Graded-CTL: Satisfiability and Symbolic Model Checking*

Alessandro Ferrante[1], Margherita Napoli[2], and Mimmo Parente[2]

[1] Embedded Systems Research Unit, Bruno Kessler Foundation, IRST
Via Sommarive, 18, 38123, Povo (TN), Italy
ferrante@fbk.eu

[2] Dip.to di Informatica ed Applicazioni "R.M. Capocelli", Università of Salerno
Via Ponte don Melillo, 84084, Fisciano (SA), Italy
{napoli,parente}@dia.unisa.it

Abstract. In this paper we continue the study of a strict extension of the Computation Tree Logic, called *graded-CTL*, recently introduced by the same authors. This new logic augments the standard quantifiers with graded modalities, being able thus to express "There exist at least k" or "For all but k" futures, for some constant k. One can thus describe properties useful in system design, which cannot be expressed with CTL, like a sort of redundant liveness property asking whether there is more than one path satisfying that "something good eventually happens", making thus the system more tolerant to possible faults. Graded-CTL formulas can also be used to determine whether there are more than a given number of bad behaviors of a system: this, in the model-checking framework, means that one can verify the existence of a user-defined number of counterexamples for a given specification and generate them, in a unique run of the model-checker.

Here we show both theoretical and applicative contributions. On the theoretical side we give a simple algorithm to *decide* this logic, and we prove that the satisfiability problem is ExpTime-complete when the constants of the quantifiers are represented in unary. On the applicative side we propose *symbolic* algorithms to solve the model checking problem. One of the main characteristics of these algorithms is that, though the computation of "distinct" counterexamples has inherently high complexity when the model is represented symbolically, we have designed them to make the generation of multiple counterexamples as easy and quick as possible. The symbolic algorithms have been implemented using BDD data structures, and have been integrated into the well known NuSMV model checker, that has been modified to accept specifications expressed in graded-CTL. The test results we report are very comfortable in the sense that both the running time and the size of the BDDs produced are comparable to those obtained with specifications expressed in classical CTL.

* Work partially supported by M.I.U.R. grant ex-60%:"Metodi formali per la verifica automatica di sistemi" and by the National Research Project (PRIN'07) "Integrating automated reasoning in model checking: towards push-button formal verification of large-scale and infinite-state systems".

K. Breitman and A. Cavalcanti (Eds.): ICFEM 2009, LNCS 5885, pp. 306–325, 2009.

1 Introduction

Recently a new logic strictly more expressive than CTL has been introduced by the same authors in [FNP08, FNP10], called graded-CTL. It augments the existential and universal quantifiers with *graded* modalities that allow to reason about either *at least* or *all but* any number of futures. In literature, graded modalities have been intensively studied in various logic frameworks. In classical logics $\exists^{>k}$ and $\forall^{\leq k}$ are called *counting quantifiers*, see e.g. [GOR97, GMV99, PST00], in modal logics they are called *graded* modalities, see e.g. [Fin72, Tob01], and in description logics one speaks about *number restriction* of properties describing systems, see e.g. [HB91]. A different extension of CTL (RCTL) has been also defined in [EMSS92], where bounds are placed on the temporal modalities, instead of on the path quantifiers, bounding thus the maximum number of permitted transitions along a path.

Simple examples of graded-CTL are the formula $E^{>k}\mathcal{F}(critic1 \wedge critic2)$, which expresses that there exist more than k possibilities to violate the mutual exclusion property to enter the critical section of a system, and the formula $E^{>k}\mathcal{F}good$ which expresses the fact that the system has several ways to reach a good state. Formulas of these types cannot be expressed in CTL and not even in classical μ-calculus. Consider the two Kripke structures in the following figure, they cannot be distinguished by any CTL formula, while on the contrary, only the first is a model for the graded-CTL formula $E^{>1}\mathcal{X}p$, which says that there is more than one next state where p holds.

Another favorable point for studying this new logic is when we want to express that exactly one path satisfies a path formula, say for example $\mathcal{G}\varphi$, which means that φ holds forever in the states along a path: we can use the graded-CTL formula $E^{>0}\mathcal{G}\varphi \wedge \neg E^{>1}\mathcal{G}\varphi$. This latter example also shows that there is a great difference between a graded-CTL formula and the CTL formula obtained simply by ignoring the constant grading the path quantifiers: not only the models of the two formulas are different but even the satisfiability may change since, the deletion of the constants from $E^{>0}\mathcal{G}p \wedge \neg E^{>1}\mathcal{G}p$ produces a CTL formula which is not satisfiable.

Our contribution is on both a theoretical and an applicative side. For the former we give a simple algorithm to decide the satisfiability of a graded-CTL formula φ in time $2^{\mathcal{O}(|\varphi|^4)}$, when the grading constants occurring in the quantifiers of φ are expressed in unary. On the applicative side, we deal with the model-checking framework for the graded-CTL logic. In [FNP10] it has been shown that the graded-CTL model-checking problem can be solved in polynomial time and independently from the constant values grading the path quantifiers of the formula (and thus the representation of the constants does not affect the

running time of the model checking algorithm). Here, we propose symbolic algorithms for solving the model checking problem. As widely known, symbolic model checking [BCM+90] is a technique that allows, by representing and manipulating sets of vertices, to manage models with very high number of states. This have also been applied to the model checking of CTL [CE81] by revealing a very high efficiency in practice, especially in hardware verification [McM93].

We have implemented our algorithms with *Binary Decision Diagrams*, BDDs [Bry92], and have integrated them in the NuSMV model-checker [CCG+02] (and actually are collaborating with the development team for the integration of graded-CTL in the next official release of NuSMV). Besides being more expressive than CTL, a motivation for the use of graded-CTL in the model checking framework, is its close relation to the counterexamples generated by the model-checker tools. In fact these tools generate one counterexample for each run and they are used as a step in the *Check/Analyze/Fix* loop: *Check* the model against a specification, *Analyze* the counterexample generated by the tool and re-design the model after having *Fixed* the errors. The Check stage is often expensive, in terms of time resources, so it would be desirable to minimize the number of runs of the model-checkers. The complexity of the Analyze stage depends on the time the designer needs to interpret the counterexamples, and this task can be facilitated by providing more meaningful counterexamples. With respect to this, we think that graded-CTL can be much useful, in fact by using the graded modalities we can get more counterexamples with a unique run of the model checker and, hopefully, one does not have to undergo again through the time-consuming three stage cycle, c.f. [CG07, CIW+01, DRS03].

Clearly, it is possible in principle, to modify a tool checker to let it generate multiple counterexamples without changing the logic. Anyway, this is not likely to be done for essentially two reasons. First suppose that for example a system designer desires two evidences to the CTL formula $E\mathcal{F}E\mathcal{G}p$. He cannot choose the "type" of the evidences unless he uses a graded-CTL formula, either $E^{>0}\mathcal{F}E^{>1}\mathcal{G}p$ or $E^{>1}\mathcal{F}E^{>0}\mathcal{G}p$, which allow to get different evidences according to the needs (and not following a policy hard-coded once and for all into the tool). Second, it is not a trivial task to symbolically implement an algorithm that analyzes the model looking for *distinct* counterexamples (consider, for example, the inherent difficulty in the symbolic implementation of a DFS); our algorithms, instead, have been explicitly designed to make the computation of distinct counterexamples as quick and easy as possible.

We have implemented the symbolic algorithms into the well known NuSMV model checker, version 2.4.3, and have tested it on various examples. We report some results obtained from examples of the official NuSMV web site. Other tests and the package for graded-CTL can be found at `http://gradedctl.dia.unisa.it`. The experimental results indicate that there is no substantial overhead both in time and in size of BDDs needed to process graded-CTL formulas with respect to the classical CTL ones.

Related Works. In [KSV02], complexity issues related to the satisfiability problem for the μ-calculus when the universal and existential quantifiers are

augmented with graded modalities, have been investigated. They have shown that this problem is ExpTime-complete, retaining thus the same complexity as in the case of classical μ-calculus, though strictly extending it. There, the values of the constants grading the quantifiers are represented in binary.

In [BMM09] a logic with the same expressivity of our graded-CTL logic is considered, though their interpretation of the graded quantifiers is different and seems to be less natural than ours. Consider in fact the formula $E^{>k}\mathcal{G}\text{True}$: in our logic it extends the CTL formula $E\mathcal{G}\text{True}$ (having the intuitive meaning that *at least k different paths stem*); in their interpretation it is a contradiction, since it has no models. Also they solve the satisfiability problem, anyway the complexity of the translations between our and their logics prevents the two results to be derived from each other. The main differences between these two results is that theirs is in time $2^{\mathcal{O}(|\varphi|^5)}$.

