub(\varphi)$.

We define the semantics for a minimal set of operators. Let $s \in S$ be a state and $\overline{w} \in \mathbb{N}_0^n$. We then write $K, s \vDash_{\overline{w}} \varphi$ when K satisfies the formula φ in the state s with the cost $\overline{w}$.

$K, s \vDash_{\overline{w}} \text{TRUE}$	
$K, s \vDash_{\overline{w}} a$	iff $a \in L(s)$
$K, s \vDash_{\overline{w}} \neg\varphi$	iff $s \nvDash_{\overline{w}} \varphi$
$K, s \vDash_{\overline{w}} \varphi_1 \vee \varphi_2$	iff $s \vDash_{\overline{w}} \varphi_1$ or $s \vDash_{\overline{w}} \varphi_2$
$K, s \vDash_{\overline{w}} E\varphi_1 U \varphi_2$	iff there exists $(\rho = s_1 \xrightarrow{\bar{c}_1} s_2 \dots) \in \Pi_K^{Max}$ where $s = s_1$ and a position $i \geq 1$ such that $K, \rho(i) \vDash_{\overline{w}+cost_\rho(i)} \varphi_2$ and $K, \rho(j) \vDash_{\overline{w}+cost_\rho(j)} \varphi_1$ for all $j < i$
$K, s \vDash_{\overline{w}} A\varphi_1 U \varphi_2$	iff for all $(\rho = s_1 \xrightarrow{\bar{c}_1} s_2 \dots) \in \Pi_K^{Max}$ where $s = s_1$ there is a position $i \geq 1$ such that $K, \rho(i) \vDash_{\overline{w}+cost_\rho(i)} \varphi_2$ and $K, \rho(j) \vDash_{\overline{w}+cost_\rho(j)} \varphi_1$ for all $j < i$
$K, s \vDash_{\overline{w}} EX(\varphi)$	iff there is a state s' such that $s \xrightarrow{\bar{c}} s'$, and $K, s' \vDash_{\overline{w}+\bar{c}} \varphi$
$K, s \vDash_{\overline{w}} \textit{reset \#I in } \varphi$	iff $K, s \vDash_{\overline{w}[I \to 0]} \varphi$
$K, s \vDash_{\overline{w}} e_1 \bowtie e_2$	iff $eval_{\overline{w}}(e_1) \bowtie eval_{\overline{w}}(e_2)$

The evaluation of e is:

$$
\begin{aligned}
eval_{\overline{w}}(c) &= c \\
eval_{\overline{w}}(\#i) &= \overline{w}[i] \\
eval_{\overline{w}}(e_1 \oplus e_2) &= eval_{\overline{w}}(e_1) \oplus eval_{\overline{w}}(e_2)
\end{aligned}
$$

The remaining operators, from the syntax, can be derived from the minimal set, and likewise can their semantics. The derived operators are defined as:

$$
\begin{aligned}
AF\ \varphi &\equiv A(\text{TRUE})U(\varphi) & EF\ \varphi &\equiv E(\text{TRUE})U(\varphi) \\
AG\ \varphi &\equiv \neg EF \neg\varphi & EG\ \varphi &\equiv \neg AF \neg\varphi \\
AX(\varphi) &\equiv \neg EX(\neg\varphi) & \varphi_1 \wedge \varphi_2 &\equiv \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\varphi_1 \Rightarrow \varphi_2 &\equiv \neg\varphi_1 \vee \varphi_2 & \varphi_1 \Leftrightarrow \varphi_2 &\equiv (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)
\end{aligned}
$$

Remark 1
Another minimal set of operators can be found by replacing $A\varphi_1 U \varphi_2$ with $EG\varphi$ as $A\varphi_1 U \varphi_2 \equiv \neg(EG(\neg\varphi_2) \vee E\neg\varphi_2 U(\neg(\varphi_1 \vee \varphi_2))))$. This set will be used Theorem 4.

Given an n-WKS $K = (S, s_0, \mathcal{AP}, L, T)$ and a WCTL formula φ the model checking problem is the question of whether $K, s \vDash_{\overline{w}} \varphi$ where $s \in S$ and $\overline{w} \in \mathbb{N}_0^n$. When the n-WKS K is obvious from context and the vector $\overline{w}$ is 0^n we simply write $s \vDash \varphi$.

Example 2 (Reset example)
Consider again the n-WKS presented in Example 1. To demonstrate the reset syntax we consider the following query: "Can we construct a schedule where 10 tasks are completed within 60 seconds while guaranteeing that we can always stop the process

within 5 seconds". To concisely express this we construct two subformula. First we specify that we are either working and able to stop within 5 seconds or are stopped and can return to work,

$$\varphi_1 = (working \wedge reset\ \#2\ in\ (EF(stopped \wedge \#2 \leq 5))) \vee (stopped \wedge EF(working)).$$

The reset timing component ensure that the bound is not affected by time spend on any previous tasks.

Second we expresses the required final state, where all tasks are done (indicated by the proposition $\{done\}$ and the bound on component one) and the total time is within 60 seconds.

$$\varphi_2 = (\{done\} \wedge \#1 = 10 \wedge \#2 \leq 60).$$

By combining these into the existential until $E\varphi_1 U \varphi_2$ we can express the full query. We find that the query is satisfied, and an example of a satisfying run can be found by repeating the cycle $s_0 \xrightarrow{(1,5)} s_0$ 10 times and then taking the final transition $s_0 \xrightarrow{(0,0)} s_2$. By using the reset operator we also know that the process could have been stopped while it was working within the time limit.

$$s_0 \xrightarrow{(1,5)} s_0 \xrightarrow{(1,5)} s_0 \xrightarrow{(1,5)} \dots \xrightarrow{(1,5)} s_0 \xrightarrow{(0,0)} s_2$$

3. Model checking

In this section we show that the model checking problem for WCTL is undecidable on a finite n-WKS. We then modify the logic and prove it to be PSPACE-complete.

3.1. Undecidability of WCTL

We begin by looking at decidability of the model checking problem for WCTL. First we show that the model checking problem for WCTL is undecidable on a finite 3-WKS.

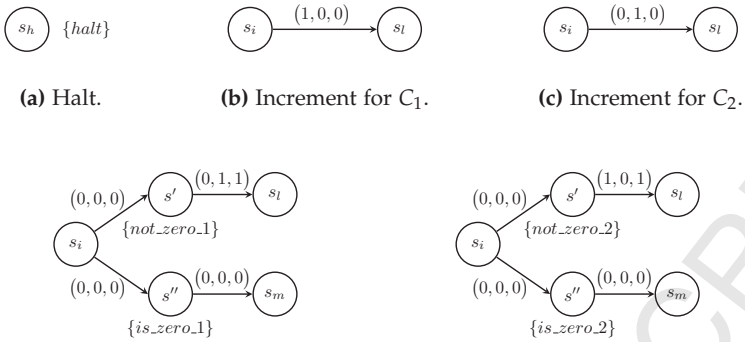
Theorem 1 (Undecidability of WCTL)
The model checking problem for WCTL is undecidable on a finite 3-WKS.

Proof. We use reduction from the halting problem for two-counter Minsky machines [17]. Let M be a two-counter-machine (2CM) with two nonnegative counters C_1 and C_2 and a finite set of instructions where each instruction Ins_i is either

- e: HALT
- Increment i: $C_j := C_j + 1$; GOTO(l)
- Decrement i: If $C_j > 0$ then $(C_j := C_j - 1$; GOTO(l)) else GOTO(m)

where $j \in (1, 2)$ and $1 \leq l, m \leq e$. To simulate the machine, let K be a finite 3-WKS where whenever K is in state s_i then M is in Ins_i. We use the vector of length three to increase and decrease the value of the counters, by the cost of a run $\rho \in \Pi_K^{fin}$ s.t.

$$C_1 = \text{COST}_\rho(i)[1] - \text{COST}_\rho(i)[3],$$



(a) Halt. **(b)** Increment for C_1. **(c)** Increment for C_2.

(d) Decrement for C_1. **(e)** Decrement for C_2.

Figure 3: 3-WKS simulation of a 2CM

$$C_2 = \textsc{Cost}_\rho(i)[2] - \textsc{Cost}_\rho(i)[3],$$

where $0 \leq i$ is the position of s_i in the run ρ. In the simulation, the instructions are translated as seen in Figure 3 s.t. HALT is illustrated in Figure 3a, Increment in Figure 3b and 3c and Decrement in Figure 3d and Figure 3e.

We then construct the formula,

$$\varphi = E(A \wedge B)U(halt)$$

where

$$A := ((not\_zero\_1) \Rightarrow \#1 > \#3) \wedge ((not\_zero\_2) \Rightarrow \#2 > \#3)$$
$$B := ((is\_zero\_1) \Rightarrow \#1 = \#3) \wedge ((is\_zero\_2) \Rightarrow \#2 = \#3)$$

We now want to show that M will halt for the empty input $(C_1 = 0, C_2 = 0)$ iff $s_0 \vDash \varphi$.

