Decision Procedures for CTL*

Oliver Friedmann and Markus Latte

Dept. of Computer Science, University of Munich, Germany

Abstract. We give an overview over three serious attempts to devise an effective decision method for CTL*, namely Emerson and Jutla's automata-theoretic decision procedure, Reynolds' tableau search method and our recent approach based on an infinite tableau system with natural rules and with global conditions on the branches.

1 Introduction

The full branching-time temporal logic CTL* is an important tool for the specification and verification of reactive systems [7] and of agent-based systems [9], for program synthesis [11], etc. Emerson and Halpern have introduced CTL* [2] as a formalism which supersedes both the branching-time logic CTL and the linear-time logic LTL. As much as this has led to an easy unification of CTL and LTL, it has also proved to be quite a difficulty in obtaining decision procedures for this logic.

In this paper we present an overview over three serious attempts to devise an effective decision method for CTL*. The first was automata-theoretic [4], requiring the determinisation of ω-word automata resulting from linear-time formulas. A series of improvements in this part has eventually led to Emerson and Jutla's automata-theoretic decision procedure [3]. The second serious attempt was Reynolds' tableau method [12] based on finite tableaux with loops and every node labeled with sets of Hintikka-like sets. Third, we present our infinite tableau system [6] with natural rules and with global conditions on the branches.

The rest of the paper is organised as follows. Sect. 2 recalls CTL*. Sect. 3 describes the decision method by Emerson and Jutla [3], Sect. 4 outlines the recent tableau method by Reynolds [12] and Sect. 5 presents our (joint work with Martin Lange) new infinite-tableaux method [6]. Sect. 6 highlights the advantages and disadvantages of the presented approaches to effectively decide CTL* satisfiability.

2 CTL*

Let $\mathcal{P}$ be a countably infinite set of propositional constants. A transition system is a tuple $\mathcal{T} = (\mathcal{S}, \rightarrow, \lambda)$ with $(\mathcal{S}, \rightarrow)$ being a directed graph, $\mathcal{S}$ being a set of states, and $\lambda : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ being a labeling function. We assume transition systems to be total, i.e. every state has at least one successor. A *path* π in $\mathcal{T}$ is an infinite

sequence of states $s_0, s_1, \ldots$ s.t. $s_i \to s_{i+1}$ for all i. With π^k we denote the suffix of π starting with state s_k, and $\pi(k)$ denotes s_k in this case.

Branching-time temporal formulas are given by the following grammar.

$$\varphi \quad ::= \quad q \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{X}\varphi \mid \varphi \mathtt{U} \varphi \mid \mathtt{E}\varphi$$

where $q \in \mathcal{P}$. Formulas of the form q or $\neg q$ are called *literals*. The *size* $|\varphi|$ of a formula φ is defined as usual, i.e. as the size of the set of subformulas in φ.

Other propositional constructs like $\mathtt{tt}, \mathtt{ff}, \vee, \rightarrow$ are derived as usual, and so are the temporal ones $\varphi \mathtt{R} \psi := \neg(\neg\varphi \mathtt{U} \neg\psi)$, $\mathtt{G}\varphi = \mathtt{ff}\mathtt{R}\varphi$, $\mathtt{F}\varphi = \mathtt{tt}\mathtt{U}\varphi$, and $\mathtt{A}\varphi := \neg\mathtt{E}\neg\varphi$.

These formulas are interpreted over paths π of a TS $\mathcal{T} = (\mathcal{S}, s^*, \rightarrow, \lambda)$.

$$\begin{array}{lll}
\mathcal{T}, \pi \models q & \text{iff } q \in \lambda(\pi(0)) \\
\mathcal{T}, \pi \models \neg\varphi & \text{iff } \mathcal{T}, \pi \not\models \varphi \\
\mathcal{T}, \pi \models \varphi \wedge \psi & \text{iff } \mathcal{T}, \pi \models \varphi \text{ and } \mathcal{T}, \pi \models \psi \\
\mathcal{T}, \pi \models \mathtt{X}\varphi & \text{iff } \mathcal{T}, \pi^1 \models \varphi \\
\mathcal{T}, \pi \models \varphi \mathtt{U} \psi & \text{iff } \exists k \in \mathbb{N}, \mathcal{T}, \pi^k \models \psi \text{ and } \forall j < k : \mathcal{T}, \pi^j \models \varphi \\
\mathcal{T}, \pi \models \mathtt{E}\varphi & \text{iff } \exists \pi', \text{ s.t. } \pi'(0) = \pi(0) \text{ and } \mathcal{T}, \pi' \models \varphi
\end{array}$$