In the last years, symbolic computations have also been applied to other kinds of problems. In [BGS06] the authors show a symbolic algorithm for the computation of the maximum flow in a 0-1 network, while in [GPP07] graph connectivity related problems are studied from a symbolic point of view. Recently symbolic techniques have also been applied to the satisfiability problem for the modal logic **K** [PSV05] and for CTL [Mar05].

The rest of the paper is organized as follows: in Section 2 we give the definitions of graded-CTL. In Section 3 we solve the satisfiability problem. In Section 4 we give the symbolic algorithms for the graded-CTL model checking problem. In Section 5 we describe the implementation of our algorithm into NuSMV and present the experimental results of the tests. In Section 6 we give our conclusions and outline some future research directions.

2 Graded-CTL Logic

In this section we recall the graded-CTL logic introduced in [FNP10]. The well-known temporal logic CTL [CE82] is a branching-time logic in which temporal operators express properties about a possible future and are preceded by a path quantifier. With this logic one can express properties that have to be true either *immediately after now* ($\mathcal{X}$), or *each time from now* ($\mathcal{G}$), or *from now until something happens* ($\mathcal{U}$), and it is possible to specify, through a path quantifier, that each property must hold either in *some possible futures* (E) or in *each possible future* (A). The graded-CTL logic extends CTL with graded quantifiers allowing to express also that a temporal property must hold either in *more than a given number* or in *all but a given number* of possible futures. The graded-CTL operators consist of the temporal operators $\mathcal{U}$ and $\mathcal{X}$, the boolean connectives $\wedge$ and $\neg$, and the graded path quantifier $E^{>k}$ (*for at least* $k+1$ *distinct futures*). Given a set of atomic propositions AP, the syntax of the graded-CTL formulas is:

$$\varphi := p \mid \neg\psi_1 \mid \psi_1 \wedge \psi_2 \mid E^{>k}\mathcal{X}\psi_1 \mid E^{>k}\mathcal{G}\psi_1 \mid E^{>k}\psi_1\mathcal{U}\psi_2$$

where $p \in AP$, ψ_1 and ψ_2 are graded-CTL formulas and k is a non-negative integer. The graded-CTL formulas, as in standard CTL, are also called *state-formulas* and $\mathcal{X}\psi_1$, $\mathcal{G}\psi_1$ and $\psi_1\mathcal{U}\psi_2$, are called, as usual, *path-formulas*. The

semantics of graded-CTL is defined with respect to a *Kripke Structure*, by means of a satisfiability relation $\models$. A Kripke structure over a set of atomic propositions AP is a tuple $\mathcal{K} = \langle S, s_{in}, R, L\rangle$, where S is a finite set of states, $s_{in} \in S$ is the initial state, $R \subseteq S \times S$ is a transition relation, with the property that for each state s there is a *successor* t such that $(s,t) \in R$, and $L : S \to 2^{AP}$ is a state labeling function. In the rest of the paper, with $\mathcal{K}$ we always denote the Kripke structure $\langle S, s_{in}, R, L\rangle$.

The relation $\models$ is defined as follows:

- $(\mathcal{K}, s) \models p$ iff $p \in L(s)$;
- $(\mathcal{K}, s) \models \psi_1 \wedge \psi_2$ iff $(\mathcal{K}, s) \models \psi_1$ and $(\mathcal{K}, s) \models \psi_2$;
- $(\mathcal{K}, s) \models \neg\psi_1$ iff $\neg((\mathcal{K}, s) \models \psi_1)$
- $(\mathcal{K}, s) \models E^{>k}\mathcal{X}\psi_1$ iff there exist $k+1$ different successors $s_0, \ldots, s_k$ of s such that $(\mathcal{K}, s_i) \models \psi_1$ for all $0 \leq i \leq k$;

To define the semantics for $\mathcal{G}$ and $\mathcal{U}$ operators, let us first introduce the notion of *distinct* paths which plays an important role. The length $|\pi|$ of a path π in $\mathcal{K}$ is the number of its states, and $\pi[i]$ denotes the i-th state in π, $0 \leq i < |\pi|$. Two paths π_1 and π_2 are *distinct* if there exists an index $0 \leq i < \min\{|\pi_1|, |\pi_2|\}$ such that $\pi_1[i] \neq \pi_2[i]$. Observe that from this definition if a path is the prefix of another path, then they are not distinct.

- $(\mathcal{K}, s) \models E^{>k}\mathcal{G}\psi_1$ iff there exist $k+1$ pairwise distinct infinite paths π_j, $0 \leq j \leq k$, starting from s and such that $(\mathcal{K}, \pi_j[h]) \models \psi_1$, for all $h \geq 0$. These paths π_j are said to satisfy the path-formula $\mathcal{G}\psi_1$.
- $(\mathcal{K}, s) \models E^{>k}\psi_1\mathcal{U}\psi_2$ iff there exist $k+1$ pairwise distinct finite paths π_j of length $i_j + 1$, for $0 \leq j \leq k$, and starting from s such that:

  1. $(\mathcal{K}, \pi_j[i_j]) \models \psi_2$, and
  2. for every $0 \leq h < i_j$, $(\mathcal{K}, \pi_j[h]) \models \psi_1$;

  These paths π_j are said to satisfy the path-formula $\psi_1\mathcal{U}\psi_2$.

We say that a state s in $\mathcal{K}$ satisfies a state-formula φ if $(\mathcal{K}, s) \models \varphi$ and $\mathcal{K}$ models (or also is a model of) φ, if $(\mathcal{K}, s_0) \models \varphi$

Observe that we have expressed the syntax of graded-CTL with one of the possible minimal sets of operators. Other temporal operators can be easily derived from those. For example, the temporal operator $\mathcal{F}$ (*eventually*) can be expressed by: $E^{>k}\mathcal{F}\psi_1 \Leftrightarrow E^{>k}\textsc{True}\mathcal{U}\psi_1$. Moreover, the path quantifier $E^{=k}$ can be expressed, as shown in the introduction, since $E^{=k}\psi$ is equivalent to $E^{>k-1}\psi \wedge \neg E^{>k}\psi$, and also the graded extension of the universal quantifier, $A^{\leq k}$, can be defined, with the meaning that *all the paths starting from a node s, but at most k pairwise distinct paths, satisfy a given path-formula*. The quantifier $A^{\leq k}$ is the dual operator of $E^{>k}$ and can obviously be re-written in terms of $\neg E^{>k}$. The formulas $A^{\leq k}\mathcal{X}\psi_1$ and $A^{\leq k}\mathcal{G}\psi_1$ are equivalent to respectively $\neg E^{>k}\mathcal{X}\neg\psi_1$ and $\neg E^{>k}\mathcal{F}\neg\psi_1$, while the formula $A^{\leq k}\psi_1\mathcal{U}\psi_2$ with $k > 0$ deserves more attention. In fact, we have that $A^{\leq k}\psi_1\mathcal{U}\psi_2$ is equivalent to $\neg E^{>k}\neg(\psi_1\mathcal{U}\psi_2)$, but this

formula is not a graded-CTL formula because of the occurrence of the innermost negation. This latter can be expressed in graded-CTL in the following way:

$$A^{\leq k}\psi_1\mathcal{U}\psi_2 \iff \neg E^{>k}\mathcal{G}(\psi_1 \wedge \neg\psi_2) \ \wedge \neg E^{>k}(\psi_1 \wedge \neg\psi_2)\mathcal{U}(\neg\psi_1 \wedge \neg\psi_2) \wedge \bigwedge_{i=0}^{k-1} (\neg E^{>k-1-i}\mathcal{G}(\psi_1 \wedge \neg\psi_2) \ \vee \ \neg E^{>i}(\psi_1 \wedge \neg\psi_2)\mathcal{U}(\neg\psi_1 \wedge \neg\psi_2)) \quad (1)$$

Equivalence (1) holds because if a path does not satisfy $\psi_1\mathcal{U}\psi_2$ then it satisfies either $\theta_1 = \mathcal{G}(\psi_1 \wedge \neg\psi_2)$ or $\theta_2 = (\psi_1 \wedge \neg\psi_2)\mathcal{U}(\neg\psi_1 \wedge \neg\psi_2)$ and, moreover, the paths satisfying θ_1 are all distinct from the paths satisfying θ_2.

Let us now recall the definitions of the graded-CTL satisfiability and model-checking problems. The **graded-CTL SAT** is the problem of verifying whether a Kripke structure exists which models a given graded-CTL formula. The **graded-CTL model-checking**, given a Kripke structure $\mathcal{K}$ and a graded-CTL formula φ, is the problem of verifying whether $\mathcal{K}$ models φ.

In spite of the augmented expressiveness, the complexity of the graded-CTL model-checking problem remains the same as that of CTL, since this problem is solved in polynomial time and independently from the constant values grading the path quantifiers of the formula.

