Graded CTL* over finite paths

Aniello Murano, Sasha Rubin, Loredana Sorrentino

Università degli Studi di Napoli Federico II

Abstract. In this paper we introduce the logic Graded Computation Tree Logic over finite paths (GCTL^*_f, for short), a variant of the Computation Tree Logic CTL^* in which path quantifiers are interpreted over finite paths and can count the number of such paths. State formulas of GCTL^*_f are interpreted over Kripke structures with a designated set of states, which we call "check points". These special states serve to delineate the end points of the finite executions. The syntax of GCTL^*_f has path quantifiers of the form $\mathtt{E}^{\geq g}\psi$ which express that there are at least g many distinct finite paths that a) end in a check point, and b) satisfy ψ. After defining and justifying the logic GCTL^*_f, we solve its model checking problem providing an upper bound on its computational complexity.

1 Introduction

Temporal logics are a special kind of modal logics in which the truth values of the assertions vary with time [24]. Introduced in the seventies, these logics provide a very useful framework for checking the reliability of reactive systems, i.e., systems that continuously interact with the external environment. In formal verification [8,9,21,25] to check whether a system meets a desired behaviour, we check, by means of a suitable algorithm, whether a mathematical model of the system satisfies a formal specification of the required behaviour, the latter usually given in terms of a temporal-logic formula.

Depending on the view of the underlying nature of the time, we distinguish between two types of temporal logics. In *linear-time temporal logics* (such as LTL [24]) each moment in time follows a *unique* possible future. On the other hand, in *branching-time temporal logics* (such as CTL [8] and $\mathrm{CTL}^\star$ [14]) each moment in time can be split into various possible futures. In order to express properties along one or all the possible futures, branching logics make use of existential and universal path quantifiers.

Driven by the need to capture some specific system requirements, several variants and extensions of classic temporal logics have been considered over the years. One direction concerns the use of *graded path modalities* in branching-time temporal logics [3,18,26,15,7,1,17,20,2,22]. These modalities allow us to express properties such as "there exists at least n possible paths that satisfy a formula" or "all but n paths satisfy a formula". In [5,6], the authors introduce the extension of CTL with graded modalities, namely GCTL. They prove that, although GCTL is more expressive than CTL, the satisfiability problem for GCTL remains solvable in ExpTime, even in the case the graded numbers are coded in binary.

In [16] the authors consider the model-checking problem using formulas expressed in GCTL and investigate its complexity: given a GCTL formula φ and a system model represented by a Kripke structure $\mathcal{K}$, the model-checking problem can be solved in time $(|\mathcal{K}| \cdot |\varphi|)$, that is the same running time required for CTL. In [2], the logic GCTL* was investigated. First, it was proved that it is equivalent, over trees, to *Monadic Path Logic*. Then, the authors turn to the satisfiability problem and show that it is 2ExpTime-complete. Finally, they show that the complexity of the model checking problem is PSpace-complete. So, for both decision problems we have the same complexity as for CTL*, although GCTL* is strictly more expressive.

Another meaningful direction concerning variations of classic temporal logics concerns the interpretation of formulas over *finite* paths [11,10,19,12,13]. The motivation for this is that many areas of Artificial Intelligence and Computer Science involve finite executions. A seminal work in this setting is [11], where the LTL logic framework was revisited under this assumption. In [11] it was proved that the resulting logic, namely LTL$_f$, has the expressive power of *First Order Logic*. Also, it was proved that satisfiability and model-checking are PSpace-complete, thus not harder than LTL. Branching-time temporal logics interpreted over finite paths were also introduced and studied in the literature [27,14]. Very recently, this was also investigated for logics of strategic reasoning [4].

Although the interpretation of graded formulas over finite computations seems natural and useful in practice, surprisingly this specific combination has never been addressed in the literature. In formal verification, such a formalism could be useful to accelerate the process of finding bad computations (similarly to what was done in [16] over infinite computations) or to check an unambiguous satisfaction of a property (similarly to what was done in [1] over infinite computations).

Our Contribution. In this paper we introduce a variant of Computation Tree Logic (CTL*), namely GCTL$^*_f$, in which path quantifiers $E^{\geq g}$ are graded and interpreted over finite paths. Thus this logic, with respect to CTL*, on the one hand, restricts the evaluation of formulas to finite paths and, on the other hand, it makes use of a grade g (a natural number or infinity), that is used along the path quantifier `E` and `A` in order to count paths. GCTL$^*_f$ formulas are interpreted over Kripke structures where additionally we make use of marked states which we call *check points*. As the name advocates, these states represent moments along a system computation in which we can check some specific requirements. Specifically, these states correspond to the unique moments in which the computation can end or should be inspected.