- $\Rightarrow$ If M halts, then $K, s_0 \vDash_{0^n} \varphi$. We know that when running M the n-WKS K can simulate the exact same instructions, so that if M will halt, then a state s_h is reached in K by a run ρ s.t. $\textsc{Last}(\rho) = s_h$ and for all $0 \leq i$ s.t. $\rho(i) \neq \textsc{Last}(\rho)$ it holds that $K, \rho(i) \vDash A \wedge B$, hence if M halts, then $K, s_0 \vDash_{0^n} \varphi$.
- $\Leftarrow$ If $K, s_0 \vDash_{0^n} \varphi$ then M will halt. The formula φ ensures that M is simulated faithfully, as the counters are calculated in φ. When encountering the decrement rules, φ enforces that the correct choice is taken, as the next state of the path will never satisfy both pre-conditions A and B if it is not allowed in M. To simulate the empty input, the weight vector $\overline{w} = 0^n$. As M is faithfully simulated, we have that if $K, s_0 \vDash_{0^n} \varphi$ then M will halt.

Hence M will halt for the empty input $(C_1 = 0, C_2 = 0)$ iff $s_0 \vDash \varphi$ and therefore we can conclude that WCTL is undecidable on a finite 3-WKS. $\square$

We note that this result holds for a sublogic which only uses existential until (EU) and either subtraction of vector components and comparison with constants ($\#i - \#j \bowtie c$) or simply comparison of components ($\#i \bowtie \#j$).

3.2. Decidability of Constant Bound WCTL

As comparison of vector components and subtraction plays a key part in encoding a two-counter Minsky machine, we remove these from the logic. These changes result in the following logic where every bound is specified with a nonnegative constant.

Definition 3.1 (Constant Bound WCTL (cb-WCTL)) We define cb-WCTL as WCTL where bounds are now of the form $(e \bowtie c)$ where

$$e := c \mid \#i \mid e_1 \oplus e_2$$

and $\bowtie \in \{<, \leq, =, \geq, >\}$, $\oplus \in \{+, \cdot\}$, $c \in \mathbb{N}_0$, and $1 \leq i \leq n$ is a component index in a vector.

We observe that as the cost of a run is nondecreasing and bounds are specified by some positive constant, then at some point the cost of a run no longer affects the satisfiability of a cb-WCTL formula.

Definition 3.2 (Boundary vector) Let K be an n-WKS and φ be a cb-WCTL formula. Recall that $Sub(\varphi)$ is the set of all subformula in φ. Then given the largest bound $gb = \max\{c \mid (e \bowtie c) \in Sub(\varphi)\}$ we define the boundary vector of φ as $\overline{b} = gb^n$.

We say the boundary $\overline{b}$ is *derived from* φ. With this we can define a function used to limit the amount of vectors represented in K, allowing us to finitely unfold K. We call this function *cut* and define it as:

Definition 3.3 (cut) Given an n-WKS K a cb-WCTL formula φ and let $\overline{b}$ be the bound derived from φ. We then define the function $cut : \mathbb{N}_0^n \to \mathbb{N}_0^n$ s.t. for all $1 \leq i \leq n$:

$$cut(\overline{w})[i] = \begin{cases} \overline{w}[i] & \text{if } \overline{w}[i] \leq \overline{b}[i] \\ \overline{b}[i] + 1 & \text{otherwise.} \end{cases}$$

Observe that given a vector $\overline{w} \in \mathbb{N}_0^n$, it holds that $eval_{\overline{w}}(e) \geq eval_{cut(\overline{w})}(e)$.

Lemma 2
Let $K = (S, s_0, \mathcal{AP}, L, T)$ be an n-WKS and let φ be a cb-WCTL formula. Then given a state $s \in S$ and a vector $\overline{w} \in \mathbb{N}_0^n$, it holds that $K, s \vDash_{\overline{w}} \varphi$ iff $K, s \vDash_{cut(\overline{w})} \varphi$.

Proof. The proof goes by structural induction on φ. First we show that if $K, s \vDash_{\overline{w}} \varphi$ then $K, s \vDash_{cut(\overline{w})} \varphi$ for $\varphi = e \bowtie c$. All remaining cases follow trivially as no other operators depend on the cost.

$\varphi = e \leq c$: Then $eval_{\overline{w}}(e) \leq c$ and as $eval_{\overline{w}}(e) \geq eval_{cut(\overline{w})}(e)$ we have that $eval_{cut(\overline{w})}(e) \leq eval_{\overline{w}}(e) \leq c$.

$\varphi = e \geq c$: This case goes by structural induction on the expression e. If e is a constant c' then $eval_{\overline{w}}(c') = eval_{cut(\overline{w})}(c') \geq c$ as the cost of the run does not affect the evaluation of a constant. If $e = \#i$ then either $eval_{\overline{w}}(\#i) \geq eval_{cut(\overline{w})}(\#i) = \overline{b}[i] + 1 \geq c$ or $eval_{\overline{w}}(\#i) = eval_{cut(\overline{w})}(\#i) \geq c$. Otherwise $e = e_1 \oplus e_2$ and if $eval_{\overline{w}}(e_1 \oplus e_2) \geq c$ then $eval_w(e_1) \geq c_1$ and $eval_w(e_2) \geq c_2$ s.t. $c_1 \oplus c_2 = c$. By induction hypothesis then $eval_{cut(\overline{w})}(e_i) \geq c_i$ for $i \in \{1, 2\}$ and thus $eval_{cut(\overline{w})}(e_1 \oplus e_2) \geq (c_1 \oplus c_2)$.

Second we show that if $K, s \vDash_{cut(\overline{w})} \varphi$ then $K, s \vDash_{\overline{w}} \varphi$ for $\varphi = e \bowtie c$. Again, all remaining cases follow trivially as no other operators depend on the cost.

$\varphi = e \leq c$: This case goes by structural induction on the expression e. If e is a constant c' then again we have that $eval_{cut(\overline{w})}(c') = eval_{\overline{w}}(c') \leq c$. If $e = \#i$ then $eval_{cut(\overline{w})}(\#i) = eval_{\overline{w}}(\#i) \leq c \leq \overline{b}[i] + 1$. Otherwise $e = e_1 \oplus e_2$ and if $eval_{cut(\overline{w})}(e_1 \oplus e_2)$ then $eval_{cut(w)}(e_1) \leq c_1$ and $eval_{cut(w)}(e_2) \leq c_2$ s.t. $c_1 \oplus c_2 = c$. By induction hypothesis then $eval_{\overline{w}}(e_i) \leq c_i$ for $i \in \{1, 2\}$ and thus $eval_{\overline{w}}(e_1 \oplus e_2) \leq (c_1 \oplus c_2)$.

$\varphi = e \geq c$: Then $eval_{cut(\overline{w})}(e) \geq c$ and as $eval_{\overline{w}}(e) \geq eval_{cut(\overline{w})}(e)$ we have that $eval_{\overline{w}}(e) \geq eval_{cut(\overline{w})}(e) \geq c$.

Thus we can conclude that $K, s \vDash_{\overline{w}} \varphi$ iff $K, s \vDash_{cut(\overline{w})} \varphi$. □

Lemma 3
The model checking problem for cb-WCTL on a finite n-WKS is PSPACE-hard.

Proof. We show this by reduction from the totally quantified boolean formula (TQBF) problem, which is PSPACE-complete [21]. The TQBF problem is to decide whether a fully quantified formula Φ is true or false. Given the formula Φ ranging over n boolean variables, we construct a n-WKS and encode the variables as weights s.t. the value of x_i is determined by the value of the i-th component $\#i$. Given a formula with 3 variables we construct the n-WKS illustrated in Figure 4.

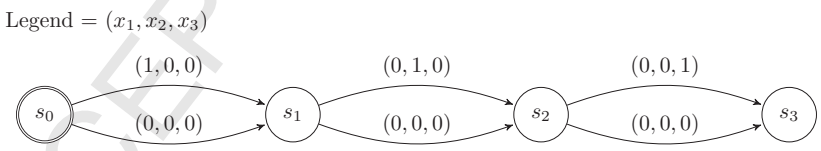


Figure 4: Encoding of variable assignment

We encode the quantifiers s.t. $\exists$ is encoded as EX and $\forall$ as AX. Furthermore the formula is encoded as bounds where x_i becomes $\#i \geq 1$ and $\neg x_i$ becomes $\#i \leq 0$. To illustrate consider the following formula:

$$\Phi = \exists x_1 \forall x_2 \exists x_3 (x_1 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_1).$$

This translate to the corresponding cb-WCTL formula:

$$\varphi = EX(AX(EX((\#1 \geq 1 \vee \#3 \geq 1) \wedge (\#1 \leq 0 \vee \#2 \geq 1) \wedge (\#3 \leq 0 \vee \#1 \leq 0))))$$

Clearly we have that there exist an assignment of variables s.t. Φ is true iff $K, s_0 \vDash_{0^n} \varphi$. □

Theorem 4
The model checking problem for cb-WCTL on a finite n-WKS is PSPACE-complete.

Proof. To show that the problem is PSPACE-complete we must show that the problem is PSPACE-hard and that it can be solved in PSPACE. By Lemma 3 we have that the model checking problem for cb-WCTL on a finite n-WKS is PSPACE-hard.