Let $|\varphi|$ be the number of the temporal and the boolean operators occurring in a graded-CTL formula φ.

Theorem 1. *[FNP10] The graded-CTL model-checking problem for a Kripke structure $\mathcal{K}$ and a graded-CTL formula φ can be solved in time $\mathcal{O}(|R| \cdot |\varphi|)$.*

Distinct paths. Since graded-CTL requires to count the paths satisfying a formula, we have introduced the notion of *distinct* paths. Now we briefly discuss this definition. For the globally operator we had no choice since we have to distinguish infinite paths. On the contrary the other temporal operators require a deeper reasoning. The most reasonable choice in this case is to count distinct finite evidences of a formula. In fact, the different choice to count infinite distinct paths (as done for the globally operator) may cause loss of information, as illustrated by the fact that the validity of the formula $E^{>k}\mathcal{F}$ *safe* no longer ensures that a system has more ways to reach safe states. In fact also paths that diverge after the last safe state would be counted as distinct.

Another possible choice is to consider as distinct two evidences of $\psi_1\mathcal{U}\psi_2$ also in the case that one is the prefix of the other (this logic, in a certain sense, allows to count the number of states satisfying a formula in a Kripke structure). However, it can be proved quite immediately, that this choice leads to a logic that is no more expressive than ours.

3 The SAT Problem

In this section we show that SAT problem for graded-CTL is ExpTime-complete when the grading constants in the path quantifiers are expressed in unary. The membership proof is based on the reduction of the SAT problem to the emptiness problem for Büchi Automata on Infinite Trees. Since an infinite tree can be seen

as a special Kripke structure with an infinite set of states, we can easily extend the semantics of the graded-CTL logic to infinite trees and we say that an infinite tree T is a model (i.e. satisfies) a graded-CTL formula φ iff $(T, root(T)) \models \varphi$.

Given a graded-CTL formula φ, we consider the set E_φ of the subformulas $E^{>k}\theta$ of φ occurring in positive form, that is we do not include in E_φ the subformulas $\neg E^{>k}\theta$. Our algorithm to solve the graded-CTL SAT problem is based on the fact that graded-CTL, as stated in the following lemma, obeys to a *Tree Model Property*.

First, we define for a graded-CTL formula φ the set $ecl(\varphi)$ (that we call *extended closure* of φ) as the minimal set of graded-CTL formulas such that

- TRUE is in $ecl(\varphi)$ and φ is in $ecl(\varphi)$;
- if $\psi_1 \wedge \psi_2$ is in $ecl(\varphi)$, then both ψ_1 and ψ_2 are in $ecl(\varphi)$
- if $E^{>k}\mathcal{X}\psi_1$ $(k \geq 0)$ is in $ecl(\varphi)$, then ψ_1 is in $ecl(\varphi)$;
- if $E^{>0}\mathcal{G}\psi_1$ is in $ecl(\varphi)$, then ψ_1 is in $ecl(\varphi)$;
- if $E^{>0}\psi_1\mathcal{U}\psi_2$ is in $ecl(\varphi)$, then both ψ_1 and ψ_2 are in $ecl(\varphi)$;
- if $E^{>k}\theta$ is in $ecl(\varphi)$ with $k > 0$ and either $\theta = \mathcal{G}\psi_1$ or $\theta = \psi_1\mathcal{U}\psi_2$, then $E^{>i}\theta$ is in $ecl(\varphi)$ for all $0 \leq i \leq k-1$;
- if ψ is in $ecl(\varphi)$ then $\neg\psi$ is in $ecl(\varphi)$;

To get *ecl* finite, we assume that $\neg\neg\varphi$ is replaced by φ.

Lemma 1. *If a graded-CTL formula φ is satisfiable, then it is satisfiable on a 2^{AP}-labeled infinite tree with branching degree bounded by $b = \hat{k} + l + 1$, where $\hat{k}$ is the sum of the grading constants occurring in the subformulas in E_φ and l is the number of these subformulas.*

Proof. Let $\mathcal{K}$ be a model of φ and let us consider its unwinding T; obviously T satisfies φ. Suppose that the branching degree of T is greater than b; we will show how to modify T to obtain a tree with branching degree at most b and that still satisfies φ. Let $x \in T$ be a node having $n > b$ children. Let us denote by $F(x)$ a minimal subset $ecl(\varphi)$ containing the graded-CTL formulas that have to be satisfied in x in order to have that T is a model of φ. If $F(x)$ contains only boolean combinations of atomic proposition or formulas of the type $\neg E^{>k}\theta$, we choose one child of x and prune all subtrees rooted in the remaining children of x. Each other formula in $F(x)$ not containing path quantifiers still holds true, because it only depends on the labeling of the node x, and each other formula in $F(x)$ of the kind $\neg E^{>k}\theta$ is still satisfied in x because we have only deleted paths starting from x.

Suppose now that $F(x)$ contains also the formulas $E^{>k_1}\theta_1, \ldots, E^{>k_t}\theta_t$. From the minimality of $F(x)$, it follows that $\theta_i \neq \theta_j$, for $i \neq j$, and thus $k_1 + \ldots + k_t \leq \hat{k}$ and $t \leq l$. Consider, for each $1 \leq i \leq t$, a minimal set C_i of children of x that, all together, allow x to satisfy $E^{>k_i}\theta_i$. More precisely, called $m_y = max\{h | (K, y) \models \exists^{>h-1}\theta_i\}$, for a child y of x, $k_i < \sum_{y \in C_i} m_y$, and, for any proper subset C_i' of C_i, $k_i \geq \sum_{y \in C_i'} m_y$. From this definition it follows that $m_y \neq 0$ for every $y \in C_i$ and thus $|C_i| \leq k_i + 1$ for all $1 \leq i \leq t$. Now we prune from T all the subtrees rooted in children of x that are not in $C_1 \cup \ldots \cup C_t$. Since $|C_1 \cup \ldots \cup C_t| \leq k_1 + \ldots + k_t + t \leq b$,

now the node x has at most b children. Reasoning as above, formulas in $F(x)$ that either do not contain path quantifiers or are of kind $\neg E^{>k}\theta$ are still satisfied in x. Moreover, the formula $E^{>k_i}\theta_i$ is still satisfied in x because of the x's children that are in C_i.

By iterating this procedure on each node with degree greater than b, we obtain that there exists an infinite tree with branching degree at most b whose root satisfies φ, and this completes the proof. □

Now we show how to solve the SAT problem for a graded-CTL formula in time exponential in the size of the formula. Since $|\varphi|$ is the number of the temporal and the boolean operators occurring in a graded-CTL formula φ, we analyze the complexity of the problem with respect to $|\varphi|_u = |\varphi| + \hat{k}$

Theorem 2. *The satisfiability of a graded-CTL formula φ can be decided in time $2^{\mathcal{O}(|\varphi|_u^4)}$ if the constants appearing in the graded operators of φ are expressed in unary.*

Proof. To prove the theorem, we reduce the satisfiability problem for graded-CTL to the nonemptiness problem of *Nondeterministic Büchi Tree Automata* (NBTA). In particular, we show that for each graded-CTL formula there is an NBTA $\mathcal{A}_\varphi = \langle 2^{AP}, Q, Q_0, \delta, \mathcal{F}\rangle$ that accepts all and only the 2^{AP}-labeled infinite trees satisfying φ, whose branching degree is bounded by b. It is easy to show that $\mathcal{L}(\mathcal{A}_\varphi) \neq \emptyset$ iff φ is satisfiable. From Lemma 1 we have that if $\mathcal{L}(\mathcal{A}_\varphi) = \emptyset$ then φ is not satisfiable. On the other side, if $\mathcal{L}(\mathcal{A}_\varphi) \neq \emptyset$, since an NBTA accepts only regular trees, an infinite regular tree exists satisfying φ; and thus a model for φ exists, since, as it is well known, any regular infinite tree is the unwinding of a Kripke structure.

Let us now describe the automaton $\mathcal{A}_\varphi$. The idea is that each state of the automaton is a set of graded-CTL formulas that have to be satisfied in a node x and the automaton decides, based on the current state and on the label of x, the formulas that have to be satisfied in each child of x. More precisely, the set of the states of the automaton $\mathcal{A}_\varphi$ is the subset $Q \subseteq 2^{ecl(\varphi)}$ such that for all $q \in Q$ the following consistency rules hold:

- if $\psi_1 \wedge \psi_2 \in q$ then $\psi_1 \in q$ and $\psi_2 \in q$,
- if $\neg(\psi_1 \wedge \psi_2) \in q$ then either $\neg\psi_1 \in q$ or $\neg\psi_2 \in q$,
- if $E^{>k}\mathcal{G}\psi_1 \in q$ $(k \geq 0)$ then $\psi_1 \in q$,
- if $E^{>0}\psi_1\mathcal{U}\psi_2 \in q$ then either $\psi_1 \in q$ or $\psi_2 \in q$,
- if $E^{>k}\psi_1\mathcal{U}\psi_2 \in q$ $(k > 0)$ then $\psi_1 \in q$,
- if $\neg E^{>0}\psi_1\mathcal{U}\psi_2 \in q$ then $\neg\psi_2 \in q$,
- for all $\psi \in ecl(\varphi)$, $\psi \in q$ iff $\neg\psi \notin q$.