We address the model checking problem for GCTL$^*_f$ and solve it by means of an automata-theoretic approach.

Outline This paper is structured in the following way. In Section 2 we present some basic known concepts about automata and directed graphs that we use to solve our problem. In Section 3 we introduce and discuss the syntax and semantics of GCTL$^*_f$. In Section 4 we solve the model checking problem for GCTL$^*_f$ and provide an upper bound on its computational complexity. Finally, in Section 5 we summarize the obtained results and we give some possible future directions.

2 Preliminaries

In this section, we give some concepts about the graphs and the automata we are going to use along the paper in order to solve the model checking problem we address. As these are common definitions, an expert reader can skip this part.

Directed Graphs A graph $\mathcal{G}$ is an ordered pair (V, E) such that V is a finite set of vertices (also, named states nodes or points) and E is a set of edges (also, named arcs or lines), i.e., ordered pairs of elements of V.

Let $\mathcal{G}$ be a directed graph. A *path* in $\mathcal{G}$ is a sequence of m vertices $v_0, ..., v_{m-1}$ such that, for each $i = 1, ..., m-1$ we have that $(v_{i-1}, v_i) \in \mathrm{E}$. Moreover, we define the *length* of the path as the number m of vertices which constitute the path. Given two vertex v_a and v_b, we also say that v_b *is reachable* from v_a if there exists a path of length at least 1 from v_a to v_b. A path is called *simple* if no vertex appears more than once.

Nondeterministic Finite Word Automaton. A Nondeterministic Finite Word Automaton (NFW, for short) is a tuple $\mathcal{A} = (\mathrm{AP}, \mathrm{N}, \mathrm{I}, \delta, \mathrm{F})$ where

- AP is a finite non-empty set of atomic propositions;
- N is a finite non-empty set of states;
- $\mathrm{I} \subseteq \mathrm{N}$ is a non-empty set of initial states;
- $\delta : \mathrm{N} \times 2^{\mathrm{AP}} \to 2^{\mathrm{N}}$ is a transition function.
- $\mathrm{F} \subseteq \mathrm{N}$ is a set of final states.

Intuitively, $\delta(n, p)$ is the set of states that $\mathcal{A}$ can move into when it is in the state n and reads a subset of atoms $p \in 2^{\mathrm{AP}}$. The automaton $\mathcal{A}$ is *deterministic* (DFW, for short) if $|\mathrm{I}| = 1$ and $|\delta(n, p)| = 1$ for each state n and atom $p \in \mathrm{AP}$.

A *run* r of $\mathcal{A}$ on a word $w = p_0, p_1, ..., p_{m-1} \in (2^{\mathrm{AP}})^*$ is a sequence $n_0, n_1, ..., n_m$ of $m+1$ states in N such that $n_0 \in \mathrm{I}$ and $n_{i+1} = \delta(n_i, p_i)$ for $0 \leq i < m$. A nondeterministic automaton can have many runs of a given input word. The run r is accepting if $n_m \in \mathrm{F}$. The word w is accepted by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on w. The language of $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$.

3 Graded Computation Tree Logic over finite paths

In this section, we introduce the logic Graded Computation Tree Logic over finite paths (GCTL^*_f, for short), a variant of CTL^* in which the path quantifiers are graded and interpreted over finite paths. In particular, the syntax of GCTL^*_f is an adaptation of branching-time logic with the following main features. On the one hand it restricts $\mathrm{GCTL}^\star$ [5] by considering finite paths that end in check points, and on the other hand it extends $\mathrm{CTL}^\star$ [14] by means of grades $g \in \mathbb{N} \cup \{\infty\}$ applied to the existential path quantifier $\mathtt{E}$.

The obtained existential path operators $\mathtt{E}^{\geq g}\psi$ express that there are *at least* g distinct paths satisfying ψ. Just as for $\mathrm{CTL}^\star$, the syntax includes *path-formulas*, expressing properties of paths, and *state-formulas*, expressing properties of states.