To show that the problem belongs to PSPACE, we construct an algorithm which solves the model checking problem for a minimal set of cb-WCTL operators in polynomial space. Given the n-WKS $K = (S, s_0, \mathcal{AP}, L, T)$ and the cb-WCTL formula φ we have by Lemma 2 that all vectors strictly greater than the formula derived boundary vector $\overline{b}$ are unnecessary to consider. By Definition 3.2 and 3.3 it follows that the set of cut configurations is included in the set $S \times \{0, \ldots, gb\}^n$ and has a size of at most $k = |S| \times |\{0, \ldots, gb\}^n|$. The algorithm now takes the input $((s, \overline{w}), \varphi)$ where $(s, \overline{w}) \in S \times \{0, \ldots, gb\}^n$ and returns a Boolean value corresponding to whether the formula is satisfied. It is defined as a recursive procedure VERIFY and is presented in Algorithm 1.

Algorithm 1 Deterministic polynomial space model checking algorithm

```
1: procedure VERIFY((s, w̄), φ)
2:     if φ is a or (e ⋈ c) or TRUE then
3:         return s ⊨_w̄ φ
4:     else if φ is ¬φ′ then
5:         return !VERIFY((s, w̄), φ′)
6:     else if φ is Reset #R in φ′ then
7:         return VERIFY((s, w̄[R → 0]), φ′)
8:     else if φ is φ1 ∨ φ2 then
9:         for all i ∈ {1, 2} do
10:            if VERIFY((s, w̄), φ_i) then return TRUE
           return FALSE
11:    else if φ is EXφ′ then
12:        for all (s′, w̄′) ∈ {(s′, cut(w̄ + c̄)) | (s, c̄, s′) ∈ T} do
13:            if VERIFY((s′, w̄′), φ′) then return TRUE
           return FALSE
14:    else if φ = EGφ′ then
15:        for all (s′, w̄′) ∈ S × {0, ..., gb}^n do
16:            if VERIFY((s′, w̄′), φ′) then
17:                if s′ ↛ and PATH((s, w̄), (s′, w̄′), k, φ′, TRUE) then return TRUE
18:                else if PATH((s, w̄), (s′, w̄′), k, φ′, FALSE) then return TRUE
           return FALSE
19:    else if φ is Eφ1Uφ2 then
20:        if VERIFY((s, w̄), φ2) or VERIFY((s, w̄), φ1) then
21:            for all (s′, w̄′) ∈ S × {0, ..., gb}^n do
22:                if VERIFY((s′, w̄′), φ2) then
23:                    if PATH((s, w̄), (s′, w̄′), k, φ1, TRUE) then return TRUE
           return FALSE
```

In Algorithm 1 we utilize the PATH algorithm presented in Algorithm 2 which is a boolean function that attempts to find a path between two configu-

rations in m steps while satisfying a formula φ. The algorithm returns TRUE if such a path exists and FALSE otherwise. The PATH algorithm is based on the method presented in Savitch's Theorem [20] which finds a path between an exponential number of nodes in a directed graph using polynomial space. The algorithm essentially works by trying to find a middle point of a path between the two nodes of m steps, and then constructing each half of the path by recursively calling itself while cutting the length required in half.

Algorithm 2 Find path of length m (or shorter) between two configurations $(s, \overline{w}), (s', \overline{w}')$ while satisfying φ

```
1:  procedure PATH((s, w̄), (s', w̄'), m, φ, shorter)
2:      if m = 1 then
3:          if there exist (s, c̄, s') ∈ T s.t. Cut(w̄ + c̄) = w̄' then return TRUE
4:          else if shorter and (s, w̄) = (s', w̄') then return TRUE
5:      else
6:          for all (s_t, w̄_t) ∈ S × {0, ..., gb}^n do
7:              if VERIFY((s_t, w̄_t), φ) then
8:                  if PATH((s, w̄)(s_t, w̄_t), floor(m/2), φ, shorter) then
9:                      if PATH((s_t, w̄_t), (s', w̄'), ceil(m/2, φ, shorter) then return TRUE
10:     return FALSE
```

As input the PATH algorithm takes two configurations, a constant m specifying the length of the path, a formula φ which should hold on each intermediate configuration and a boolean signal *shorter* which is used to determine whether the path is allowed to be shorter than the specified m steps. The signal is used in Line 4 in Algorithm 2 where we also accept a path between two identical configurations even if there is no transition between them, thus allowing the path to be shorter than defined in the input. Whenever we try a new configuration $(s_t, \overline{w}_t)$ for the middle position we call VERIFY$((s_t, \overline{w}_t), \varphi)$ to verify that the formula φ is satisfied for each intermediate configuration.

Recall that $k = |S| \times |\{0, \ldots, gb\}^n|$. We assume that a new configuration can be stored in $O(log(k))$. As the algorithm has a recursion level of $O(log(k))$ and each level uses $O(log(k))$ space to store the stack frame, local variables and the new middle configuration, we reach a space use of $O(log(k)^2)$ plus the space used by the call to VERIFY.

All that remains now is to show that for each case presented in VERIFY we use at most polynomial space and that the depth of the recursion is bounded by the size of input. We go through each case below.

$\varphi = a$ or $\varphi = (e \bowtie c)$ or $\varphi =$ true First we have that for propositions and bounds we can simply evaluate the formula directly.

$\varphi = \neg\varphi'$ For negation we reuse the space and call the procedure again on the subformula formula φ'. We then flip the result.

$\varphi =$ *Reset #R in* φ' For the reset operator we reuse the space and overwrite the configuration and call the procedure again on the subformula φ'.

$\varphi = \varphi_1 \vee \varphi_2$ For disjunction we reuse the space and call the procedure again on each subformula.

$\varphi = EX\varphi'$ For the existential next we reuse the space and call the procedure on each of the possible successors for the subformula φ'. Here we only have to remember the currently explored configuration.

$\varphi = EG\varphi'$ For the existential global operator we search for a configuration $(s', \overline{w}')$ where φ' holds and attempt to find some maximal path from the input configuration $(s, \overline{w})$ to $(s', \overline{w}')$ where all intermediate configurations satisfy φ'. We rely on Lemma 2 and thus look though at most k configurations to establish the existence of a maximal path.

If $s' \not\to$ we simply need to find some path to that configuration. We do so by calling the PATH algorithm with the configurations $(s, \overline{w}), (s', \overline{w}')$, the constant k, the subformula φ' and the boolean flag set to TRUE.

If $s' \to$ we need to ensure that s and s' are part of some infinite path on which φ' always holds. We do this by requiring that there is at least k steps in the path, thus ensuring that two configurations repeat and we have found a loop. To find such a path we simply call PATH again but with FALSE as the last input instead.

For this case we store a single extra configuration which is the intended final configuration plus the space required by the calls to the PATH algorithm. Recall that the call to PATH uses $O(log(k)^2)$ space plus the space needed for a call to VERIFY with the subformula φ' as input. This is in polynomial space as the recursion level of VERIFY is bounded by the input (we always call VERIFY with a subformula).

$\varphi = E\varphi_1 U \varphi_2$ For the existential until operator we first ensure that the original formula either satisfies φ_1 or φ_2. Otherwise there is no way of satisfying $E\varphi_1 U \varphi_2$ regardless of any future successor configurations.

Next we search for a configuration $(s', \overline{w}')$ satisfying the second formula φ_2. Once we find such a configuration we check whether it is reachable by calling the PATH algorithm to find a path from the original configuration to our target configuration in at most k steps where all intermediate configurations satisfy φ_1. Again we rely on Lemma 2 to determine that we need to look though at most k configurations.

In this case we store a single extra configuration which is the intended final configuration satisfying φ_2, plus the space required by the call to PATH. As above this is in polynomial space as the recursion level of VERIFY is bounded by the input.

For every case we have that every recursive call of VERIFY is made with a subformula. As VERIFY can also call the PATH algorithm we have at most $O(|\varphi| \times log(k))$ function calls, where each frame uses $O(log(k))$ space to store the function arguments and local variables (extra configurations). Thus the final space consumption is $O(log(k)^2 \times |\varphi|)$ which is polynomial in the size of the input. □

Remark 2

The algorithm presented in the proof of Theorem 4 is clearly not usable in practise. Instead we suggest to extend an existing model checking algorithm presented in [6]

to our weighted formalism. The idea is to reduce the model checking problem, to the problem of calculating the minimum fixed-point assignment of a dependency graph. While this approach runs in exponential time in the worst case (exponential size encoding), it works on-the-fly and is shown to be efficient on regular CTL model checking [6].

4. Synthesis

In this section we present a game theoretic framework based on the multi-weighted formalism presented in Section 2 and formally define the synthesis problem. We comment on the memory requirements of a winning strategy and prove that for a cost-convergent model w.r.t. an upper-bounded specification the synthesis problem is decidable.

4.1. Game Theoretic Framework

An n-Weighted Game (n-WG) is a two-player game where one player acts as the controller and the other player acts as the environment. A game is played on a game graph, where the transitions have been partitioned between the two players.

Definition 4.1 (n-Weighted Game Graph) An n-Weighted Game Graph is a tuple $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ where T_c and T_u are disjoint sets and $K = (S, s_0, \mathcal{AP}, L, T_c \cup T_u)$ is an n-WKS.