The set of initial states is the subset $Q_0 \subseteq Q$ containing all the states q such that $\varphi \in q$. A state $q \in Q$ is final iff it satisfies the following properties: *(i)* FALSE $\notin q$, *(ii)* q doesn't contain any formula of kind $E^{>k}\theta$ with $k > 0$ and either $\theta = \mathcal{G}\psi_1$ or $\theta = \psi_1\mathcal{U}\psi_2$, *(iii)* if q contains a formula of kind $E^{>0}\psi_1\mathcal{U}\psi_2$ then it also contains ψ_2 and *(iv)* if q contains a formula of kind $\neg E^{>0}\mathcal{G}\psi_1$ then it also contains $\neg\psi_1$.

Let us now describe the transition function of $\mathcal{A}_\varphi$. Let us suppose that the automaton is in a state $q \in Q$ and is reading the label σ of a node x with $deg(x)$ children. With its transition function, the automaton assigns to each child x a state including a set of formulas chosen as follows, for each $\psi \in q$.

- if $\psi = \text{TRUE}$, then TRUE is added to the set of formulas of each child; analogously for $\psi = \text{FALSE}$;
- if $\psi = p$ ($p \in AP$) and $p \in \sigma$ (resp. $p \notin \sigma$), then TRUE (resp. FALSE)is added to the set of formulas of each child; analogously for $\psi = \neg p$;
- if $\psi = E^{>k}\mathcal{X}\psi_1$ ($k \geq 0$), then $k+1$ children are chosen and ψ_1 is added to the sets of formulas of these children;
- if $\psi = \neg E^{>k}\mathcal{X}\psi_1$ ($k \geq 0$), then at most k children are chosen and ψ_1 is added to the sets of formulas of these children and $\neg\psi_1$ is added to the sets of formulas of the remaining children;
- if $\psi = E^{>k}\mathcal{G}\psi_1$ ($k \geq 0$), then t children $x_1, \ldots, x_t$ and t positive integers $k_1 \ldots, k_t$ are chosen, such that $k_1 + \ldots + k_t + t = k + 1$, and $E^{>k_j}\mathcal{G}\psi_1$is added to the set of formulas of x_j, for all $1 \leq j \leq t$;
- if $\psi = E^{>0}\psi_1\mathcal{U}\psi_2$ and $\neg\psi_2 \in q$, then a child is chosen and $E^{>0}\psi_1\mathcal{U}\psi_2$ is added to the set of formulas of that child;
- if $\psi = E^{>k}\psi_1\mathcal{U}\psi_2$ ($k > 0$), then t children $x_1, \ldots, x_t$ and t positive integers $k_1, \ldots, k_t$ are chosen, such that $k_1 + \ldots + k_t + t = k + 1$, and $E^{>k_j}\psi_1\mathcal{U}\psi_2$ is added to the set of formulas x_j, for all $1 \leq j \leq t$;
- if $\psi = \neg E^{>k}\theta$ (with either $\theta = \mathcal{G}\psi_1$ or $\theta = \psi_1\mathcal{U}\psi_2$ and $k \geq 0$) and $\psi_1 \in q$, then $deg(x)$ non negative integers $k_1, \ldots, k_{deg(x)}$ are chosen, such that $k_1 + \ldots + k_{deg(x)} \leq k$, and $\neg E^{>k_j}\theta$ is added to the set of formulas of the j-th child, for all $1 \leq j \leq deg(x)$;
- in the remaining cases, TRUE is added to the set of formulas of each child.

Let us evaluate the size of the automaton and the running time of the algorithm. It is easy to see that $|ecl(\varphi)| = \mathcal{O}(|\varphi|_u)$, therefore the automaton has $2^{\mathcal{O}(|\varphi|_u)}$ states. In the worst case, the function δ contains all the tuples of states with length $b = \mathcal{O}(|\varphi|_u)$, therefore the transition function has total size $|\delta| \leq |Q|^{b+1} = 2^{\mathcal{O}(|\varphi|_u^2)}$ and the size of the automaton is $|\mathcal{A}_\varphi| = 2^{\mathcal{O}(|\varphi|_u^2)}$. Since the nonemptiness problem for an NBTA can be solved in time quadratic in the length of the string representing the automaton [VW86], we obtain that our algorithm works in time $\mathcal{O}(|\mathcal{A}_\varphi|^2) = 2^{\mathcal{O}(|\varphi|_u^4)}$. □

From the previous theorem and the EXPTIME-completeness of the SAT problem for CTL, the following corollary holds.

Corollary 1. *The SAT problem for graded-CTL is* EXPTIME*-complete.*

4 Symbolic Model Checking Algorithms

In this section we give symbolic algorithms to solve the graded-CTL model checking problem. Let us recall that a *symbolic algorithm* manipulates sets and uses

basic set operations, such as union, intersection, and complementation. In symbolic model-checking, states and transitions are represented as boolean functions on the set of the atomic propositions, that in turn, can be represented as the set of variable assignments. In this framework, the fundamental symbolic operation is the computation of the *pre-image* of a set of destination states, (i.e., the states having a successor in the given set). This is performed by using the classical *existential quantification* operation on boolean functions (corresponding to a *projection* on sets). In graded-CTL model-checking, the counterpart of the pre-image is the computation of the *number of successors* that a state has in the destination set, called *image-size*. The image-size function is computed with the existential quantification on multisets (corresponding to a projection on multiset). A Multiset is used to distinguish multiple occurrences of elements and is represented by a pair (M, m) where M is a set of elements and m is a multiplicity function that returns the number of occurrences of the element in input. The projection on multiset is performed with the $\uplus$ operator which sums the multiplicity functions of two multisets. Moreover, our algorithms use also the function $multisetToSet((M, m), i)$ that returns the set of the elements of M having multiplicity greater than i, that is $multisetToSet((M, m), i) = \{s \in M \mid m(s) > i\}$. Some details on the implementation of the above functions can be found in the next section.

It is known that symbolic model-checking algorithms are in the practical cases very efficient, and this depends on the practical efficiency of the data structures used to represent and manipulate sets and multisets. Therefore, as also suggested in [BGS06], we will measure the asymptotic complexity of our algorithms in terms of the *number of pre-image and image-size computations* (we will call *pre* and $imgSize$ the functions that compute respectively the pre-image and the image-size).

We denote, for a graded-CTL formula φ, with $[\varphi]$ the set of states of the Kripke structure where φ holds.

Let us now show how to model check formulas of kind $E^{>k}\theta$ $(k \geq 0)$. If $\theta = \mathcal{X}\psi_1$ and $[\psi_1]$ has already been computed, then φ can be easily checked by a function $existNext(\mathcal{K}, k, [\psi_1])$ that first computes the image-size (S, m) of $[\psi_1]$ and then returns the set of states $s \in S$ such that $m(s) > k$ (i.e., $multisetToSet((S, m), k)$).

Therefore we have the following lemma.

Lemma 2. *Given a formula $\varphi = E^{>k}\mathcal{X}\psi_1$ $(k \geq 0)$, there is a symbolic algorithm that takes as input $[\psi_1]$ and solves the model checking problem for φ by using $\mathcal{O}(1)$ $imgSize$ computations.*

Now let us show how to solve the model checking problem for a formula $\varphi = E^{>k}\theta$ with either $\theta = \psi_1 \mathcal{U} \psi_2$ or $\theta = \mathcal{G}\psi_1$ $(k \geq 0)$.

Lemma 3. *Given a formula $\varphi = E^{>k}\psi_1 \mathcal{U} \psi_2$ $(k \geq 0)$, there is a symbolic algorithm that takes in input $[\psi_1]$ and $[\psi_2]$ and solves the model checking problem for φ with $\mathcal{O}(k \cdot |S|)$ pre and $imgSize$ computations.*

Proof. We can solve the model checking problem for φ by using the following function *existUntil*.