Definition 1 (GCTL$_f^*$ syntax). GCTL$_f^*$ *formulas are inductively built from a set of atomic propositions* AP*, by using the following grammar:*

$$\phi := p \mid \neg\phi \mid \phi \wedge \phi \mid \mathtt{E}^{\geq g}\psi \mid \mathtt{E}^{\geq \infty}, \qquad \textit{where } p \in \mathrm{AP} \textit{ and } g \in \mathbb{N}$$
$$\psi := \phi \mid \neg\psi \mid \psi \wedge \psi \mid \mathtt{X}\psi \mid \psi \mathtt{U} \psi$$

All the formulas generated by a ϕ-rule are called *state-formulas*, while the formulas generated by a ψ-rule are called *path-formulas*. The *temporal operators* are `X` (read "next") and `U` (read "until"). The *path quantifier* is $E^{\geq g}$ (read "there exists at least g distinct paths..."). We introduce the following abbreviations: $\mathtt{A}^{<g}\psi = \neg E^{\geq g}\neg\psi$ (read "all but less than g paths satisfy ψ"), $\psi_1 \mathtt{R} \psi_2 = \neg(\neg\psi_1 \mathtt{U} \neg\psi_2)$ (read "release"), $\mathtt{F}\psi = \mathit{true}\mathtt{U}\psi$ (read "eventually"), $\mathtt{G}\psi = \mathit{false}\mathtt{R}\psi$ (read "globally"), and $\psi_1 \vee \psi_2 = \neg\psi_1 \wedge \neg\psi_2$ (read "or").

The semantics for GCTL$_f^*$ is defined *w.r.t.* a particular Kripke Structure in which there are special states which we call *check points*. It is a structure similar to a nondeterministic automaton and it is defined as follows.

Definition 2 (Kripke Structure System with check points). *A* Kripke Structure System with check points *(*KSc*, for short) is a tuple* $\mathcal{K} = \langle \mathrm{St}, \mathrm{AP}, \lambda, \delta, s_0, \mathrm{C}\rangle$ *such that:*

- St *is a finite non-empty set of states;*
- AP *is a set of atomic propositions;*
- $\lambda : \mathrm{St} \to 2^{\mathrm{AP}}$ *is the labeling function mapping each state to the atomic propositions true in that state;*
- $\delta \subseteq \mathrm{St} \times \mathrm{St}$ *is a transition relation;*
- $s_0 \in \mathrm{St}$ *is an initial state;*
- $\mathrm{C} \subseteq \mathrm{St}$ *is a set of states called* check points.

A *path* ρ in $\mathcal{K}$ is a finite (resp., infinite) sequence of states $\rho \in \mathrm{St}^*$ (resp., St^ω) such that, for all $i \in [0, |\rho| - 1[$ (resp., for all $i \in \mathbb{N}$) it holds that $(\rho_i, \rho_{i+1}) \in \delta$.

Also, we define $\mathrm{Pth}(\mathcal{K}) \subseteq \mathrm{St}^\omega \cup \mathrm{St}^*$ to denote the sets of paths π in $\mathcal{K}$ and we define $\mathrm{Pth}(s)$ to denote the set of paths starting from s. The first element of π is denoted by $\mathsf{fst}(\pi) \triangleq \pi_0$, and its last by $\mathsf{lst}(\pi)$. Furthermore, we write $(\pi)_i \triangleq \pi_i$ to denote the i-th element of π.

The semantics for GCTL$_f^*$ is defined *w.r.t.* KSc. The existential quantifiers $\mathtt{E}^{\geq g}\psi$ express that there are *at least* g distinct paths ending in check points that satisfy ψ. We distinguish finite paths $\pi_1, \pi_2 \in \mathrm{Pth}(\mathcal{K})$ of $\mathcal{K}$ in the natural way: two paths are *distinct* if $|\pi_1| \neq |\pi_2|$ or there exists an index $0 \leq i < |\pi_1|$ such that that $\pi_1(i) \neq \pi_2(i)$.