The underlying data structure of the game graph is an n-WKS and we denote the specific n-WKS for the game graph $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ as $K_{\mathcal{G}} = (S, s_0, \mathcal{AP}, L, T_c \cup T_u)$. The set of transitions T_c is owned by the controller, and the set T_u is owned by the environment. We write:

$$s \xrightarrow{\overline{c}} s' \text{ if } (s, \overline{c}, s') \in T_c$$
$$s \overset{\overline{c}}{\dashrightarrow} s' \text{ if } (s, \overline{c}, s') \in T_u$$

From here on we will refer to transitions of the type $\rightarrow$ as controllable transitions and $\dashrightarrow$ as uncontrollable transitions. We write $s \rightarrow$ when there is some controllable outgoing transitions from s and $s \dashrightarrow$ when there is some uncontrollable outgoing transition from s.

Lastly we define a winning condition (objective) as a cb-WCTL formula φ over the n-WKS $K_{\mathcal{G}}$.

Definition 4.2 (Game) A game is a tuple $(\mathcal{G}, \varphi)$ where $\mathcal{G}$ is an n-Weighted Game Graph (n-WGG) and φ is a cb-WCTL formula expressing a winning condition.

To define the state of a game we say that a configuration $(s, \overline{w}) \in S \times \mathbb{N}_0^n$ in the game consists of the current state s and the accumulated cost $\overline{w}$. We define the set of all configurations as $\mathcal{C}$. The game is played by moving from configuration to configuration, where $(s_0, 0^n)$ is the initial position. Given a configuration $(s, \overline{w})$ we proceed in the following manner:

- If all outgoing transitions from s belong to T_c, then the controller must choose.
- If all outgoing transitions from s belong to T_u, then the environment must choose.
- If there are both outgoing transitions in T_c and T_u from s, then the environment may choose one from T_u or force the controller to choose from T_c.

We give precedence to the environment when there are both controllable and uncontrollable outgoing transitions as this enforces the notion of an environment beyond our control. Once either the controller or the environment has chosen an edge $(s, \overline{c}, s') \in T_u \cup T_c$ the next configuration becomes $(s', \overline{w} + \overline{c})$.

The controller's choices are based on a strategy σ that, given the history of the game, outputs the controller's next move. Recall that $\Pi^{fin}_{K_\mathcal{G}}$ is the set of all finite runs in $K_\mathcal{G}$. We now define a strategy as a function mapping from finite runs to controllable transitions.

Definition 4.3 (Strategy) Let $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ be an n-WGG, then the strategy of the controller is a function $\sigma : \Pi^{fin}_{K_\mathcal{G}} \to T_c \cup \{\textsc{nil}\}$ mapping a finite run ρ to a transition s.t.

$$\sigma(\rho) = \begin{cases} \textsc{Last}(\rho) \xrightarrow{\overline{c}} s' \in T_c & \text{if } \textsc{Last}(\rho) \to \\ \textsc{nil} & \text{otherwise} \end{cases}$$

and NIL is the choice to do nothing. Notice that we restrict the use of NIL s.t. for any $\rho \in \Pi^{fin}_{K_\mathcal{G}}$ we have $\sigma(\rho) = \textsc{nil}$ only if $\textsc{Last}(\rho) \not\to$.

Based on the strategy's choices and all choices available to the environment, we unfold the game into an n-WKS.

Definition 4.4 (Strategy restricted n-WKS) Given a game graph $K = (S, s_0, \mathcal{AP}, L, T)$ and a strategy σ, we define $\mathcal{G}\restriction\sigma = (S', s_0, \mathcal{AP}, L', T_c\restriction\sigma \cup T'_u)$ as the n-WKS resulting from restricting the game graph under the strategy σ.

- $S' = \Pi^{fin}_{K_G}$
- $L'(\rho) = L(\textsc{Last}(\rho))$
- $T'_c\restriction\sigma = \{(\rho, \overline{c}, (\rho \circ \sigma(\rho))) \mid \sigma(\rho) = (\textsc{Last}(\rho) \xrightarrow{\overline{c}} s') \in T_c\}$
- $T'_u = \{(\rho, \overline{c}, \rho \circ (\textsc{Last}(\rho) \xrightarrow{\overline{c}} s')) \mid (\textsc{Last}(\rho) \xrightarrow{\overline{c}} s') \in T_u\}$.

In future illustrations we only include the part of $\mathcal{G}\restriction\sigma$ reachable from s_0. When the unfolded game restricted by the strategy, satisfy the winning condition φ we define it as a winning strategy. Hence the controller wins the game if they have a winning strategy.

Definition 4.5 (Winning Strategy) The strategy σ is a *winning strategy* over the game $(\mathcal{G}, \varphi)$ iff $\mathcal{G}\restriction\sigma, s_0 \vDash_{0^n} \varphi$, where $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ and φ is a cb-WCTL formula.

In Example 3 we return to the introductory scenario presented in Section 1.1 with a simpler specification. We determine a winning strategy for the resulting game and illustrate how model checking relates on the strategy restricted n-WKS.

Example 3 (Strategy Unfolding)
Let $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ be the n-WGG originally presented in Section 1.1 (repeated in Figure 5 for convenience) and let $AX(AX(EX(\#1 \leq 0 \wedge \#2 \geq 50)))$ be the winning objective expressed as an cb-WCTL formula.

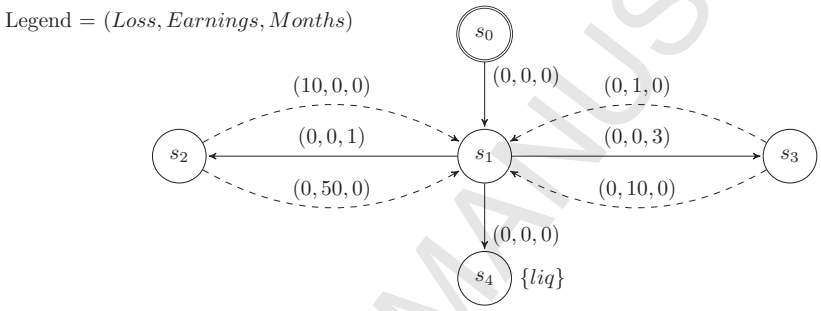


Figure 5: Model of investment scheme

Then our goal is to plan our investment so that no matter what our first two steps are there exists a transition leading to a state where we have no losses and a 50 units earning. For this purpose we define the following strategy:

$$\sigma(s_0) = s_0 \xrightarrow{(0,0,0)} s_1 \qquad \sigma(s_0 \xrightarrow{(0,0,0)} s_1) = s_1 \xrightarrow{(0,0,1)} s_2$$

for the remainder of the runs we simply assign transitions arbitrarily. Given the strategy σ we can now construct the strategy restricted n-WGG $\mathcal{G}\restriction\sigma$ which is illustrated below in Figure 6.

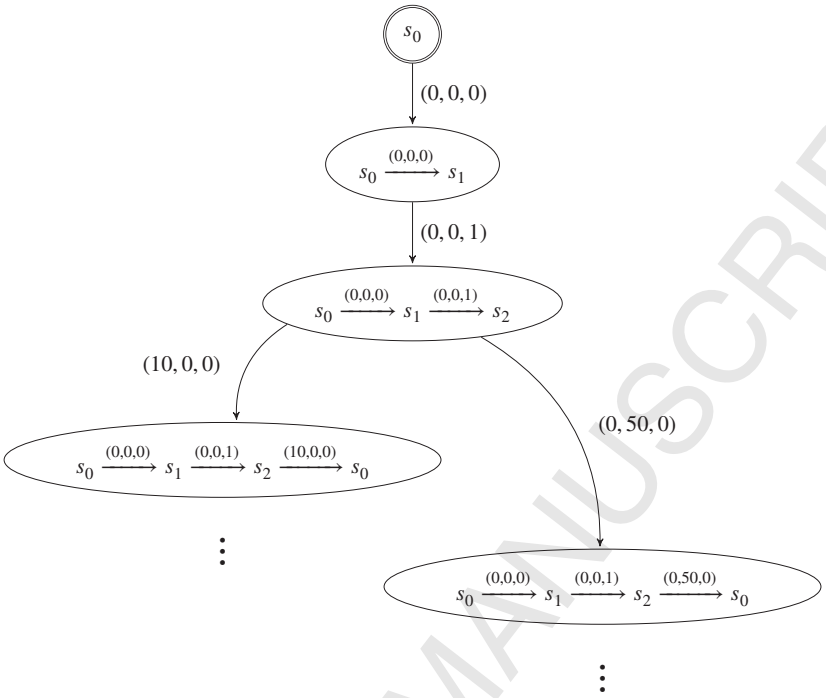


Figure 6: Strategy restricted n-WGG $\mathcal{G}\restriction\sigma = (S', s_0, \mathcal{AP}, L', T_c\restriction\sigma \cup T'_u)$

To show that σ is winning we must have that for all runs of length two, it is possible to take a transition leading to a state with no losses and earnings of 50 units. In this example our strategy imposes that we choose the transition $s_1 \xrightarrow{(0,0,1)} s_2$, representing the high risk investment, and then the only run of length two is,

$$\rho = s_0 \xrightarrow{(0,0,0)} s_1 \xrightarrow{(0,0,1)} s_2$$

and we have that $cost(\rho) = (0, 0, 1)$. Now there are two possible outgoing transitions $s_2 \xrightarrow{(10,0,0)} s_1$ and $s_2 \xrightarrow{(0,50,0)} s_1$, both uncontrollable and thus both part of the strategy restricted n-WKS. As the second transition does not generate any losses and 50 units of earnings, we have that σ is indeed a winning strategy for the game $(\mathcal{G}, \varphi)$ as $\mathcal{G}\restriction\sigma, s_0 \models_{0^n} AX(AX(EX(\#1 \leq 0 \wedge \#2 \geq 50)))$.

