Graded CTL* over finite paths

Aniello Murano, Sasha Rubin, Loredana Sorrentino

Università degli Studi di Napoli Federico II

Abstract. In this paper we introduce Graded Computation Tree Logic over finite paths (GCTL^*_f, for short), a variant of 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 and establish that its computational complexity is PSPACE-complete.

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 [7,8,21,25] 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 [7] and $\mathrm{CTL}^\star$ [13]) 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 [3,18,26,14,6,1,17,20,2,22]. 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], 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 [10,9,19,11,12]. The motivation for this is that many areas of Artificial Intelligence and Computer Science involve finite executions. A seminal work in this setting is [10], where the LTL logic framework was revisited under this assumption. In [10] 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 [27,13]. 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 [15,16] for infinite computations) or to check an unambiguous satisfaction of a property (similarly to what was done in [1] for 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. The difference with CTL* is that we restrict the evaluation of formulas to finite paths and, that it makes use of a grade g (a natural number or infinity) that is coupled with the path quantifiers `E` and `A` in order to count paths. GCTL$^*_f$ formulas are interpreted over Kripke structures that are additionally annotated by marked states which we call *check points*. As the name advocates, these states represent moments along a system computation that should be inspected (by the formula). We address the computational complexity of the model-checking problem for GCTL$^*_f$. The complexity is PSpace-hard, a fact that already holds for the fragment of formulas in which $g = 1$ [4]. On the other hand, we provide a matching upper-bound using a mix of automata-theory and nondeterministic algorithms.

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 provide a PSpace algorithm for the model checking problem of GCTL$^*_f$. Finally, in Section 5 we give some possible future directions.

2 Preliminaries

In this section, we provide basic concepts about graphs and automata that we will use. As these are common definitions, an expert reader can skip this part.

Directed Graphs A *graph* $\mathcal{G}$ is pair (V, E) where V is a finite set of vertices (also, called states nodes or points) and $\mathrm{E} \subseteq \mathrm{V} \times \mathrm{V}$ is a set of edges (also, called arcs or lines). A *path* in $\mathcal{G}$, of *length* m, is a sequence of m vertices $v_0, ..., v_{m-1}$ such that, for each $i = 1, ..., m-1$ we have that $(v_{i-1}, v_i) \in \mathrm{E}$. Given two vertices 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 that starts with v_a and ends with 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, $\delta(s, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in the state s and reads a subset of atoms $\sigma \in 2^{\mathrm{AP}}$. The automaton $\mathcal{A}$ is *deterministic* (DFW, for short) if $|\mathrm{I}| = 1$ and $|\delta(s, \sigma)| = 1$ for each state s and input $\sigma \in 2^{\mathrm{AP}}$.

A *run* r of $\mathcal{A}$ on a word $w = \sigma_0, \sigma_1, ..., \sigma_{m-1} \in (2^{\mathrm{AP}})^*$ is a sequence $s_0, s_1, ..., s_m$ of $m+1$ states (i.e., elements of N) such that $s_0 \in \mathrm{I}$ and $s_{i+1} = \delta(s_i, \sigma_i)$ for $0 \leq i < m$. A run r is *accepting* if $s_m \in \mathrm{F}$. A 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 $\mathcal{A}$.

The *size* of an NFW $\mathcal{A}$, denoted $|\mathcal{A}|$, is the cardinality of N.

3 Graded Computation Tree Logic over finite paths

In this section, we introduce 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 $\mathrm{CTL}^\star$ [13] by means of grades $g \in \mathbb{N} \cup \{\infty\}$ applied to the existential path quantifier $\mathtt{E}$.

The obtained existential path operators $\mathtt{E}^{\geq g}\psi$ expresses 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 $\mathtt{X}$ (read "next") and $\mathtt{U}$ (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 $\psi_1 \vee \psi_2 = \neg(\neg\psi_1 \wedge \neg\psi_2)$ (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* $\mathcal{K} = \langle \mathrm{St}, \mathrm{AP}, \lambda, \tau, s_0, \mathrm{C} \rangle$ *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;*
- $\tau \subseteq \mathrm{St} \times \mathrm{St}$ *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 $(\rho_i, \rho_{i+1}) \in \tau$. 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 they have different lengths, or if they have the same length, say m, and there exists an index $0 \leq i < m$ such that that $\pi_1(i) \neq \pi_2(i)$.

Definition 3 (GCTL^*_f Semantics). *The semantics of* GCTL^*_f *formulas is recursively defined for a* KSc $\mathcal{K}$*, a state* s*, a path* π *and a natural number* $i \in \mathbb{N}$*.*

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. To see this, simply restrict our logic to formulas in which every degree g is equal to 1.

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 define the decision problem.

Definition 4. *The model-checking problem for* GCTL^*_f *is the following: 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.