Definition 3 (GCTL$_f^*$ Semantics). *The semantics of* GCTL$_f^*$ *formulas is recursively defined as follows. For a Kripke Structure System with check points* KSc $\mathcal{K}$*, a state* s*, a path* π *and a natural number* $i \in \mathbb{N}$*, we have that:*

- *For state-formulas* ϕ*,* ϕ_1*, and* ϕ_2*:*
  - $\mathcal{K}, s \models p$ *if* $p \in \lambda(s)$*;*
  - $\mathcal{K}, s \models \neg\phi$ *if* $\mathcal{K}, s \not\models \phi$*;*

- $\mathcal{K}, s \models \phi_1 \wedge \phi_2$ *if both* $\mathcal{K}, s \models \phi_1$ *and* $\mathcal{K}, s \models \phi_2$*;*
- $\mathcal{K}, s \models \mathtt{E}^{\geq g}\psi$ *if there exists at least* g *distinct paths* π *in* $\mathrm{Pth}(s)$ *such that (a)* $\mathsf{lst}(\pi) \in \mathrm{C}$ *and (b)* $\mathcal{K}, \pi, 0 \models \psi$*;*
- $\mathcal{K}, s \models \mathtt{E}^{\geq \infty}\psi$ *if there exists infinitely many distinct paths* π *in* $\mathrm{Pth}(s)$ *such that (a)* $\mathsf{lst}(\pi) \in \mathrm{C}$ *and (b)* $\mathcal{K}, \pi, 0 \models \psi$*;*

– *For path-formulas* ϕ*,* ψ*,* ψ_1*, and* ψ_2*:*

- $\mathcal{K}, \pi, i \models \phi$ *if* $\mathcal{K}, (\pi)_i \models \phi$*;*
- $\mathcal{K}, \pi, i \models \neg\psi$ *if* $\mathcal{K}, \pi, i \not\models \psi$*;*
- $\mathcal{K}, \pi, i \models \psi_1 \wedge \psi_2$ *if both* $\mathcal{K}, \pi, i \models \psi_1$ *and* $\mathcal{K}, \pi, i \models \psi_2$*;*
- $\mathcal{K}, \pi, i \models \mathtt{X}\psi$ *if* $i + 1 < |\pi|$ *and* $\mathcal{K}, \pi, i + 1 \models \psi$*;*
- $\mathcal{K}, \pi, i \models \psi_1 \mathtt{U} \psi_2$ *if there exists* $k \in \mathbb{N}$ *such that* $\mathcal{K}, \pi, i + k \models \psi_2$ *and* $\mathcal{K}, \pi, i + j \models \psi_1$*, for all* $j \in [0, k[$*;*

We say that π *satisfies the path formula* ψ *over* $\mathcal{K}$*, and write* $\mathcal{K}, \pi \models \psi$*, if* $\mathcal{K}, \pi, 0 \models \psi$*. Also, we say that* $\mathcal{K}$ *satisfies the state formula* ϕ*, and write* $\mathcal{K} \models \phi$*, if* $\mathcal{K}, s_0 \models \phi$*. We call a path formula without a path quantifier a* flat path formula*.*

Note that a finite path π satisfies $\mathtt{X}\psi$ at position i implies, in particular, that i is not the last position in π.

Remark 1. The logic CTL^*_f introduced in [4] is similar to GCTL^*_f except that it does not have graded quantifiers, only $\mathtt{E}$. That logic is a fragment of ours, simply restrict to formulas in which every degree g is equal to 1.

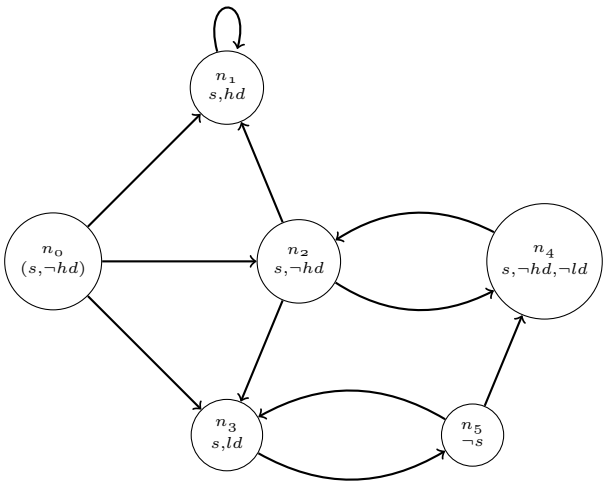

Fig. 1: Example of a KSc $\mathcal{K}$ that models a map in a military scenario.