We have now introduced the notion of a game and a winning strategy and are ready to define the synthesis problem.

Definition 4.6 (Synthesis Problem) Let $(\mathcal{G}, \varphi)$ be a n-WG where $K = (S, s_0, \mathcal{AP}, L, T)$ is an n-WGG and φ is a cb-WCTL formula. The *synthesis problem* is to decide if there is a strategy σ s.t. $\mathcal{G}\restriction\sigma, s_0 \models_{0^n} \varphi$.

The synthesis problem is then the question of whether there exist a strategy s.t. a specification is satisfied on the computation tree resulting from applying

the strategy. As all possible branches are now important to the existentially or universally quantified formula, intuitively it does not make sense to talk about a specific strategy for the environment.

Example 4 (Motivating Example Formalised)
With n-WGs we can now express the motivating example presented in Section 1.1 as a synthesis problem. Recall the informal objective "Within a year, can we achieve earnings of at least 600 units, while never experiencing a loss of more than 30 units over a period of 6 months." While this specification is slightly ambiguous we interpret the 600 unit earnings as optimistic case, achievable if the environment cooperates. We specify this part of the objective as the cb-WCTL formula φ_1.

$$\varphi_1 = EF(\#2 \geq 600 \wedge \#3 \leq 12 \wedge \{liq\})$$

The specification is read as follows: Is there any possibility that can earn at least 600 units (component two is larger or equal to 600) within a year (component three is smaller or equal to 12) and be at a state where we liquidate our assets ($\{liq\}$)*?*

For the second part of the informal objective, we must ensure that we do not end up losing more than 30 units within 6 months. This requirement is specified in φ_2 *as an invariant property which, given fresh values for the first and third vector, ensures that component one is below 30 until component three is above 6.*

$$\varphi_2 = AG(reset\ \#\{1,3\}\ in\ (A(\#1 \leq 30)U(\#3 \geq 6))$$

Now our final specification is simply the conjunction of these requirements.

$$\varphi = \varphi_1 \wedge \varphi_2$$

The synthesis problem is then to find a strategy σ *s.t.* $\mathcal{G}\restriction\sigma, s_0 \vDash_{0^n} \varphi$.

4.2. Strategy Memory Requirements

In Definition 4.3 we have introduced what we refer to as a *full memory strategy* where each decision is based on the full run leading to the current state. For reachability and safety games it is known that so called memory-less strategies, which only take the current state into account, are sufficient [22]. We will later demonstrate that when adding the current cost to the state information, this is also sufficient for weighted reachability games.

In contrast the memory needed for expressing a winning strategy increases when we consider a larger subset of the cb-WCTL logic. Even if one considers expanding the memory required to a sequence of visited states, this is not always sufficient to define a winning strategy. To demonstrate this we refer to Example 5.

Example 5 (Memory requirements)
Let $(\mathcal{G}, \varphi)$ *be a n-WG where the n-WGG* $\mathcal{G}$ *is illustrated in Figure 7 and the winning condition is the formula* $\varphi = \varphi_1 \wedge \varphi_2$ *where* $\varphi_1 = EX(AF(\#1 = 4 \wedge \alpha))$ *and* $\varphi_2 = EX(AF(\#1 = 4 \wedge \beta))$.

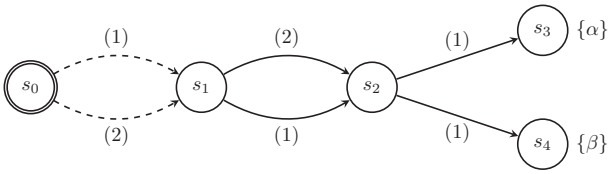


Figure 7: Illustrating memory requirements

It is relatively easy to verify that as both $s_0 \xrightarrow{(1)} s_1$ *and* $s_0 \xrightarrow{(2)} s_1$ *are part of the resulting strategy restricted game graph, then there exist a winning strategy. To illustrate consider the strategy* σ *where given transition* $s_0 \xrightarrow{(1)} s_1$ *the strategy choice is* $s_1 \xrightarrow{(2)} s_2$ *followed by* $s_2 \xrightarrow{(1)} s_3$*. This will result in a branch fulfilling the subformula* φ_1*. The strategy then chooses the opposite combination given the transition* $s_0 \xrightarrow{(2)} s_1$*, fulfilling the second subformula* φ_2*.*

In order to define such a strategy we need to remember both the part of the run consisting of uncontrollable transitions (which causes the branching) as well as the current state. Let us shortly argue this point in relation to the presented game. For the winning condition to be satisfied, we need to be able to visit both s_3 *and* s_4 *with the cost* 4*. Thus at the position* $(s_2, 3)$ *the strategy must able to make different choices. Here the current configuration as well as the sequence of states is identical. The only identifying piece of memory is the particular uncontrollable transitions chosen. Without this information we are forced to make the same choice every time we reach* $(s_2, 3)$*, and we are thus not able to satisfy the formula.*

4.3. Decidability of Upper-bounded WCTL Synthesis

We find that if we restrict the logic to upper-bounds on the temporal operators and require that all infinite runs of the game graph increase the cost components w.r.t. those upper-bounds, we achieve a simple decidability result for the resulting synthesis problem. First we formally define the new logic as follows:

Definition 4.7 (Upper-bounded WCTL (ub-WCTL)) We define an ub-WCTL formula as a cb-WCTL formula where the temporal operators have upper-bounds. We use the following syntax abbreviations:

$$E(\varphi_1)U_{(e\leq c)}(\varphi_2) \equiv E(\varphi_1)U(\varphi_2 \wedge (e \leq c)) \quad EF_{(e\leq c)}(\varphi) \equiv EF(\varphi \wedge (e \leq c))$$
$$A(\varphi_1)U_{(e\leq c)}(\varphi_2) \equiv A(\varphi_1)U(\varphi_2 \wedge (e \leq c)) \quad AF_{(e\leq c)}(\varphi) \equiv AF(\varphi \wedge (e \leq c))$$

$$AG_{(e\leq c)}\varphi \equiv AG((e \leq c) \implies \varphi)$$
$$EG_{(e\leq c)}\varphi \equiv EG((e \leq c) \implies \varphi)$$

We define the set of all upper-bounds $(e \leq c)$ on the temporal operators in a ub-WCTL formula φ as $UB(\varphi)$. Whenever there exist a bounded expression $(e \bowtie c) \in UB(\varphi)$ and $\#i$ is part of the expression e we say that the component $\#i$ is bounded in φ.

Based on an ub-WCTL formula, we can now impose a restriction on the model, s.t. all cycles have to increase in all upper-bounds expressed in the formula.

Definition 4.8 (Cost-convergent *n*-WKS) We say that a n-WKS K is *cost convergent* w.r.t. a ub-WCTL φ if for all cycles $(\rho_c = s \rightarrow^+ s) \in \Pi_K^{fin}$ and for all $1 \leq i \leq n$ where $\#i$ is bounded there exists a position j s.t. $\rho_c(j) \xrightarrow{\bar{c}} \rho_c(j+1)$ where $\bar{c}[i] > 0$.

Together these restrictions allow us to state the following result.

Theorem 5 (Decidability of ub-WCTL synthesis)
Given an n-WG $(\mathcal{G}, \varphi)$ where φ is an ub-WCTL formula and $K_{\mathcal{G}}$ is a cost convergent n-WKS w.r.t. φ, then the synthesis problem is decidable.

Proof. We define two strategies σ and σ' as $n-$equivalent, denoted $\sigma =_n \sigma'$ iff $\sigma(\rho) = \sigma'(\rho)$ for all input ρ of length $n \in \mathbb{N}_0$. We now argue that for cost convergent games there exist a sufficiently large number n s.t. if $\sigma =_n \sigma'$ then $\mathcal{G}\restriction\sigma, s \models_{\overline{w}} \varphi$ iff $\mathcal{G}\restriction\sigma', s \models_{\overline{w}} \varphi$.

To find the number n we first identify the largest bound in the formula $gb = max(\{c \mid (e \leq c) \in UB(\varphi)$. By Definition 4.8 we know that for all cycles the cost increases in all bounded components. By the semantics, we have that whenever the cost of a run reaches some upper-bound $(e \leq c)$ then any formula containing this (or a more strict) bound, becomes either trivially false (reachability) or trivially true (safety). As any cycle contains at most $|S|$ states then $(|S| \times gb)$ number of steps will ensure that all bounds are reached. Of course any subformula φ' can be prefixed with the reset operator, which naturally resets (and thus in effect extends) the length of the run required to determine φ'. However, as each reset only affects a specific subformula, then for the formula φ we arrive at the following length $n = (|S| \times gb \times |Sub(\varphi)|)$.