Function $existUntil(\mathcal{K}, k, [\psi_1], [\psi_2])$

1. $S^0 \leftarrow [\psi_2]$; $PRED \leftarrow pre(\mathcal{K}, S^0) \cap [\psi_1]$;
2. **while** $PRED \not\subseteq S^0$ **do** $S^0 \leftarrow S^0 \cup PRED$; $PRED \leftarrow pre(\mathcal{K}, S^0) \cap [\psi_1]$;
3. $[E^{>0}\psi_1 \mathcal{U} \psi_2] \leftarrow S^0$; Let $(S, sumSucc)$ be such that $sumSucc(s) = 0$ for all $s \in S$;
4. **for** $i \leftarrow 1$ *to* k **do**
5. $\quad (S, succ) \leftarrow imgSize([E^{>i-1}\psi_1 \mathcal{U} \psi_2])$;
6. $\quad (S, sumSucc) \leftarrow (S, sumSucc) \uplus (S, succ)$;
7. $\quad S^i \leftarrow multisetToSet((S, sumSucc), i) \cap [\psi_1]$;
8. $\quad PRED \leftarrow pre(\mathcal{K}, S^i) \cap [\psi_1]$;
9. $\quad$ **while** $PRED \not\subseteq S^i$ **do** $S^i \leftarrow S^i \cup PRED$; $PRED \leftarrow pre(\mathcal{K}, S^i) \cap [\psi_1]$;
10. $\quad [E^{>i}\psi_1 \mathcal{U} \psi_2] \leftarrow S^i$;
11. **end**
12. **return** $[E^{>k}\psi_1 \mathcal{U} \psi_2]$;

For $k = 0$, this function essentially resembles the classical CTL symbolic model checking algorithm [BCM+90]. For $k > 0$ we use, for a state $s \in S$ and $1 \leq i \leq k$, the functions *succ* and *sumSucc*, defined as follows:

$$succ_s^{i-1} = |\{s' \in [E^{>i-1}\psi_1 \mathcal{U} \psi_2] \text{ s.t. } (s, s') \in R\}| \text{ and}$$
$$sumSucc_s^i = \sum_{j=0}^{i-1} succ_s^j.$$

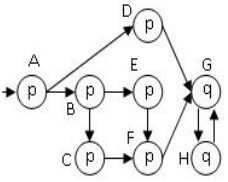
The function $succ_s^{i-1}$ is the number of successors of s satisfying $E^{>i-1}\psi_1 \mathcal{U} \psi_2$. Let us observe that if t is a successor of s from which i paths start, each satisfying $\psi_1 \mathcal{U} \psi_2$, then t satisfies $E^{>j}\psi_1 \mathcal{U} \psi_2$, for $0 \leq j < i$, and thus t contributes for i times in the computation of $sumSucc_s^k$. Then $(\mathcal{K}, s) \models \psi$ iff $s \in [\psi_1]$ and one of these two conditions holds:

1. $sumSucc_s^k > k$, that is from the successors of s, $k + 1$ paths stem, each satisfying $\psi_1 \mathcal{U} \psi_2$;
2. there is one successor of s satisfying $E^{>k}\psi_1 \mathcal{U} \psi_2$.

Based on the above observations, the function *existUntil* satisfies the following invariants, at the end of the i-th iteration:

- the multiset $(S, succ)$ contains the values $succ_s^{i-1}$ for all $s \in S$,
- the multiset $(S, sumSucc)$ contains the values $sumSucc_s^i$ for all $s \in S$ and
- $S^i = [E^{>i}\psi_1 \mathcal{U} \psi_2]$.

The proof can be easily obtained by induction on i (it is useful to recall that, given (M, m_1) and (M, m_2), $(M, m_1) \uplus (M, m_2) = (M, m)$ with $m(s) = m_1(s) + m_2(s)$ for all $s \in M$). In particular, to compute S^i, the function computes first the set $\{s \in S \text{ s.t. } sumSucc_s^i > i\}$ (line 6), that is the set of the states satisfying the condition 1 above, and then it applies a least fixpoint algorithm starting from this set (lines 7-8) to compute the set of the states having a successor satisfying $E^{>i}\psi_1 \mathcal{U} \psi_2$, according to the condition 2. □



(a)

$E^{>3}p\mathcal{U}q$			
i	1	2	3
S^{i-1}	A, B, C, D, E, F, G, H	A, B, C, E, F	A, B
$(S, succ)$	$(A,1),(B,3),(C,1),(D,1),(E,1),(F,2),(G,1),(H,1)$	$(A,1),(B,2),(C,1),(D,0),(E,1),(F,0),(G,0),(H,0)$	$(A,1),(B,0),(C,0),(D,0),(E,0),(F,0),(G,0),(H,0)$
$(S, sumSucc)$	$(A,1),(B,3),(C,1),(D,1),(E,1),(F,2),(G,1),(H,1)$	$(A,2),(B,5),(C,2),(D,1),(E,2),(F,2),(G,1),(H,1)$	$(A,3),(B,5),(C,2),(D,1),(E,2),(F,2),(G,1),(H,1)$
S^i (line 6)	B, F	B	B
S^i (line 9)	A, B, C, E, F	A, B	A, B

(b)

Fig. 1. An execution of the function *existUntil*

In Figure 1(b) an execution of the function *existUntil* on the Kripke structure of figure 1(a) and the formula $E^{>3}p\mathcal{U}q$ is reported.

Lemma 4. *Given a formula $\varphi = E^{>k}\mathcal{G}\psi_1$ ($k \geq 0$), there is a symbolic algorithm that takes in input $[\psi_1]$ and solves the model checking problem for φ with $\mathcal{O}(k \cdot |S|)$ pre and imgSize computations.*

Proof. By using a similar reasoning as done in Lemma 3, we can solve the model checking for φ by using the following function *existGlobally*, that is quite similar to the function *existUntil*.

Function $existGlobally(\mathcal{K}, k, [\psi_1])$

1. $S^0 \leftarrow [\psi_1]$; $PRED \leftarrow pre(\mathcal{K}, S^0) \cap [\psi_1]$;
2. **while** $PRED \neq S^0$ **do** $S^0 \leftarrow PRED$; $PRED \leftarrow pre(\mathcal{K}, S^0) \cap [\psi_1]$;
3. $[E^{>0}\mathcal{G}\psi_1] \leftarrow S^0$; Let $(S, sumSucc)$ be such that $sumSucc(s) = 0$ for all $s \in S$;
4. **for** $i \leftarrow 1$ *to* k **do**
5. $\quad (S, succ) \leftarrow imgSize([E^{>i-1}\mathcal{G}\psi_1])$;
5. $\quad (S, sumSucc) \leftarrow (S, sumSucc) \uplus (S, succ)$;
6. $\quad S^i \leftarrow multisetToSet((S, sumSucc), i) \cap [\psi_1]$;
7. $\quad PRED \leftarrow pre(\mathcal{K}, S^i) \cap [\psi_1]$;
8. $\quad$ **while** $PRED \not\subseteq S^i$ **do** $S^i \leftarrow S^i \cup PRED$; $PRED \leftarrow pre(\mathcal{K}, S^i) \cap [\psi_1]$;
9. $\quad [E^{>i}\mathcal{G}\psi_1] \leftarrow S^i$;
10. **end**
11. **return** $[E^{>k}\mathcal{G}\psi_1]$;

They essentially differ in the calculus of the base $[E^{>0}\theta]$, for which they resemble the classical CTL symbolic model checking algorithm [BCM+90]. □

Now we are ready to show our symbolic algorithm to solve the graded-CTL model checking problem.

Theorem 3. *The graded-CTL model checking problem can be solved with a symbolic algorithm that performs $\mathcal{O}(\tilde{k} \cdot |S| \cdot |\varphi|)$ calls to the functions pre and imgSize, where $\tilde{k}$ is the maximum grading constant appearing in φ.*

Proof. Algorithm 1 solves the graded-CTL model-checking. It uses the functions *existNext*, *existUntil* and *existGlobally* of Lemmas 2, 3 and 4.