Example 1. As an example we consider the KSc $\mathcal{K}$ depicted in Figure 1. It models a map available to a military aircraft-pilot carrying relief supplies and food. The states $\mathrm{St} = \{n_0, n_1, n_2, n_3, n_4, n_5\}$ correspond to different places, and the only checkpoint is n_4 (so $\mathrm{C} = \{n_4\}$) and corresponds to a neutral zone. The initial state is n_0. Each state of $\mathcal{K}$ is labeled by a set of atomic proposition where $\mathrm{AP} = \{\mathrm{s}, \mathrm{hd}, \mathrm{ld}\}$ and s stands for "safe", hd stands for "high danger" and ld stands for "low danger". They indicate the degree of danger in crossing that area. In detail, the places are labeled as follows: $\lambda(n_0) = \{s, \neg\mathrm{hd}\}$, $\lambda(n_1) = \{\neg s, \mathrm{hd}\}$, $\lambda(n_2) = \{s, \neg\mathrm{hd}\}$, $\lambda(n_3) = \{s, \mathrm{ld}\}$, $\lambda(s_4) = \{s, \neg\mathrm{hd}, \neg\mathrm{ld}\}$ and $\lambda(n_5) = \{\neg s\}$. The GCTL_f^* formula $\phi = \mathtt{E}^{\geq 2}\psi$ where $\psi = \mathtt{F}(\mathrm{s} \wedge \neg\mathrm{ld} \wedge \neg\mathrm{hd})$ holds in the given KSc since there are many distinct paths (at least two) that satisfy the formula. In particular, all paths matching the regular expression

$$n_0 n_2 n_4 + n_0 (n_3 n_5)^* n_4 + n_0 n_2 (n_3 n_5)^* n_4$$

satisfy ψ. Instead, if we use the GCTL_f^* formula $\phi = \mathtt{E}^{\geq 2}\psi$ where $\psi = \neg\mathrm{hd}\ \mathtt{U}\ (\mathtt{F}\ (\mathrm{s} \wedge \neg\mathrm{ld} \wedge \neg\mathrm{hd}))$ we have that the formula does not hold since there is only a single path satisfying ψ, i.e., $n_0 n_2 n_4$.

4 Model Checking

In this section, we solve the model checking GCTL_f^* and we provide an upper bound on its computational complexity. First, we formulate the decision problem as follows.

Definition 4 (Model Checking). *Given a* KSc $\mathcal{K}$ *and a* GCTL_f^* *formula* ϕ *decide if* $\mathcal{K} \models \phi$.

In order to measure the complexity, we need to define the size of the input $\mathcal{K}$ and ϕ. The *size of a structure* $\mathcal{K}$ is its number of states, and the *size of a formula* ϕ is defined as usual, except that $|\mathtt{E}^{\geq g}\psi| = 1 + |g| + |\psi|$ for $g \neq \infty$, and $|\mathtt{E}^{\geq \infty}\psi| = 1 + |\psi|$, i.e., the grades are represented in unary.

Here is the main theorem of this section:

Theorem 1. *Model checking* GCTL_f^* *is in ExpSpace.*

We establish this upper bound as follows. To check whether $\mathcal{K} \models \phi$ we mark each state of $\mathcal{K}$ by the state sub-formulas of ϕ true at that state, see Algorithm 3. In order to deal with formulas of the form $E^{\geq g}\psi$, we view ψ as an LTL formula over the marked Kripke structure and call Algorithm 2. This algorithm converts ψ into an DFW accepting the models of ψ, then builds the product with the Kripke structure resulting in a graph with a single initial state and a set of target vertices. We count the number of paths from the initial vertices to the target vertices using classical graph reachability algorithms, see Algorithm 1. The product graph is double-exponential in $|\phi|$ and polynomial in the Kripke structure. However, the counting can be done in NLogSpace in the product graph and polynomial space in g. Thus we obtain that the model checking problem GCTL_f^* is in ExpSpace.

Here are the details. We begin by describing the NLogSpace for counting.