We can now simply enumerate every possible strategy up to n steps and verify the formula on the strategy restricted n-WKS using the model checking result from Theorem 4. □

While this provides us with a decidability result for a logic which allows nesting of all CTL operators and the use of both lower- and upper-bound as propositions, it is also limiting as it requires upper-bounds on the temporal operators. In the following section we look at a subset of the cb-WCTL, which does not require a upper-bound or cost convergence in the model.

5. Synthesis of Weighted Reachability

In this section we investigate the synthesis problem for a reachability subset of cb-WCTL, where there are no restrictions on the logic or the model. We show that the synthesis problem for this reachability logic is EXPTIME-complete and provide an on-the-fly algorithm for calculating the winning strategy if one exist.

Definition 5.1 (Weighted Reachability) We define the reachability subset Weighted Reachability (WReach) as cb-WCTL with the following syntax:

$$\begin{aligned}\varphi &:= AF\psi \\ \psi &:= a \mid \neg a \mid (e \bowtie c) \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \\ e &:= c \mid \#i \mid e_1 \oplus e_2\end{aligned}$$

where $a \in \mathcal{AP}$, $\bowtie \in \{<, \leq, =, \geq, >\}$, $c \in \mathbb{N}_0$, and $1 \leq i \leq n$ is a component index in a vector.

We define a *WReach game* as a game $(\mathcal{G}, \varphi)$ where $\mathcal{G}$ is n-WGG and φ is a WReach formula.

5.1. Dependency Graphs

Given the complexity of the problem it is preferable that the solution works on-the-fly. We propose a reduction from the synthesis problem for WReach games to the problem of calculating the minimum fixed-point assignment of a dependency graph.

Definition 5.2 (Dependency graph) A dependency graph is a tuple $G = (V, E)$ where

- V is a finite set of nodes and
- $E \subseteq V \times \mathcal{P}(V)$ is a finite set of hyper-edges.

Given a hyper-edge $e = (s, T) \in E$ we say that $s \in V$ is the source node and $T \subseteq V$ is the set of target nodes.

An assignment of a DG $G = (V, E)$ is a function $A : V \to \{0, 1\}$ that assigns value 0 (false) or 1 (true) to each node in the graph. We assume an ordering $\sqsubseteq$ of assignments s.t. $A_1 \sqsubseteq A_2$ whenever $A_1(v) \leq A_2(v)$ for all $v \in V$. The set of all assignments is denoted $\mathcal{A}$ and clearly $(\mathcal{A}, \sqsubseteq)$ is a complete lattice. Given the monotonic function $F : \mathcal{A} \to \mathcal{A}$ defined as:

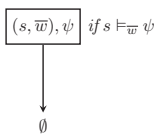
$$F(A)(v) = \bigvee_{(v,T)\in E} \bigwedge_{u \in T} A(u)$$

then by Knaster-Tarski we have that there exist a minimum fixed-point assignment denoted A_G^{min}. A pre fixed-point assignment of G is an assignment A where it holds for every $(v, T) \in E$ that if $A(u) = 1$ for all $u \in T$ then $A(v) = 1$. To compute the minimum fixed-point assignment we apply F repeatedly on the assignment starting with $F(A^0)$ where A^0 is defined s.t. $A^0(v) = 0$ for all $v \in V$.

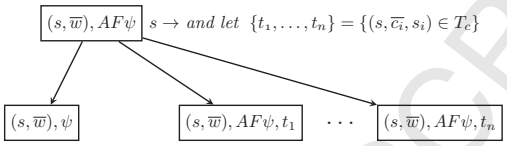
In [15] Liu and Smolka provide a local algorithm for calculating the minimal fixed-point assignment of a finite dependency graph, and show that this algorithm runs in linear time in the size of the graph.

5.2. Encoding of the WReach Synthesis Problem

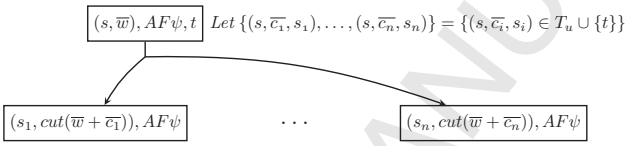
We now provide an encoding of an n-WG into a finite dependency graph. Given a finite game $(\mathcal{G}, AF\psi)$ where $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ is an n-WGG and $AF\psi$ is an WReach formula we construct a dependency graph G s.t. the initial node is $\langle (s_0, 0^n), AF\psi \rangle$. The rules for generating the successors are defined Figure 8.



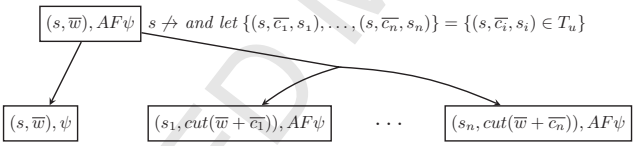
(a) Propositions



(b) Choosing the strategy given the configuration $(s, \overline{w})$



(c) Encoding of the full $AF\psi$ based on the chosen transition t



(d) Encoding of the full $AF\psi$ when there are no controllable transitions

Figure 8: Encoding of WReach games

In Figure 8a we illustrate the case where we directly evaluate the propositions ψ given the current position $(s, \overline{w})$. In Figure 8b we have the case where there is some controllable transition to choose as the strategy given the position $(s, \overline{w})$. The moves of the game are then to check whether player 1 has already won (if $s \vDash_{\overline{w}} \psi$) or explore all possible strategies. In Figure 8c we then take the strategy into account and explore all new reachable configurations of the game. The last Figure 8d shows the case where given the position $(s, \overline{w})$ there is no outgoing controllable transition. Again we check if player 1 has already achieved the objective or explore all reachable configurations of the game, in relation to the full formula.

Notice that even in a finite game a step-wise unfolding can be infinite, as the cost of cycle can increase infinitely. To ensure a finite representation we apply *Cut* to the cost of the successors. To illustrate this approach we return to the motivating example presented in Section 1.1 in Example 6 where we show the encoding to dependency graphs.

Example 6 (Encoding of WReach game)
Let $(\mathcal{G}, \varphi)$ be a WReach game where $\mathcal{G}$ is the n-WGG presented in Section 1.1 and $\varphi = AF(\#1 \leq 0 \wedge \#2 \geq 4 \wedge \#3 \leq 12 \wedge \{liq\})$ is a WReach formula.

Then the dependency graph G illustrated in Figure 9 is based on the game $(\mathcal{G}, \varphi)$ with root $\langle (s_0, (0,0,0)), \varphi \rangle$ where the successors are defined by the rules in Figure 8. To simplify the illustration we omit the intermediate nodes generated in Figure 8b and jump directly to the position we get from taking the chosen transition illustrated in Figure 8c.

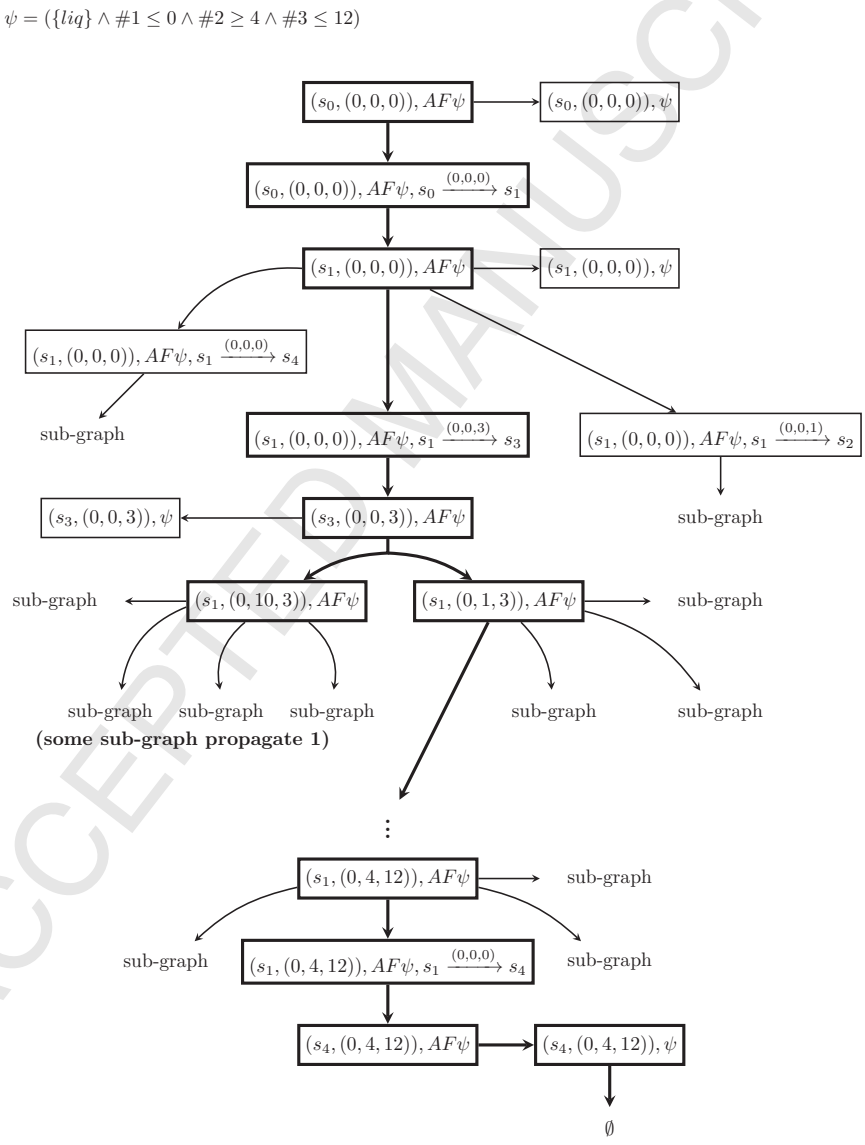


Figure 9: Dependency graph G with root $\langle (s_0, (0,0,0)), AF(\#1 \leq 0 \wedge \#2 \geq 4 \wedge \#3 \leq 12 \wedge \{liq\})\rangle$

Intuitively then whenever a node $\langle (s, \overline{w}), \varphi\rangle$ has the value 1, then there is a winning strategy from the position $(s, \overline{w})$ for the winning condition φ.