The main result of this section is that model checking GCTL^*_f is in PSpace. Since model-checking CTL^*_f is PSpace-hard [4], we get that model-checking GCTL^*_f is PSpace-complete.

Theorem 1. *The Model Checking Problem of* GCTL^*_f *is in* PSpace*.*

To prove this, we first provide an algorithm (Algorithm 1) that processes the state sub-formulas φ of ϕ, starting from the innermost one and, for each state s decides if $(\mathcal{K}, s) \models \varphi$ holds. The algorithm 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 whose atoms

Algorithm 1 ModelChecking

```
1: function ModelChecking(K, φ) ▷ Returns the set of states s for which K, s ⊨ φ
2:     Introduce a new atom p_φ
3:     if φ = p ∈ AP then
4:         return the set of states s such that p ∈ λ(s).
5:     end if
6:     if φ = ¬φ_1 then
7:         return the set of states s such that s ∉ ModelChecking(K, φ_1).
8:     end if
9:     if φ = φ_1 ∧ φ_2 then
10:        return the set of states s such that s ∈ ⋂_{i=1,2} ModelChecking(K, φ_i).
11:    end if
12:    if φ = E^{≥g} ψ then
13:        for all maximal state-subformulas φ of ψ do
14:            Introduce a fresh atom p_φ
15:            Redefine λ so that p_φ ∈ λ(s) iff s ∈ ModelChecking(K, φ).
16:            Replace every occurrence of φ in ψ by p_φ.
17:        end for
18:        Convert ψ into an equivalent NFW N.
19:        Return the set of states s such that ExistsPaths(K, s_0, N, g)
20:    end if
21: end function
```

are elements of a maximal state-subformula ϕ as is usual for branching-time logics, e.g., see [21].

Looking at Algorithm 1, we see that 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. In Line 18, it converts ψ into an NFW accepting all strings (over the alphabet 2^{AP}) that satisfy the formula ψ. This can be done using an adaptation of the classic Vardi-Wolper construction ([28]) for finite words [10]. In the last step, thanks to the algorithm *ExistsPaths* (Algorithm 2) we verify (in NLogSpace in $|\mathcal{K}|$ and $|\mathcal{N}|$ and PSpace in g) if there exist at least g distinct paths in $\mathcal{K}$, each of which starts with s and ends with some vertex in C, each one labeling a word accepted by $\mathcal{N}$. Note that the NFW may be of exponential size, but it can be built on-the-fly, i.e., there is a PSpace algorithm that computes the transition function of $\mathcal{N}$ (which is needed in Line 8 of Algorithm 2).

We now describe the Algorithm 2 for counting paths. 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 idea of the algorithm is to guess g many different paths in $\mathcal{K}$. It does this step-by-step, only storing the last state of each path. Note that to ease readability of the algorithm, there are implicit loops over the indices $i, j \leq g$, e.g., in line 2, we set $Kcur_i$ to be s for every i.

Algorithm 2 ExistsPaths

1: **function** ExistsPaths($\mathcal{K}, s, \mathcal{N}, g$) ▷ Returns YES if there are at least g distinct paths in $\mathcal{K} = \langle \mathrm{St}, \mathrm{AP}, \lambda, \tau, s, \mathrm{C} \rangle$ from $s \in \mathrm{St}$ to the vertices in C, each one labeling a word accepted by $\mathcal{N} = (AP, N, I, \delta, F)$.
2: $Kcur_i \leftarrow s$ ▷ $\forall i : i \leq g$
3: assign $Ncur_i \in I$ non-deterministically ▷ $\forall i : i \leq g$
▷ $ended_i = \bot$ indicates that the current element i is not ended
4: $ended_i = \bot$ ▷ $\forall i : i \leq g$
▷ $Kdiff_{i,j} = \bot$ indicates that the two paths are not different
5: $Kdiff_{i,j} = \bot$ ▷ $\forall i, j : i \neq j \leq g$
6: **while** `true` **do**
7: **if** $\neg ended_i$ **then** ▷ $\forall i : i \leq g$
8: Replace $Ncur_i$ by an element of $\delta(Ncur_i, \lambda(Kcur_i))$ ▷ nondet
9: **end if**
10: **if** $\neg ended_i$ and $Kcur_i \in$ C and $Ncur_i \in$ F **then** ▷ $\forall i : i \leq g$
11: assign $ended_i \in \{\bot, \top\}$ non-deterministically
12: **end if**
13: **if** $\neg ended_i$ and $ended_j$ **then** ▷ $\forall i, j : i \neq j \leq g$
14: $Kdiff_{i,j} \leftarrow \top$ ▷ If one ends now but the other has not ended
15: **end if**
16: **if** $\bigwedge_i ended_i$ and $\bigwedge_{i \neq j \leq g} Kdiff_{i,j}$ **then**
17: **Return YES**
18: **end if**
19: **if** $\neg ended_i$ **then** ▷ $\forall i : i \leq g$
20: Replace $Kcur_i$ by an element of $Adj(Kcur_i)$ ▷ nondet
21: **end if**
22: **if** $\neg ended_i$ and $\neg ended_j$ and $Kcur_i \neq Kcur_j$ **then** ▷ $\forall i, j : i \neq j \leq g$
23: $Kdiff_{i,j} \leftarrow \top$ ▷ If both have not ended, but they have different states
24: **end if**
25: **end while**
26: **end function**