Algorithm 1 ExistsPaths

function ExistsPaths($\mathcal{G}, s, \mathrm{T}, g$) ▷ Returns YES if there is at least g distinct paths from s to the vertices in T.
  if there exists a vertex $t \in \mathrm{T}$ and a vertex v such that $Reach(\mathcal{G}, s, v)$ and $Reach(\mathcal{G}, v, v)$ and $Reach(\mathcal{G}, v, t)$ **then**
    return YES ▷ There are infinitely many paths
  end if
  if $g = \infty$ **then**
    return NO
  end if ▷ There are finitely many paths and $g < \infty$
  for all $i \leq g$ **do** ▷ Initialise g many paths
    $cur_i \leftarrow s$
    $fin_i \leftarrow 0$ ▷ $fin_i = 0$ means the ith path is not finished
  end for
  for all $i \neq j \leq g$ **do** ▷ $diff_{i,j} = 1$ means the indexed paths are different
    $diff_{i,j} = 0$
  end for
  for $steps = 1$ to number of vertices **do** ▷ We are testing for simple paths
    for all i **do**
      if fin_i=0 and $cur_i \in \mathrm{T}$ **then**
        non-deterministically let $fin_i \in \{0, steps\}$
      end if ▷ $fin_i \neq 0$ represents the length of the ith path
      if $fin_i = 0$ **then**
        replace cur_i by an element of $Adj(cur_i)$ chosen non-deterministically
      end if
    end for
    for all $i \neq j \leq g$ **do**
      If $cur_i \neq cur_j$ then $diff_{i,j} \leftarrow 1$
    end for
  end for
  if $\bigwedge_i cur_i \in \mathrm{T}$ and $\bigwedge_{i \neq j}(fin_i = fin_j \rightarrow diff_{i,j} = 1)$ **then**
    return YES
  else
    return NO
  end if
end function

Proposition 1 (Counting Paths). *The Algorithm $ExistsPaths(\mathcal{G}, s, \mathrm{T}, g)$ in Algorithm 1 decides if there exist at least g distinct paths in a graph $\mathcal{G}$, each of which starts with s and ends with some vertex in* T*. It works in NLogSpace in $\mathcal{G}$ and PSpace in g.*

Proof. Consider the first condition. It asks if there exists a vertex v such that the following properties are verified: (a) $Reach(\mathcal{G}, s, v)$ is *true* (b) $Reach(\mathcal{G}, v, v)$ is *true* and (c) $Reach(\mathcal{G}, v, t)$ is *true*. This just says that there is a path between s and some element of T in which some vertex is traversed more than once. But since the graph is finite, this is equivalent to the property that there are infinitely many paths from s to T. If this is the case we return YES no matter the value of g. Otherwise, we proceed with two cases.

If $g = \infty$ we return NO. If $g \neq \infty$ the rest of the algorithm checks if there are at least g *simple* paths from s to T. Note that this is sufficient because if there are at least g, not necessarily simple, paths from s to T then there are infinitely many paths from s to T (just repeat the segment between two occurrences of the same vertex), which we know there are not since the first condition failed.

The rest of the algorithm uses the following terminology: for a vertex v we let $Adj(v)$ denote the set of vertices v' such that $(v, v') \in \mathrm{E}$. The algorithm uses the following variables: an integer variable *steps* (which is used to count the number of traversed vertices); variables cur_i (for $i \leq g$) that vary over states of $\mathcal{G}$ (and are used to store the current vertices of paths), variables $diff_{i,j}$ (for $i, j \leq g$ with $i \neq k$) that take the value 1 once a difference is found in paths of equal length), and fin_i for $i \leq g$ which takes value 0 if the ith path is not finished, and value $fin_i \neq 0$ if the length of the path is fin_i. The algorithm runs for as many steps as there are vertices in the graph. It consists of three phases: it first guesses if a given path ending in T finishes (it may continue and finish at a later time), and if so it records this fact in fin_i; it then guesses the next vertex of all paths that have not finished; it then records if it finds a difference in the paths. The return condition says that each path should end in T and if two paths are the same length then they should be different. □

The Algorithm 2, presented below, shows how to model flat path formulas.

Algorithm 2 ModelCheckingFlat

1: **function** ModelCheckingFlat($\mathcal{K}, \psi, g$) ▷ Returns the set of states s such that $(\mathcal{K}, s) \models \mathtt{E}^{\geq g}\psi$. ▷ Here ψ is a flat path-formula
2: Convert ψ into an equivalent DFW($\mathrm{N}, i, \Delta, \mathrm{C}'$).
3: Form the product graph $\mathcal{G}$ of KSc and the DFW. It has vertex set $\mathrm{St} \times N$.
4: Return the set of states s such that $ExistsPaths(\mathcal{G}, (s, i), \mathrm{C} \times \mathrm{C}', g)$
5: **end function**

Proposition 2 (Flat Formulas). *Model Checking* CTL^*_f *formulas of the form* $\mathtt{E}^{\geq g}\psi$ *where ψ is a flat path formula is in ExpSpace in the structure and PSpace in g.*

Proof. For the second step of the Algorithm 2 convert the formula into an NFW using an adaptation of the classic Vardi-Wolper construction for finite words[28,11]. The NFW accepts all paths (defined over the atomic propositions AP) that satisfy the formula ψ. The number of states of the NFW is at most exponential in the size of the formula. To finish the second step of the algorithm, determinise this NFW to get a DFW. This step also costs an exponential.