It is well-known and easy to see that every formula is equivalent to one in positive normal form. CTL* is the set of all branching-time formulas which are state formulas. A CTL* formula φ is *satisfiable* if there is a transition system $\mathcal{T}$ s.t. $\mathcal{T}, \pi \models \varphi$ for some path π.

3 Emerson-Jutla-Method

Let φ be a CTL*-formula. The decision procedure by Emerson et. al. [5, 3] consists of three parts. First, φ is transformed into a certain normal form. Second, they construct a tree automaton which accepts—roughly speaking—all tree-like models of φ and, finally, an emptiness-test is applied for this automaton.

The said normal form is a disjunctive normal form built on top of formulas of the type $\mathtt{A}\lambda$, $\mathtt{E}\lambda$ and $\mathtt{AGE}\lambda$ where λ stands for an LTL-formula, i.e. a CTL*-formula without any path quantifier. After turning φ into positive normal form, every inconveniently quantified subformula is pulled out. For instance, we have $\varphi[\mathtt{E}\lambda/p] \equiv \varphi \wedge \mathtt{AGE}(p \rightarrow \lambda) \wedge \mathtt{AG}(\neg p \rightarrow \neg\lambda)$. The size of the resulting formula, say ψ, is linear in $|\varphi|$. So, the transformation introduces new propositional variables. Any model of φ can be extended to a model of ψ. Vice versa, a model of ψ is a model of φ. It is known that for a LTL-formula λ, there is non-deterministic Büchi automaton (on ω-words) which accepts exactly all runs which model λ. The size of such an automaton is exponential in $|\lambda|$. As the existential path quantifier commutes with the nondeterminism, we can construct a tree automaton which accepts the models of ψ using *deterministic* automata for each LTL-formula λ which occur as $\mathtt{A}\lambda$ in ψ. A non-deterministic automaton for an universally quantified formula is insufficient because its runs might even differ on the common prefix of different branches.

Emerson et. al. use deterministic Street automata [5] to handle LTL-formulas. Alternatively, one could also use deterministic parity automata [10] for instance.

In both settings, the kind of acceptance condition for the deterministic automaton for the LTL-subformulas also determines the acceptance condition class of the tree automaton. The size of the respective tree automaton is doubly exponential in $|\psi|$, and its index is exponential in $|\psi|$. The runtime of the respective emptiness tests are polynomial in the size and exponential in the index of the tree automaton. Therefore, this method leads to a decision procedure with an optimal [15] worst-case time complexity, that is, doubly exponential in $|\varphi|$. However, the determinization of Büchi automata has a drawback, as Emerson [1] notes himself: "*... due to the delicate combinatorial constructions involved, there is usually no clear relationship between the structure of the automaton and the candidate formula.*"

4 Reynolds' Tableaux

Reynolds' recent decision method [12] is based on finite tableaux. Satisfiability is then expressed in terms of the existence of a tableau for a given CTL* formula. The nodes in these tableaux are labeled with sets of sets of formulas. The inner sets – called *hues* – essentially form a Hintikka-style closure of subformulas and correspond to a full path in some model of the given formula. The collection of the hues of one node – called a *colour* – captures all hues at the hypothetical world associated with the node of the tableau that correspond to different full paths. Successor colours c' of a node are required to be consistent with their parent colour c in the sense that every hue h' in c' is a hypothetical successor world of some hue h in c.

Finiteness is obtained through looping back: the so-called "good" loops essentially witness the fact that a greatest fixpoint is unfolded infinitely often in a thread of hues while every occurring eventuality is satisfied after a finite number of steps. On the other hand, "bad" loops tell the tableau-searcher that it is reasonable to stop to dive more deeply into an unexpanded branch with no good loops yet.