Algorithm 1. $gradedCTL(\mathcal{K}, \varphi)$

1. **Input:** A Kripke structure and a graded-CTL formula φ.
2. **Output:** The set of states where φ holds.
3. **If** $\varphi = p$ $(p \in AP)$ **then return** $\{s \in S \text{ s.t. } p \in L(s)\}$;
4. **If** $\varphi = \neg\psi_1$ **then return** $S \setminus GradedCTL(\mathcal{K}, \psi_1)$;
5. **If** $\varphi = \psi_1 \wedge \psi_2$ **then return** $GradedCTL(\mathcal{K}, \psi_1) \cap GradedCTL(\mathcal{K}, \psi_2)$;
6. **If** $\varphi = E^{>k}\mathcal{X}\psi_1$ $(k \geq 0)$ **then return** $existNext(\mathcal{K}, k, GradedCTL(\mathcal{K}, \psi_1))$;
7. **If** $\varphi = E^{>k}\psi_1\mathcal{U}\psi_2$ $(k \geq 0)$ **then return** $existUntil(\mathcal{K}, k, GradedCTL(\mathcal{K}, \psi_1), GradedCTL(\mathcal{K}, \psi_2))$;
8. **If** $\varphi = E^{>k}\mathcal{G}\psi_1$ $(k \geq 0)$ **then return** $existGlobally(\mathcal{K}, k, GradedCTL(\mathcal{K}, \psi_1))$;

From previous Lemmas, the number of calls to the functions pre and $imgSize$ is

$$T(\mathcal{K}, \varphi) = \begin{cases} \mathcal{O}(1) & \text{if } \varphi = p \\ \mathcal{O}(1) + T(\mathcal{K}, \psi_1) & \text{if } \varphi = \neg\psi_1 \\ \mathcal{O}(1) + T(\mathcal{K}, \psi_1) + T(\mathcal{K}, \psi_2) & \text{if } \varphi = \psi_1 \wedge \psi_2 \\ \mathcal{O}(1) + T(\mathcal{K}, \psi_1) & \text{if } \varphi = E^{>k}\mathcal{X}\psi_1 \text{ with } k \geq 0 \\ \mathcal{O}(k \cdot |S|) + T(\mathcal{K}, \psi_1) + T(\mathcal{K}, \psi_2) & \text{if } \varphi = E^{>k}\psi_1\mathcal{U}\psi_2 \text{ with } k \geq 0 \\ \mathcal{O}(k \cdot |S|) + T(\mathcal{K}, \psi_1) & \text{if } \varphi = E^{>k}\mathcal{G}\psi_1 \text{ with } k \geq 0 \end{cases}$$

from which we have that $T(\mathcal{K}, \varphi) = \mathcal{O}(\tilde{k} \cdot |S| \cdot |\varphi|)$. □

Let us remark that in the syntax of graded-CTL logic we have not included the operator $A^{\leq k}$. In fact, as stated in section 2, this operator can be expressed in terms of $\neg E^{>k}$. Anyway, doing so there is an efficiency problem for the $\mathcal{U}$ operator that causes an efficiency loss for the model checking algorithm. In fact, from equivalence (1) one should evaluate $k+1$ formulas of kind $E^{>k}\mathcal{G}\theta_1$ and $k+1$ formulas of kind $E^{>k}\theta_1\mathcal{U}\theta_2$. Anyway, we show here that it is possible to avoid these extra evaluations by using a smarter algorithm.

From the equivalence (1), indeed, it is easy to see that, given a state $s \in S$, if $max_1(s)$ and $max_2(s)$ denote the maximum number of distinct paths, starting from s, satisfying $\mathcal{G}(\psi_1 \wedge \neg\psi_2)$ and $(\psi_1 \wedge \neg\psi_2)\mathcal{U}(\neg\psi_1 \wedge \neg\psi_2)$ respectively, then $(\mathcal{K}, s) \models E^{>k}\neg(\psi_1\mathcal{U}\psi_2)$ iff $max_1(s) + max_2(s) > k$. In fact, there do not exist paths satisfying both the two path-formulas, thus the two sets are disjoint and $max_1(s) + max_2(s)$ is the maximum number of distinct paths violating $\psi_1\mathcal{U}\psi_2$.

Function $forallUntil(\mathcal{K}, k, [\psi_1], [\psi_2])$

1. $[\psi_1 \wedge \neg\psi_2] \leftarrow [\psi_1] \cap (S \setminus [\psi_2])$; $[\neg\psi_1 \wedge \neg\psi_2] \leftarrow (S \setminus [\psi_1]) \cap (S \setminus [\psi_2])$;
2. $(S, max_1) \leftarrow maxPathsGlobally(\mathcal{K}, [\psi_1 \wedge \neg\psi_2], k+1)$;
3. $(S, max_2) \leftarrow maxPathsUntil(\mathcal{K}, [\psi_1 \wedge \neg\psi_2], [\neg\psi_1 \wedge \neg\psi_2], k+1)$;
4. **return** $multisetToSet((S, max_1) \uplus (S, max_2), k)$;

The function $forallUntil$ uses the functions $maxPathsGlobally(\mathcal{K}, [\theta_1], i)$ and $maxPathsUntil(\mathcal{K}, [\theta_1], [\theta_2], i)$ to compute $max_1(s)$ and $max_2(s)$, respectively, for all $s \in S$. These two functions returns, for each state $s \in S$, the maximum number (bounded by i) of distinct paths starting from s and satisfying $\mathcal{G}\theta_1$ and $\theta_1\mathcal{U}\theta_2$, respectively. The function $maxPathUntil$ can be implemented with a

simple modification of the function *existUntil*: execute the *for* loop until $k+1$, instead of k, and return the multiset $(S, sumSucc)$. Analogously, the function $maxPathGlobally$ can be implemented as a simple modification of the function $existGlobally$. In this way, if $[\psi_1]$ and $[\psi_2]$ are known, also the model checking of a formula $A^{\leq k}\psi_1 \mathcal{U} \psi_2$ requires time $\mathcal{O}(k \cdot |S|)$ and we have the following theorem.

Theorem 4. *The model checking problem for a graded-CTL formula φ possibly including* forall *subformulas can be solved with a symbolic algorithm that performs $\mathcal{O}(\tilde{k} \cdot |S| \cdot |\varphi|)$ calls to the functions pre and imgSize, where $\tilde{k}$ is the maximum grading constant appearing in φ.*

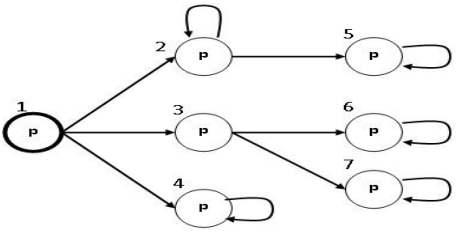
5 Implementation and Experimental Results

In this section we show the experimental results obtained by implementing our symbolic algorithms that we have integrated into the model checker tool NuSMV [CCG+02]. NuSMV is based on the CUDD library [Som05], for the treatment of BDDs, that allows also the use of *Algebraic Decision Diagrams*, ADDs in short [BFG+97]. These are a generalization of BDDs in which the leaves can be integers and are used to represent multisets. By using ADDs to manipulate multisets one gets several advantages, like natural transformations between sets and multisets (that are frequently used in our algorithms) and very efficient performances in practice. As said in the previous section, the basic operations we use are $\uplus$, $imgSize$ and $mutlisetToSet$. Their implementation quite naturally come from simple applications on ADDs of known BDD operations. More precisely, the $\uplus$ operator is obtained by applying to ADDs the classical *apply*: when the computation reaches two leaves x and y of an ADD, the result is a leaf with the sum of x and y. In a similar way, the implementation of the $imgSize$ function is an application of the classical *pre* function, with the $\cup$ operator substituted by the $\uplus$ operator. Finally, the $multisetToSet$ function is implemented by moving into a 1-node each leaf with value greater than k and into a 0-node the remaining leaves and then by compacting the resulting BDD. The enrichment of NuSMV with the graded-CTL model-checking capability, has implied the modification of the internal parser to allow the use of the syntax for graded-CTL specifications. Let us recall that NuSMV implements direct CTL model-checking procedures only for the operators $E\mathcal{X}$, $E\mathcal{G}$ and $E\mathcal{U}$ and derives from these the procedures for all the other operators (since all the transformations are linear in the size of the formula). In our setting instead, since the transformation of formulas $A^{\leq k}\mathcal{U}$ in terms of $E^{>k}\mathcal{G}$ and $E^{>k}\mathcal{U}$ is inefficient (the size of the resulting formula is $2(k+1)$ times the size of the original formula), we directly implement all the procedures given in section 4.

We have also implemented the *generation* of the counterexamples, that differently from what happens in CTL, returns trees (constituted by evidences of the negation of the formula) instead of paths. As said into section 1, the implementation of an algorithm for the generation of $k+1$ distinct evidences of a non trivial path-formula is not easy. More precisely, given a state s, while it

is easy to implement an algorithm that returns $k+1$ distinct evidences of the path-formula $\mathcal{X}\psi_1$ (simply compute the forward-image of s, intersect it with $[\psi_1]$ and pick $k+1$ states from the resulting set), for $\mathcal{G}\psi_1$ and $\psi_1\mathcal{U}\psi_2$ we have to implement a more complex algorithm. We explain here how to use some partial results of our symbolic algorithms (namely, the sets $[E^{>i}\theta]$, $0 \leq i \leq k$, with $\theta = \mathcal{G}\psi_1$ or $\theta = \psi_1\mathcal{U}\psi_2$) to quickly compute a tree of evidences.