In the third step of the algorithm, we create a new graph in which the set of vertex is given by the product of the set of states St of the KSc $\mathcal{K}$ and the set of states of the DFW $\mathcal{A}$. There is an edge $(s, n) \to (s', n')$ if $\delta(s, s')$ and $\Delta(n, \lambda(s)) = n'$. Finally, in the last step, thanks to the algorithm *ExistsPaths* we verify in LOGSPACE (in the size of graph), for each state s of KSc, if there exists at least g distinct paths starting from the vertex of (s, i) (thus the DFW is also in its initial state) and ending in a vertex of the form (s, n) where n is a final state of the DFW. These paths correspond, exactly, to the paths in KSc that satisfy ψ. Since the graph is of double exponential size in $|\psi|$, the whole algorithm runs in exponential space. □

In Algorithm 3 we give our algorithm for model checking arbitrary formulas. It uses the following notions.

A formula φ is a *maximal state-subformula* of ϕ if φ is a state-subformula of ϕ and φ is not a proper sub-formula of any other state sub-formula of ϕ. Every flat path formula ψ of GCTL^*_f formula ϕ can be viewed as an LTL formula Ψ whose atoms are elements of a maximal state-subformula ϕ as is usual for branching-time logics, e.g., see [21].

Algorithm 3 ModelChecking

function ModelChecking($\mathcal{K}, \phi$)▷ Returns the set of states s for which $\mathcal{K}, s \models \phi$
  Introduce a new atom p_ϕ
  if $\phi = p \in$ AP **then**
    return the set of states s such that $p \in \lambda(s)$.
  end if
  if $\phi = \neg\phi_1$ **then**
    return the set of states s such that $s \notin$ ModelChecking($\mathcal{K}, \phi_1$).
  end if
  if $\phi = \phi_1 \wedge \phi_2$ **then**
    return the set of states s such that $s \in \bigcap_{i=1,2}$ ModelChecking($\mathcal{K}, \phi_i$).
  end if
  if $\phi = \mathbf{E}^{\geq g}\psi$ **then**
    for all maximal state-subformulas φ of ψ **do**
      Introduce a fresh atom p_φ
      Redefine λ so that $p_\varphi \in \lambda(s)$ iff $s \in$ ModelChecking($\mathcal{K}, \varphi$).
      Replace every occurrence of φ in ψ by p_φ.
    end for
    return the set of states s such that $s \in$ ModelCheckingFlat($\mathcal{K}, \psi, g$).
  end if
end function

Theorem 2. *The Model Checking Problem of* GCTL^*_f *is in ExpSpace.*

Proof. To solve the model checking problem we give the Algorithm 3, named $ModelChecking(\mathcal{K}, \phi)$ that processes the state sub-formulas φ of ϕ, starting from the innermost one and, for each state s decides if $(\mathcal{K}, s) \models \phi$ holds. The atomic case and the Boolean operations are immediate. For the path quantifier $\mathtt{E}^{\geq g}\psi$ the algorithm relabels each state by a fresh atom p_φ iff the maximal state sub-formula φ of ψ holds in that state. This is done recursively. It then treats ψ as flat path formula by replacing each φ by p_φ. It then calls $ModelCheckingFlat$ with the new relabeled $\mathcal{K}$, the flat formula ψ and the given graded g. □

5 Conclusion and Future Work

Recently, temporal logic formalisms restricted to finite computations have received large attention in formal system verification. This concept is very important in many areas of Artificial Intelligence. For example, one may think of business processes that are modelled using finite path, or to automated planning in which the executions are often finite. In this paper we have introduced a variant of $\mathrm{CTL}^\star$, namely GCTL^*_f, in which the formulas are interpreted over finite paths that can be selected by the logic by means of a graded modality. We have addressed the model checking problem for GCTL^*_f and proved it to be in ExpSpace. We believe that one can refine the approach, or use partitioning tree automata [5], and get a better complexity. The known lower bounds is for the fragment CTL^*_f, i.e., PSpace [4].

Besides that, this articles opens to several direction for future work. First, we recall that graded modalities have been studied also in the context of the modal μ-calculus, with and without backwards modalities [7]. It would be worth reconsider that logic under the finite path semantics as we have done in this paper. Another interesting direction would be to consider enriching GCTL^*_f with knowledge operators. This would allows us to talk about a finite amount of knowledge (but unbounded) along paths. Also, recent work shows how to count the number of strategies in a graph game [23,4], and extending our work to count strategies in the finite-trace case is of interest.