Tableau-search then relies on tableau-building, loop-checking and backtracking. Good loops are detected by a model-checking-style algorithm that can be performed in polynomial time. Regarding bad loops, Reynolds says that "*... we are only able to give some preliminary results on mechanisms for tackling repetition.*". In fact, instead of detecting bad loops directly, Reynolds just limits the allowed length of a branch without good loop by the small model property obtained from [4] which is of doubly exponential complexity.

In terms of worst-case complexity – which is known to be doubly exponential for CTL* satisfiability –, Reynolds method is optimal. However, as the detection of bad loops is realized by branch-length limitation, even small unsatisfiable formulas can lead to an extremely large search space. Finally, Reynolds reports of a prototype implementation of his tableau decision procedure [12]. This implementation is, however, not publicly available, and tests are only performed on single short formulas such that no asymptotic behaviour can be inferred from those results.

5 Infinite Tableaux

Our recent approach [6] is formulated as a calculus of infinite tableaux with natural rules and with global conditions on their branches. Each node in a tableau is labeled with a so-called *sequent* which is a set of consistent literals and so-called *blocks*: a block is a path-quantified set of subformulas of the CTL* formula for which satisfiability is to be decided. The intended interpretation of an `E`- resp. `A`-quantified set is an `E`- resp. `A`-quantification over the conjunction resp. disjunction of all formulas contained in the block. The sequent then is to be interpreted as the conjunction over all contained blocks and literals.

The tableau-rules are straight-forward: for every kind of formula occurring in one of the path-quantified blocks, there is a corresponding rule that essentially decomposes the formula into the immediate subformulas and proceeds with these while being locally sound and complete. Rules that are to be applied to fixpoint formulas simply replace them by their unfoldings. The only branching rules are the so-called *modal rules* that can only be applied iff every formula in a path-quantified block is of the form $\mathtt{X}\varphi$. For every occurring `E`-quantified block, there is a distinct subgoal that consists of the respective `E`-quantified block and every `A`-quantified block with all formulas $\mathtt{X}\varphi$ being replaced by φ. Applying the tableau-rules backwardly results in a possibly infinite tree with local correctness in the sense that the sequent of a node is satisfiable iff all successors are as well.

Additionally, there are global conditions that are to be satisfied on every infinite branch to handle the satisfaction of eventualities after a finite number of steps. Every infinite sequence of connected blocks in a branch is called *trace*. Given a trace, every infinite sequence of connected formulas in the trace is called *thread*. Note that every trace is eventually either labeled with `E` or `A` and every thread is eventually solely unfolding an unique least or greatest fixpoint. We say that a trace is *bad* iff it does not contain a greatest fixpoint thread or it contains a least fixpoint thread and is eventually labeled with `E`. A successful tableau for a given formula now is a tableau-style tree as described before s.t. every infinite branch does not contain a bad trace.

The non-termination of the tableaux raises the question after an effective decision procedure based on this calculus, and it is only here that our method uses automata-theoretic machinery. Branches that do not satisfy the global condition are recognizable by nondeterministic Büchi automata, and we can then use determinisation and complementation in order to reduce the question of existence of a tableau to the problem of solving a doubly exponentially large parity game.

6 Comparison

We compare the three introduced methods with each other with respect to several aspects—c.f. Fig. 1. Every method admits a decision procedure. In the case of an satisfiable formula, the procedure also provides a model.

Usually the branching of a model capture a degree of non-determinism which arises either from an abstraction of the execution of a program or from the

Aspect / Method	Emerson et. al.	Reynolds	ours
Concept	tree-automata	tableau	tableau
Worst-case complexity	2ExpTime	2ExpTime	2ExpTime
Implementation available	no	not public	yes
Model construction	yes	yes	yes
Finite representation by	rabin	small. mod. p.	parity
Out-degree	fix., lin. bounded	var., lin. bounded	var., lin. bounded
Req. small model property	no	yes	no
Derives small model prop.	yes	no	yes
Needs Büchi determ.	yes	no	yes

Fig. 1. Comparison of three decision methods for satisfiability of CTL*-formulas

interaction with the program's environment. Hence, in the case of a program synthesis one might be interested in models which branches as rarely as possible.