When one has to compute a tree of evidences, it is necessary to decide whether it is better to compute a "wide" tree or a "tall" tree, in the sense that one should decide whether, for the case under examination, it is more significative to "distinguish" the evidences as soon as possible or as late as possible. Consider for example the following model and let us look for a counterexample of the formula $\neg E^{>3}\mathcal{G}p$.



In this case a tall evidence tree will be constituted by all paths which differ only for the number of times that they traverse the self-loop on the state 2, while a large tree includes other more significative evidences.

The technique we have implemented in our tool can be used to find both kinds of tree. The default is to return wide trees (which from our tests seems to be better). Once we have computed the sets $[E^{>i}\theta]$, $0 \leq i \leq k$, by using the functions *existUntil* and *existGlobally*, it is possible to compute a wide evidence tree by executing the following procedure with $s = s_{in}$.

Procedure *evidences*$(\mathcal{K}, k, s, \theta)$

1. **if** $k = 0$ **then**
2. | compute an evidence for θ from s as done in NuSMV;
3. **else**;
4. | $count \leftarrow 0$; $i \leftarrow 0$;
5. | **while** $count <= k$ **do**
6. | | **if** $i < k$ **then**
7. | | | $SUCC(s,i) \leftarrow forwardImg(s) \cap ([E^{>i}\theta] \setminus [E^{>i+1}\theta])$; $k' \leftarrow i$
8. | | **else**
9. | | | $SUCC(s,i) \leftarrow forwardImg(s) \cap [E^{>i}\theta]$; $k' \leftarrow \left\lceil \frac{(k-count)}{|SUCC(s,i)|} \right\rceil$
10. | | **end**
11. | | **forall** $t \in SUCC(s,i)$ **do**
12. | | | add t to as child of s; *evidences*$(\mathcal{K}, k', t, \theta)$; $count \leftarrow count + k'$
13. | | **end**
14. | | $i \leftarrow i + 1$;
15. | **end**
16. **end**

The procedure *evidences* determines $k+1$ evidences starting from a state s in the following way: for each $0 \leq i < k$ it computes the set of successors of s that satisfy $E^{>i}\theta$ and do not satisfy $E^{>i+1}\theta$ (line 7)[1] and from each of these successors it recursively computes a set of $i+1$ distinct evidences. The variable *count* counts the number of evidences that have already been computed. If they are enough (that is, $count > k$) then the procedure ends; otherwise it repeats the computation for $i+1$. The *load-balancing* applied to the successors of a node when $i = k$ (lines 9) by giving $k' \leftarrow \left\lceil \frac{(k-count)}{|SUCC(s,i)|} \right\rceil$ is used to avoid to enter in a loop of states all satisfying $E^{>k}\theta$. Note that, the procedure generates the evidences by trying to differentiate them immediately on the successors of each state, thus generating a wide evidence tree.

Our experiments have been executed on a workstation equipped with a CPU Intel Core 2 duo 2.33GHz, with 4GB of RAM and Linux 2.6 operating system. The tool software, other tests and other details (that due to lack of space we have not included in the paper) can be found at `http://gradedctl.dia.unisa.it`.

As said into the introduction, our motivation to introduce this new graded logic and use it to model check, is mainly to reduce the debugging times: thus to measure the performance of our algorithm/approach one should take into consideration also the *potential* savings obtained in avoiding to go through the cycle Check/Analyze/Fix again. We report now some results measured on CPU time and BDD size on various examples. The examples to test our algorithm have been chosen from the official site of NuSMV. As said before there is no exact method to compare the results of our approach with respect to the classical one. We have chosen to proceed as follows: for each example, we have considered a graded-CTL specification φ and have measured the CPU time and the BDD size to model-check it by varying the values of the constants grading the quantifiers in φ. The first row of each example is the result of running NuSMV on the

Table 1.

k	**syncarb5**				**p-queue**				**dme1.16**	
	$E^{>k}\mathcal{X}(e2.Token)$		$E^{>k}[!e2.Token\ \mathcal{U}\ e2.Token]$		$E^{>k}\mathcal{G}(out\_1[1]=0)$		$A^{\leq k}\mathcal{F}(out\_1[1]=0)$		$E^{>k}\mathcal{X}e\_1.req$	
	Time	♯BDD	Time	♯BDD	Time	♯BDD	Time	♯BDD	Time	♯BDD
–	0,000	1248	0,000	1255	0,024	33717	0,020	33717	1,320	244734
0	0,000	1248	0,000	1255	0,024	33866	0,020	33866	1,320	244734
1	0,000	1600	0,000	1660	0,056	82361	0,020	41377	1,460	320986
2	0,000	1602	0,000	1694	0,060	92013	0,020	41377	1,664	416145
3	0,000	1603	0,000	1695	0,060	92016	0,024	41377	1,740	443526
5	0,000	1607	0,000	1702	0,060	101671	0,024	41377	2,064	572134
8	0,000	1613	0,000	1714	0,064	102708	0,024	41378	2,736	783876
10	0,000	1614	0,000	1716	0,064	102710	0,024	41380	2,860	830999
15	0,000	1614	0,000	1721	0,068	102715	0,024	41385	3,304	949405
20	0,000	1623	0,000	1738	0,072	104081	0,028	41390	4,248	1168181
50	0,000	1614	0,000	1798	0,104	107395	0,028	41420	9,689	1732683

[1] The $forwardImg$, giving the set of successors, can be computed analogously to the computation of the $pre-image$.

given specification with (classical) non-graded quantifiers. The second row is the result of running our tool on the given specification with all the grading values to zero: this to underline that there is no overhead due to the grading, on formulas semantically equivalent. All the remaining rows are the results obtained by using other values of the grading quantifiers, to see how the complexity increases, related to those values.

The Table 1 reports the results for formulas with one graded quantifier. In two cases (*dme1.16* and *p-queue* with specification $E^{>k}\mathcal{G}(out\_1[1] = 0)$) one can observe a small increasing in the CPU time, with respect to the time needed for the corresponding CTL formula. In the remaining examples, the CPU time remains almost the same (most are approximatively equal to 0) for all the specifications and for all the values of the constant k. Moreover, also the BDD size increases

Table 2.

k_1	k_2	**robot** $A^{\leq k_1}\mathcal{G}(pT1.start \rightarrow A^{\leq k_2}\mathcal{G}\, !pT1.finish)$		**syncarb5** $A^{\leq k_1}\mathcal{X}A^{\leq k_2}\mathcal{F}(e1.ack\_out \wedge e1.Persistent)$		k_1	k_2	k_3	**abp8** $A^{\leq k_1}\mathcal{G}(A^{\leq k_2}\mathcal{F}(sender.state = get) \wedge A^{\leq k_3}\mathcal{F}(receiver.state = deliver))$		$E^{>k_1}\mathcal{G}(E^{>k_2}\mathcal{F}(sender.state = get) \wedge E^{>k_3}\mathcal{F}(receiver.state = deliver))$	
		Time	♯BDD	Time	♯BDD				Time	♯BDD	Time	♯BDD
–	–	0, 072	59240	0, 000	1321	–	–	–	3, 804	1743784	3, 052	1616154
0	0	0, 072	59449	0, 000	1321	0	0	0	3, 804	1743794	3, 296	1616164
1	0	0, 140	96107	0, 000	1739	1	0	0	3, 832	1750366	5, 316	679676
2	0	0, 152	97197	0, 000	1758	2	0	0	3, 832	1750366	6, 808	1541032
5	0	0, 172	98288	0, 000	1791	5	0	0	3, 836	1750366	11, 765	845117
10	0	0, 188	104991	0, 000	1829	10	0	0	3, 840	1750366	22, 321	845117
0	1	0, 148	96705	0, 000	1739	0	1	1	3, 832	1750366	8, 237	1345388
1	1	0, 160	96705	0, 000	1739	1	1	1	3, 840	1750366	10, 253	679137
2	1	0, 224	103426	0, 000	1828	5	1	1	3, 844	1750366	17, 177	845117
5	1	0, 236	104517	0, 000	1861	10	1	1	3, 848	1750366	27, 142	845117
10	1	0, 264	111220	0, 000	1899	0	3	1	3, 832	1750366	11, 929	1296220
0	2	0, 156	97795	0, 000	1758	1	3	1	3, 840	1750366	13, 957	679137
1	2	0, 220	103426	0, 000	1828	5	3	1	3, 860	1750366	20, 481	845117
2	2	0, 160	97795	0, 000	1758	10	3	1	3, 872	1750366	31, 722	845117
5	2	0, 240	104517	0, 000	1880	0	5	1	3, 864	1750366	15, 273	1320804
10	2	0, 268	111220	0, 000	1918	1	5	1	3, 872	1750366	17, 385	679137
0	5	0, 168	98886	0, 000	1792	10	5	1	3, 872	1750366	34, 058	845117
1	5	0, 240	104517	0, 000	1862	0	1	3	3, 860	1750366	11, 725	1345400
2	5	0, 244	104517	0, 000	1880	1	1	3	3, 864	1750366	13, 717	679137
5	5	0, 188	98886	0, 000	1792	10	1	3	3, 872	1750366	30, 718	845117
10	5	0, 276	105589	0, 000	1919	0	3	3	3, 860	1750366	14, 737	1658777
0	10	0, 196	105589	0, 000	1834	1	3	3	3, 868	1750366	16, 677	1026555
1	10	0, 256	111220	0, 000	1904	10	3	3	3, 876	1750366	32, 950	845117
2	10	0, 272	11220	0, 000	1922	0	5	3	3, 868	1750366	18, 141	1316462
5	10	0, 284	105589	0, 000	1921	1	5	3	3, 872	1750366	20, 145	679137
10	10	0, 220	105589	0, 000	1834	10	5	3	3, 880	1750366	37, 318	845117
50	10	0, 528	159213	0, 000	1946	10	10	10	3, 876	1750366	61, 120	845117
50	50	0, 564	152510	0, 004	1981	50	50	50	3, 884	1750366	310, 011	845117