Finally, note that we have assumed in this paper that graded numbers used along formulas are coded in unary. By using a binary coding we immediately loose an exponent along the model checking procedure. We leave open the question whether this blow-up is avoidable as done in [6].

References

1. Benjamin Aminof, Vadim Malvone, Aniello Murano, and Sasha Rubin. Graded modalities in strategy logic. *Information and Computation*, 261:634 – 649, 2018. 4th International Workshop on Strategic Reasoning (SR 2016).
2. Benjamin Aminof, Aniello Murano, and Sasha Rubin. CTL* with graded path modalities. *Information and Computation*, 2018.

3. Everardo Bárcenas, Edgard Benítez-Guerrero, and Jesús Lavalle. On the model checking of the graded μ-calculus on trees. In Grigori Sidorov and Sofía N. Galicia-Haro, editors, *Advances in Artificial Intelligence and Soft Computing - 14th Mexican International Conference on Artificial Intelligence, MICAI 2015, Cuernavaca, Morelos, Mexico, October 25-31, 2015, Proceedings, Part I*, volume 9413 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2015.
4. Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. Alternating-time temporal logic on finite traces. In *International Joint Conference on Artificial Intelligence*, pages 77–83, 2018.
5. A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. In *Logic in Computer Science'09*, pages 342–351. IEEE Computer Society, 2009.
6. A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. *Transactions On Computational Logic*, 13(3):25:1–53, 2012.
7. P.A. Bonatti, C. Lutz, A. Murano, and M.Y. Vardi. The Complexity of Enriched muCalculi. *Logical Methods in Computer Science*, 4(3):1–27, 2008.
8. E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs'81*, LNCS 131, pages 52–71. Springer, 1981.
9. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2002.
10. Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, pages 1027–1033, 2014.
11. Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 854–860. IJCAI/AAAI, 2013.
12. Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1558–1564. AAAI Press, 2015.
13. Giuseppe De Giacomo and Moshe Y. Vardi. LTL_f and LDL_f synthesis under partial observability. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1044–1050. IJCAI/AAAI Press, 2016.
14. E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time. *Journal of the ACM*, 33(1):151–178, 1986.
15. A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *Logical Methods in Computer Science*, 4(3):1–21, 2008.
16. A. Ferrante, M. Napoli, and M. Parente. Model Checking for Graded CTL. *Fundamenta Informaticae*, 96(3):323–339, 2009.
17. M. Kaminski, S. Schneider, and G. Smolka. Terminating tableaux for graded hybrid logic with global modalities and role hierarchies. *LMCS*, 7(1), 2011.
18. Yevgeny Kazakov and Ian Pratt-Hartmann. A note on the complexity of the satisfiability problem for graded modal logics. In *Symposium on Logic in Computer Science*, pages 407–416, 2009.
19. Jeremy Kong and Alessio Lomuscio. Model checking multi-agent systems against LDLK specifications on finite traces. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 166–174. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

20. O. Kupferman, U. Sattler, and M.Y. Vardi. The Complexity of the Graded mu-Calculus. In *Conference on Automated Deduction'02*, LNCS 2392, pages 423–437. Springer, 2002.
21. O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.
22. Vadim Malvone, Fabio Mogavero, Aniello Murano, and Loredana Sorrentino. Reasoning about graded strategy quantifiers. *Information and Computation*, 259:390 – 411, 2018. 22nd International Symposium on Temporal Representation and Reasoning.
23. Vadim Malvone, Aniello Murano, and Loredana Sorrentino. Additional winning strategies in reachability games. *Fundam. Inform.*, 159(1-2):175–195, 2018.
24. A. Pnueli. The Temporal Logic of Programs. In *Foundation of Computer Science'77*, pages 46–57. IEEE Computer Society, 1977.
25. J.P. Queille and J. Sifakis. Specification and Verification of Concurrent Programs in Cesar. In *Symposium on Programming'81*, LNCS 137, pages 337–351. Springer, 1981.
26. S. Tobies. PSPACE Reasoning for Graded Modal Logics. *Journal of Logic and Computation*, 11(1):85–106, 2001.
27. M. Y. Vardi and L. Stockmeyer. Lower bound in full (2EXPTIME-hardness for CTL-SAT). 1985.
28. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994.