Proposition 1 (Counting Paths). *The Algorithm ExistsPaths*$(\mathcal{K}, s, \mathcal{N}, g)$ *decides if there exist at least* g *distinct paths in* $\mathcal{K}$*, each of which starts with* s *and ends with some vertex in* C*, each one labeling a word accepted by* $\mathcal{N}$*. It works in* NLogSpace *in* $|\mathcal{K}|$ *and* $|\mathcal{N}|$*, and* PSpace *in* g*.*

Proof. The variables $Kcur_i$ are used in order to store the last state in $\mathcal{K}$; they are initialised to s in line 2, and each path is advanced by one state in line 20 (*i.e.*, if the path is not ended). The algorithm simultaneously also guesses g runs in $\mathcal{N}$, one for each of the guessed paths in $\mathcal{K}$. The variable $Ncur_i$ are used to store the last state of the ith guessed path in $\mathcal{N}$; it is initialised in line 3 by non-deterministically assigning it an element of the set I of initial states, and it is updated in line 8. In line 10, the algorithm also guesses when a given path in $\mathcal{K}$ finishes, and remembers this fact by setting the flag $ended_i$ to true. These flags initially are set to false in order to indicate that the ith path has not ended. The algorithm updates the variable $Kdiff_{i,j}$, initially set to false (indicating

that the two paths are not different) when it sees that the ith and jth path are different. This happens in one of two cases: (1) in some step, the ith path ended and the jth path did not (line 13), or (2) in some step, the current state of the ith path and jth path are different (line 22). Thus, if the algorithm returns YES (line 17) then there are g distinct paths in $\mathcal{K}$, that (by guard 10) each end in C and label words accepted by $\mathcal{N}$. On the other hand, suppose there exists g different sequences in $\mathcal{K}$, each ending in C, and g corresponding sequences in $\mathcal{N}$ whose states are labeled by a word accepted by the automaton. Then we can use these paths to resolve the nondeterminism in lines 3, 8, 11 and 20, so that the algorithm returns YES.

Regarding the space complexity, first note that the algorithm is nondeterministic. Now, to store a vertex of a graph with N vertices we need logspace in N. The algorithm requires to store $O(g)$ many such vertices. It also requires $O(g^2)$ many boolean variables (for the variables $ended_i$ and $Kdiff_{i,j}$). □

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 introduced a variant of CTL*, 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 addressed the model checking problem for GCTL$^*_f$ and proved it to be PSPACE-complete.

Besides that, this articles opens 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 [6]. It would be worth reconsidering 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. 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 in the complexity of the model checking procedure. We leave open the question of whether this blow-up is avoidable (cf. [5]).

References

1. Benjamin Aminof, Vadim Malvone, Aniello Murano, and Sasha Rubin. Graded modalities in strategy logic. *Information and Computation*, 261:634 – 649, 2018.
2. Benjamin Aminof, Aniello Murano, and Sasha Rubin. CTL* with graded path modalities. *Information and Computation*, 262 (Part 2):1 – 21, 2018.

3. Everardo Bárcenas, Edgard Benítez-Guerrero, and Jesús Lavalle. On the model checking of the graded μ-calculus on trees. In *MICAI 2015*, 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. *Transactions On Computational Logic*, 13(3):25:1–53, 2012.
6. 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.
7. 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.
8. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2002.
9. Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, pages 1027–1033, 2014.
10. Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI 2013*, pages 854–860. IJCAI/AAAI, 2013.
11. Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In Qiang Yang and Michael Wooldridge, editors, *IJCAI 2015*, pages 1558–1564. AAAI Press, 2015.
12. Giuseppe De Giacomo and Moshe Y. Vardi. $\mathrm{LTL_f}$ and $\mathrm{LDL_f}$ synthesis under partial observability. In Subbarao Kambhampati, editor, *IJCAI 2016*, pages 1044–1050. IJCAI/AAAI Press, 2016.
13. 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.
14. A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *Logical Methods in Computer Science*, 4(3):1–21, 2008.
15. A. Ferrante, M. Napoli, and M. Parente. Graded-ctl: Satisfiability and symbolic model checking. In *ICFEM 2009*, volume LNCS 5885, pages 306–325, 2009.
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 *AAMAS 2018*, pages 166–174. IFAAMAS/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.
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.