Interestingly, the decision methods of Emerson et. al. and ours require the determinization of Büchi word automata. Several approaches are available [5, 13, 8, 14, 10] leading to theoretical optimal decision procedures. However, these apply sophisticated and non-standard data structures which increases the program's complexity. Therefore, it is wishful to find a tableau which does not need the determinization of Büchi automata. Finally, both methods produce a tree-automata which is then tested for emptiness. Our approach does not use tree-automata as such—even though one may argue that the constructed parity games represent tree automata. However, the crucial difference is the separation between the use of tableau-machinery for the characterisation of satisfiability and the use of automata-machinery only to obtain a decision procedure. In particular, we do not translate LTL-formulas into Büchi automata. The other way around, almost all parts of Emerson's tree automaton can be rephrased as a game on sets of subformulas. Only the case which requires the determinization can not be translated directly.

Concludingly, we think that Emerson/Jutla's procedure is not the best candidate to give rise to a practical implementation due the extremely high out-degree of the involved tree automata. Maybe that is one of the reasons why there is no implementation of this decision procedure available (at least to our knowledge).

Reynolds' implementation—which is not publicly available yet—seems to be greatly outperformed by ours. For example, the formula $\mathtt{AG}(\mathtt{EX}p \wedge \mathtt{EX}\neg p) \wedge \mathtt{AG}(\mathtt{G}p \vee (\neg r)\mathtt{U}(r \wedge \neg p))$ apparently cannot be checked for satisfiability by Reynolds' implementation anymore whereas ours takes 0.04s for this task. Essentially, the reason why we claim that our system is more qualified to be utilized in practicate is that while the depth of Reynolds' tableaux is limited by a doubly-exponential function in the size of the formula, our systems stops exploring tableaux whenever a configuration of a sequent and an automaton state reappears. Hence, our way of loop checking is much more involved in the actual structure of all the eventualities.

References

1. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 16, pages 996–1072. Elsevier and MIT Press, New York, USA, 1990.
2. E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. of the ACM*, 33(1):151–178, 1986.
3. E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 2000.
4. E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, 1984.
5. E. Allen Emerson and A. Prasad Sistla. Deciding branching time logic. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 14–24, New York, NY, USA, 1984. ACM.
6. Oliver Friedmann, Markus Latte, and Martin Lange. A Decision Procedure for CTL* Based on Tableaux and Automata. In *Proc. of the 5th Int. Joint Conference on Automated Reasoning*, Edinburgh, UK, 2010. To appear.
7. D. M. Gabbay and A. Pnueli. A sound and complete deductive system for CTL* verification. *Logic Journal of the IGPL*, 16(6):499–536, 2008.
8. D. Kähler and Th. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proc. 35th Int. Coll. on Automata, Languages and Programming, ICALP'08*, volume 5125 of *LNCS*, pages 724–735. Springer, 2008.
9. X. Luo, K. Su, A. Sattar, Q. Chen, and G. Lv. Bounded model checking knowledge and branching time in synchronous multi-agent systems. In *Proc. 4th Int. Conf. on Auton. Agents and Multiagent Syst., AAMAS'05*, pages 1129–1130. ACM, 2005.
10. N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st Symp. on Logic in Computer Science, LICS'06*, pages 255–264. IEEE Computer Society, 2006.
11. A. Pnueli and R. Rosner. A framework for the synthesis of reactive modules. In *Proc. Int. Conf. on Concurrency*, volume 335 of *LNCS*, pages 4–17. Springer, 1988.
12. M. Reynolds. A tableau for CTL*. In *Proc. 16th. Int. Symp. on Formal Methods, FM'09*, volume 5850 of *LNCS*, pages 403–418. Springer, 2009. Long version availabe as technical report of the University of Western Australia.
13. S. Safra. On the complexity of ω-automata. In *Proc. 29th Symp. on Foundations of Computer Science, FOCS'88*, pages 319–327. IEEE, 1988.
14. S. Schewe. Tighter bounds for the determinisation of Büchi automata. In *Proc. 12th Int. Conf. on Foundations of Software Science and Computation Structures, FOSSACS'09*, volume 5504 of *LNCS*, pages 167–181. Springer, 2009.
15. M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th Symp. on Theory of Computing, STOC'85*, pages 240–251, Baltimore, USA, 1985. ACM.