To simplify the definition of the strategy we only give the position as the input. Thus by following the one assignment of the dependency graph we get the following strategy

$$\sigma(\rho) = \begin{cases} s_0 \xrightarrow{(0,0,0)} s_1 & \text{if } last(\rho) = s_0 \\ s_1 \xrightarrow{(0,0,3)} s_3 & \text{if } last(\rho) = s_1 \text{ and } cost(\rho)[3] \leq 9 \\ s_1 \xrightarrow{(0,0,0)} s_4 & \text{if } last(\rho) = s_1 \text{ and } cost(\rho)[3] > 9 \end{cases} \cdot$$

Notice that we chose the strategy s.t. we always choose the low risk investment. Based on this strategy we can now construct the strategy restricted n-WKS $\mathcal{G}\restriction\sigma$ and verify that $\mathcal{G}\restriction\sigma, s_0 \vDash_{0^n} AF(\#1 \leq 0 \wedge \#2 \geq 4 \wedge \#3 \leq 12 \wedge \{liq\})$.

Lemma 6
Given a finite WReach game $(\mathcal{G}, AF\psi)$ then the dependency graph G with root $\langle (s, \overline{w}), AF\psi\rangle$ constructed by the rules in Figure 8 is finite.

Proof. We have that a node in the graph $\langle (s, \overline{w}), \varphi\rangle$ consist of a configuration $(s, \overline{w})$ and either the entire formula $AF\psi$ or the evaluable part of the formula ψ. Additionally the nodes of the type $\langle (s, \overline{w}), AF\psi\rangle$ can be decorated with a controllable transition. This gives us the following total number of configurations $(|S| \times |\{cut(\overline{w}) \mid \overline{w} \in \mathbb{N}_0^n\}| \times |T| \times 2)$. □

Theorem 7 (Encoding correctness)
Let $(\mathcal{G}, AF\psi)$ be a finite WReach game where $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ is an n-WGG, $s \in S$ and $\overline{w} \in \mathbb{N}_0^n$. Let G be the constructed dependency graph with root $\langle (s, \overline{w}), AF\psi\rangle$. Then there exists a strategy σ s.t. $\mathcal{G}\restriction\sigma, s \vDash_{\overline{w}} AF\psi$ iff $A_G^{min}(\langle (s, \overline{w}), AF\psi\rangle) = 1$.

Proof. First we consider the direction from left to right. Let σ be a strategy s.t. $\mathcal{G}\restriction\sigma = (S', s_0, \mathcal{AP}, L', T_c\restriction\sigma \cup T_u')$ and $\mathcal{G}\restriction\sigma, s \vDash_{\overline{w}} AF\psi$ then we show that $A_G^{min}(\langle (s, \overline{w}), AF\psi\rangle) = 1$. By the semantics we have that $\mathcal{G}\restriction\sigma, s \vDash_{\overline{w}} AF\psi$ if there exist some smallest $j \geq 0$ s.t. for all maximal runs $(\boldsymbol{\rho} = \rho_0 \xrightarrow{\overline{c_1}} \rho_1 \xrightarrow{\overline{c}_2} \rho_2 \dots) \in \Pi_{\mathcal{G}\restriction\sigma}^{Max}$ where $\rho_0 = s$ then $\rho_i \vDash_{\overline{w}+cost_\rho(i)} e$ for some $i \leq j$. Hence all maximal runs satisfy the property within j steps. Then by the semantics we have that for all transitions $(s, \overline{c}, (s \circ (s, \overline{c}, s'))) \in T_u' \cup T_c\restriction\sigma$ then $\mathcal{G}\restriction\sigma, (s \circ (s, \overline{c}, s')) \vDash_{\overline{w}+\overline{c}} AF\psi$ within $j-1$ steps. We proceed by induction of j:

- $j = 0$ then we have that $s \vDash_{\overline{w}} e$ and by Figure 8d and by Figure 8b we have that there exists a hyper-edge $(\langle (s, \overline{w}), AF\psi\rangle, \{\langle (s, \overline{w}), e\rangle\})$ and as the condition $s \vDash_{\overline{w}} e$ is fulfilled we have that there is a single outgoing edge leading to the empty set. Hence we have that $A^{min}(\langle (s, \overline{w}), e\rangle) = 1$.
- $j > 0$ then either $\sigma(s) = \text{NIL}$ and then by Figure 8d we have that there is a hyper-edge $(\langle (s, \overline{w}), AF\psi\rangle \{\langle (s_i, \overline{c_i} + \overline{w}), AF\psi\rangle \mid (s, \overline{c_i}, s_i) \in T_u\})$ and by the induction hypothesis on j all target nodes have the assignment 1,

hence $A^{min}(\langle\langle(s,\overline{w}),AF\psi\rangle\rangle) = 1$. Otherwise $\sigma(s) = (s,\overline{c_j},s_j) \in T_c$ and then by Figure 8b there exist a hyper-edge going to the target state $\langle\langle(s,\overline{w}),AF\psi^{(s,\overline{c_j},s_j)}\rangle\rangle$. From this node we have the following hyper-edge $(\langle\langle(s,\overline{w}),AF\psi^{(s,\overline{c_j},s_j)}\rangle\rangle\{\langle\langle(s_j,\overline{c_j}+\overline{w}),AF\psi\rangle\rangle \cup \{\langle\langle(s_i,\overline{c_i}+\overline{w}),AF\psi\rangle\rangle \mid (s,\overline{c_i},s_i) \in T_u\}\})$. By the induction hypothesis on j all target nodes have the assignment 1, hence $A^{min}(\langle\langle(s,\overline{w}),AF\psi\rangle\rangle) = 1$

Next we consider the direction from right to left. Let $A^{min}_G(\langle\langle(s,\overline{w}),AF\psi\rangle\rangle) = 1$ then we show that there exists a strategy σ s.t. $\mathcal{G}\restriction\sigma, s \vDash_{\overline{w}} AF\psi$. First we notice that for a node to have the value 1 then the value must have propagated from a node of the form $\langle\langle(s',\overline{w}'),e\rangle\rangle$ where $s' \vDash_{\overline{w}'} e$. As $A^{min}(\langle\langle(s,\overline{w}),AF\psi\rangle\rangle) = 1$ then either $A^{min}(\langle\langle(s,\overline{w}),e\rangle\rangle) = 1$ and then by structural induction hypothesis there exist some σ s.t. $\mathcal{G}\restriction\sigma, s \vDash_{\overline{w}} e$ hence we also have that $\mathcal{G}\restriction\sigma, s \vDash_{\overline{w}} AF\psi$. Otherwise, there must be a finite number of steps down to a node which is assigned the value 1. We say the smallest number of steps required to reach such a node is the depth of a node. The remainder of the proof now goes by induction on the depth.

If $s \not\rightarrow$ then by Figure 8d we must have for all transitions $(s,\overline{c_i},s_i) \in T_u$ that $A^{min}(\langle\langle(s_i,\overline{c_i}+\overline{w}),AF\psi\rangle\rangle) = 1$. As the depth of these target nodes must be strictly lower than our source node, then by induction on the depth we know that there exist a strategy σ s.t. $\mathcal{G}\restriction\sigma, s_i \vDash_{\overline{w}+\overline{c_i}} AF\psi$. Thus $\sigma(s) = \text{NIL}$ is a winning strategy from the initial position $(s,\overline{w})$.

Otherwise $s \rightarrow$ and then by Figure 8b there must exist some controllable transition $(s,\overline{c'},s') \in T_c$ s.t. $A^{min}(\langle\langle(s,\overline{w}),AF\psi^{(s,\overline{c'},s')}\rangle\rangle) = 1$. By Figure 8c we have that for all uncontrollable transitions and the single controllable transition $(s,\overline{c_i},s_i) \in \{T_u \cup (s,\overline{c'},s')\}$ then $A^{min}(\langle\langle(s_i,\overline{w}+\overline{c_i}),AF\psi\rangle\rangle) = 1$. As the depth of these target nodes must be strictly lower than our source node, then by induction on the depth there exist a strategy σ s.t. $\mathcal{G}\restriction\sigma, s_i \vDash_{\overline{w}+\overline{c_i}} AF\psi$. Hence given $\sigma(s) = (s,\overline{c'},s')$ then σ is a winning strategy from the initial position $(s,\overline{w})$. □

We notice that given the encoding of the game, the strategy relies solely on the last state and the accumulated cost of a run, thus giving us a finite memory strategy.