very slowly. The Table 2 reports the results for more complex formulas, with two or three graded quantifiers. For some examples, one can observe that the CPU time and the BDD size are not affected by the value of the constant, while in the examples *robot* and *abp8* with the second specification, one can notice an increasing of the CPU time which grows.

Finally, let us note that we have used values for the grading constants in the range $[0, \ldots, 50]$, but we think that in practice a reasonable upper bound should be much smaller, as high values would mean a number of counterexamples too big to deal with.

6 Conclusions and Future Works

In this paper we have considered an expressive extension of classical CTL and have proved that the SAT problem is ExpTime-complete when the values in the formula are represented in unary. An open problem is hence to establish the complexity of the same problem when the values are in binary (recall that for the case of the model checking problem the complexity for Graded CTL is the same as CTL, even when the constants are expressed in binary). We have also shown symbolic algorithms for model checking against specifications given in this logic and have extended the NuSMV model checker to accept such specifications. The experimental results have indeed shown that the usual performances of NuSMV on classical CTL specifications are still retained.

Besides its augmented expressiveness, with respect to classical CTL specifications, the motivation to study this logic is in the possibility of reducing debugging time. As it is well established, the generation of more than one counterexample is highly desirable, though the size of the counterexamples and their poor human-readability becomes more and more crucial, when more counterexamples are generated. One of the main problems arising is that of determining counterexamples which are as much significant as possible. An approach to this problem is given by a different semantics of graded-CTL which allows to distinguish system behaviors, satisfying a formula, that are completely disjoint. This semantics, called *edge-disjoint* semantics, has been defined in [FNP10] and requires the edge-disjointness of the paths satisfying a path-formula. With this approach one can detect different counterexamples which depend on different and completely independent "errors" in the model. Moreover, this edge-disjointness requirement turns out to be useful also when fault tolerance is required. For this reason, it should be useful to investigate the symbolic model checking problem for edge-disjoint semantics of graded-CTL. Another aspect to consider is the concurrency: partial order techniques have been used to avoid the state explosion, c.f. [God90, GKPP99], also in symbolic model checking, [KGS06]. It would be worthwhile to investigate whether such approach can be usefully applied in our setting to get counterexamples which do not differ only for the interleaving ordering of concurrent actions.

Finally, let us observe that during the model checking process, the system is sometimes abstracted in order to deal with the state explosion problem of the

Kripke structures. But since graded-CTL is a logic based on counting paths satisfying given properties, it is not preserved under most of the usual abstractions (since all of these modify the number of paths in the Kripke structure). Clearly, this situation is inherent to every logic based on counting paths and/or successors such as graded-μ-calculus [KSV02] and graded-HML (Hennessy-Milner Logic) [CDL99]. However in [CDL99] the *resource bisimulation* has been introduced to "discriminate processes according to the number of different computations they can perform to reach specific states". It can be easily shown that graded-CTL logic preserves the resource bisimulation.

Acknowledgments. The authors thank Francesco Sorrentino for his help and his enthusiasm in the early stage of the implementation of our tool.

References

[BCM+90] Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science (LICS 1990), pp. 428–439. IEEE Computer Society Press, Los Alamitos (1990)

[BFG+97] Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. Formal Methods in System Design 10(2/3), 171–206 (1997)

[BGS06] Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. Formal Methods in System Design 28(1), 37–56 (2006)

[BMM09] Bianco, A., Mogavero, F., Murano, A.: Graded computation tree logic. In: LICS 2009 (2009)

[Bry92] Bryant, R.E.: Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys 24(3), 293–318 (1992)

[CCG+02] Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An openSource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)

[CDL99] Corradini, F., De Nicola, R., Labella, A.: Models of nondeterministic regular expressions. J. Comput. Syst. Sci. 59(3), 412–449 (1999)

[CE81] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)

[CE82] Clarke, E.M., Emerson, E.A.: Usig branching time temporal logic to synthesize synchronization skeletons. Science of Computer Programming 2, 241–266 (1982)

[CG07] Chechik, M., Gurfinkel, A.: A framework for counterexample generation and exploration. International Journal on Software Tools for Technololy Transfer 9(5), 429–445 (2007)

[CIW+01] Copty, F., Irron, A., Weissberg, O., Kropp, N.P., Kamhi, G.: Efficient debugging in a formal verification environment. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 275–292. Springer, Heidelberg (2001)

[DRS03] Dong, Y., Ramakrishnan, C.R., Smolka, S.A.: Model checking and evidence exploration. In: Workshop on Model Based Development: Features, Components and Architectures (ECBS 2003), pp. 214–223 (2003)

[EMSS92] Emerson, E.A., Mok, A.K., Sistla, A.P., Srinivasan, J.: Quantitative temporal reasoning. Real-Time Systems 4(4), 331–352 (1992)

[Fin72] Fine, K.: In so many possible worlds. Notre Dame Journal of Formal Logic 13(4), 516–520 (1972)

[FNP08] Ferrante, A., Napoli, M., Parente, M.: CTL model-checking with graded quantifiers. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 18–32. Springer, Heidelberg (2008)

[FNP10] Ferrante, A., Napoli, M., Parente, M.: Model checking for graded CTL. Fundamenta Informaticae (to appear, 2010); Journal version of [FNP 2008]

[GKPP99] Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A partial order approach to branching time logic model checking. Information and Computation 150(2), 132–152 (1999)

[GMV99] Ganzinger, H., Meyer, C., Veanes, M.: The two–variable guarded fragment with transitive relations. In: LICS, pp. 24–34 (1999)

[God90] Godefroid, P.: Using partial orders to improve automatic verification methods. In: Clarke, E., Kurshan, R.P. (eds.) CAV 1990. LNCS, vol. 531, pp. 176–185. Springer, Heidelberg (1991)

[GOR97] Grädel, E., Otto, M., Rosen, E.: Two–variable logic with counting is decidable. In: LICS, pp. 306–317 (1997)

[GPP07] Gentilini, R., Piazza, C., Policriti, A.: Symbolic graphs: Linear solutions to connectivity related problems. Algorithmica 50(1), 120–158 (2007)

[HB91] Hollunder, B., Baader, F.: Qualifying number restrictions in concept languages. In: KR, pp. 335–346 (1991)

[KGS06] Kahlon, V., Gupta, A., Sinha, N.: Symbolic model checking of concurrent programs using partial orders and on-the-fly transactions. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 286–299. Springer, Heidelberg (2006)

[KSV02] Kupferman, O., Sattler, U., Vardi, M.Y.: The complexity of the graded μ–calculus. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392, pp. 423–437. Springer, Heidelberg (2002)

[Mar05] Marrero, W.: Using bdds to decide ctl. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 222–236. Springer, Heidelberg (2005)

[McM93] McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Dordrecht (1993)

[PST00] Pacholski, L., Szwast, W., Tendera, L.: Complexity results for first–order two–variable logic with counting. SIAM Journal of Computing 29(4), 1083–1117 (2000)

[PSV05] Pan, G., Sattler, U., Vardi, M.Y.: Bdd-based decision procedures for the modal logic k. Journal of Applied Non-classical Logics 49 (2005)

[Som05] Somenzi, F.: Cudd: Cu decision diagram package, release 2.4.1 (2005), `http://vlsi.colorado.edu/~fabio/CUDD`

[Tob01] Tobies, S.: PSPACE reasoning for graded modal logics. Journal Log. Comput. 11(1), 85–106 (2001)

[VW86] Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. J. of Computer and System Sciences 32(2), 183–221 (1986)