We find that the synthesis problem for WReach games is EXPTIME-hard, even when we only have one weight and require cost convergence. We show this by reduction from countdown games.

Lemma 8
The synthesis problem for 1-weighted WReach games is EXPTIME-hard.

Proof. We show this by reduction from countdown games which were proved to be EXPTIME-complete in [11]. A countdown game (Q,R) consists of a set of states Q and a transition relation $R \subseteq Q \times \mathbb{N} \times Q$. We write transitions as $(q,k,q') \in R$ and say that the duration of the transition is k. A configuration in the game is a pair (q,c) where $q \in Q$ and $c \in \mathbb{N}$ and the rules of the game are now defined as follows: Given the configuration (q,c) then,

- if $c = 0$ then player one wins, else

- if for all transitions $(q, k, q') \in R$ then $k > c$ and $c > 0$ then player 2 wins,
- otherwise there exist some transition $(q, k, q') \in R$ s.t. $k < c$. Then player one must chose such a durations k, and player two will now chose a target state q' s.t. $(q, k, q') \in R$ and the new configuration is then $(q', c - k)$.

As all edges have non negative durations this game results in a finite number of rounds. Now we simply need to reduce the problem of deciding whether player 1 has a winning strategy for a configuration (q, c) in the countdown game (Q, R) to deciding whether there exist a winning strategy in a WReach game.

We create a n-WGG $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$ from the countdown game (Q, R) as follows: $S = Q \cup \{s_q^k \mid (q, k, q') \in R\}$, $\mathcal{AP} = \emptyset$, $s_0 = q_0$ is the initial position and $T_c = \{(q, k, s_q^k) \mid (q, k, q') \in R \text{ and } s_q^k \in S\}$ and $T_u = \{(s_q^k, 0, q') \mid s_q^k \in S \text{ and } (q, k, q') \in R\}$. To enforce the rules we simply create the following winning condition $\varphi = AF(\#1 \leq c \wedge c \leq \#1)$. Now it is trivial to see that if there exist a winning strategy σ s.t. $\mathcal{G}\restriction\sigma, s \vDash_0 \varphi$, then player 1 has a winning strategy in the corresponding countdown game (Q, R) given the configuration (q_0, c). □

Theorem 9
The synthesis problem for WReach games is EXPTIME-complete.

Proof. Let $(\mathcal{G}, AF\psi)$ be a WReach game. Then by Lemma 8 we have that the synthesis problem is EXPTIME-hard. Next we show that the problem is in EXPTIME.

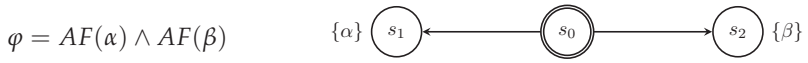
By Lemma 6 and Theorem 7 we can create a finite dependency graph G s.t. there exist a strategy σ where $\mathcal{G}\restriction\sigma, s \vDash_{\overline{w}} AF\psi$ iff $A_G^{min}(\langle\langle(s, \overline{w}), AF\psi\rangle\rangle) = 1$. Additionally by Lemma 6 we have that the number of nodes constructed is exponential in the number of dimensions in the game graph. As the minimum fixed-point algorithm runs in linear time [15], we have that the synthesis problem for WReach games belong in EXPTIME. □

5.3. Challenges in Extending The Encoding

We find that a straight forward extension of the method presented in the previous section towards more powerful logics is not possible. In our current method we build the strategy compositionally based on subformula. However, the fact that we have a winning strategy for each subformula φ_1, φ_2 does not mean there is a strategy for $\varphi_1 \wedge \varphi_2$. To illustrate this we refer to Example 7.

Example 7 (Conjunction of temporal operators)
We begin by defining the game $(\mathcal{G}, \varphi)$ where the winning condition is specified in Figure 10a and the game graph $\mathcal{G}$ is shown in Figure 10b.

We find that there exist winning strategies for both subformula. For $\varphi_1 = AF(\alpha)$ a strategy σ_1 can be defined s.t. $\sigma_1(s_0) = s_0 \to s_1$, and for $\varphi_2 = AF(\beta)$ then a strategy σ_2 can be defined where $\sigma_2(s_0) = s_0 \to s_2$. Clearly these are winning strategies for the corresponding subformula. While the existence of winning strategies for both subformula may suggest that there exists a winning strategy for the conjunction, it



(a) formula **(b)** $\mathcal{G} = (S, s_0, \mathcal{AP}, L, T_c, T_u)$

Figure 10: n-WG $(\mathcal{G}, \varphi)$ illustrating a game with separate winning strategies for each subformula

is obvious that there cannot be such a strategy σ in this example. From the definition of the strategies above we can tell that one of the controllable transitions will not be present in the strategy restricted n-WKS and both are required to satisfy the formula.

Notice that this is not a problem with disjunction, as a strategy satisfying either side will by the semantics satisfy the disjunction itself. However it is not enough to simply have separate winning strategies for both sides of a conjunction in a branching formula. They also need to be *compatible* meaning that they are consistent in their choice of controllable transitions given the same configuration.

To incorporate this we need to either radically change the method, or find a way to keep the conjuncted formula together trough the unfolding. This is an area of future work we are currently pursuing.

6. Conclusion

We studied the model checking and synthesis problem for a multi-weighted extension of Kripke structures and weighted CTL. In relation to the model checking, we found that the full WCTL model checking problem is undecidable. However, by restricting the logic to only consider constant bounds, the problem becomes decidable and is in fact PSPACE-complete.

In the synthesis setting, we extended the formalism to a game setting with two players: a controller and an environment. We then defined the synthesis problem as the question of whether the controller has a winning strategy such that the unfolded system, according to this strategy, satisfies a given WCTL formula. By imposing upper-bounds on the temporal operators and restricting ourselves to only consider cost-convergent game graphs, we demonstrated that the synthesis problem is decidable. Lastly, we provided an on-the-fly algorithm and a tight complexity bound of EXPTIME for the reachability subset of the logic which allows both lower- and upper-bounds.

In future work we plan to extend this method to include cb-WCTL, thus proving the synthesis problem decidable without any additional restrictions on the model.

Acknowledgements.. We thank the reviewers for their useful comments and suggestions. The work was funded by the center IDEA4CPS, Innovation Fund Denmark center DiCyPS and ERC Advanced Grant LASSO. The last author is partially affiliated with FI MU, Brno.

[1] Shaull Almagor, Udi Boker, and Orna Kupferman. Formally Reasoning About Quality. *Journal of the ACM*, 63(3):1–56, jun 2016.

[2] Udi Boker, Krishnendu Chatterjee, Thomas A Henzinger, and Orna Kupferman. Temporal Specifications with Accumulative Values. *TOCL*, 15(4):27:1—-27:25, 2014.

[3] Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In *FORMATS*, pages 33–47, 2008.

[4] Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy Synthesis for Multi-dimensional Quantitative Objectives. 11407: 1–27, 2012.

[5] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1982.

[6] A.E. Dalsgaard, S. Enevoldsen, P. Fogh, L.S. Jensen, T.S. Jepsen, I. Kaufmann, K.G. Larsen, S.M. Nielsen, M.C. Olesen, S. Pastva, and J. Srba. *Extended dependency graphs and efficient distributed fixed-point computation*, volume 10258 LNCS. 2017.

[7] Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jirí Srba. Energy games in multiweighted automata. In *ICTAC*, pages 95–115, 2011.

[8] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, pages 477–498. Springer, 1985.

[9] J. F. Jensen, K. G. Larsen, J. Srba, and L. K. Oestergaard. Efficient model checking of weighted CTL with upper-bound constraints. *STTT*, 18(4): 409–426, 2016.

[10] Line Juhl, Kim Guldstrand Larsen, and Jean-François Raskin. Optimal bounds for multiweighted and parametrised energy games. In *Theories of Programming and Formal Methods*, pages 244–255, 2013.

[11] Marcin Jurdziński, François Laroussinie, and Jeremy Sproston. *Model Checking Probabilistic Timed Automata with One or Two Clocks*, volume 4424 of *LNCS*, pages 170–184. Springer, 2007.

[12] Marcin Jurdziński, Ranko Lazić, and Sylvain Schmitz. Fixed-dimensional energy games are in pseudo-polynomial time. In *ICALP 2015*, volume 9135 of *LNCS*, pages 260–272. Springer, 2015.

[13] F Laroussinie, Ph Schnoebelen, and M Turuani. On the expressivity and complexity of quantitative branching-time temporal logics. Technical report, 2003.

[14] Kim G. Larsen, Radu Mardare, and Bingtian Xue. Alternation-free weighted mu-calculus: Decidability and completeness. *ENTCS*, 319:289 – 313, 2015.

[15] Xinxin Liu and Scott A. Smolka. Simple linear-time algorithms for minimal fixed points. In *ICALP 1998*, volume 1443 of *LNCS*, pages 53–66. Springer, 1998.

[16] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1):90–121, 1980.

[17] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

[18] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, POPL 1989, pages 179–190. ACM, 1989.

[19] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on Programming*, pages 337–351. Springer, 1982.

[20] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[21] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, second edition, 2006.

[22] Wolfgang Thomas. On the synthesis of strategies in infinite games. pages 1–13. 1995.