Acacia+, a Tool for LTL Synthesis*

Aaron Bohy[1], Véronique Bruyère[1], Emmanuel Filiot[2], Naiyong Jin[3], and Jean-François Raskin[2]

[1] Université de Mons (UMONS), Belgium
{aaron.bohy,veronique.bruyere}@umons.ac.be
[2] Université Libre de Bruxelles (ULB), Belgium
{jraskin,efiliot}@ulb.ac.be
[3] Synopsys Inc.
nyjin@synopsys.com

Abstract. We present Acacia+, a tool for solving the LTL realizability and synthesis problems. We use recent approaches that reduce these problems to safety games, and can be solved efficiently by symbolic incremental algorithms based on antichains. The reduction to safety games offers very interesting properties in practice: the construction of compact solutions (when they exist) and a compositional approach for large conjunctions of LTL formulas.

Keywords: Church problem, LTL synthesis, antichains, safety games, Moore machines.

1 Introduction

LTL realizability and synthesis are central problems when reasoning about specifications for reactive systems. In the LTL realizability problem, the uncontrollable input signals are generated by the environment whereas the controllable output signals are generated by the system which tries to satisfy the specification against any behavior of the environment. The *LTL realizability problem* can be stated as a two-player game as follows. Let ϕ be an LTL formula over a set P partitioned into O (output signals controlled by Player O, the system) and I (input signals controlled by Player I, the environment). In the first round of the play, Player O starts by giving a subset $o_1 \subseteq O$ and Player I responds by giving a subset $i_1 \subseteq I$. Then the second round starts, Player O gives $o_2 \subseteq O$ and Player I responds by $i_2 \subseteq I$, and so on for an infinite number of rounds. The outcome of this interaction is the infinite word $w = (i_1 \cup o_1)(i_2 \cup o_2) \dots (i_k \cup o_k) \dots$ Player O wins the play if w satisfies ϕ, otherwise Player I wins. The realizability problem asks to decide whether Player O has a winning strategy to satisfy ϕ. The *LTL synthesis problem* asks to produce such a winning strategy when ϕ is realizable.

Due to their high worst-case complexities (2ExpTime-Complete), the LTL realizability and synthesis problems have been considered for a long time only of theoretical interest. Only recently, several progresses on algorithms and efficient data structures

* This work has been partly supported by the ESF project *GASICS*, the ARC project *Game Theory for the Automatic Synthesis of Computer Systems* and the ERC Strarting Grant *inVEST.*

showed that they can also be solved in practice. It follows a renewed interest in these problems and a need for tools solving them. We participate to this research effort by providing a new tool, called Acacia+, that implements recent ideas that offer very interesting properties in practice: efficient symbolic incremental algorithms based on antichains, synthesis of small winning strategies (when they exist), compositional approach for large conjunctions of LTL formulas. This tool can be downloaded or simply used via a web interface. While its performances are better or similar to other existing tools, its main advantage is certainly the generation of *compact strategies* that are easily usable in practice. This aspect of Acacia+ is very useful in several application scenarios, like synthesis of control code from high-level LTL specifications, debugging of unrealizable specifications by inspecting compact counter strategies, and generation of small deterministic automata from LTL formulas (when they exist).

2 Underlying Approach

LTL realizability and synthesis problems have been first studied in the seminal work [2,3]. The proposed solution is based on the costly Safra's procedure for the determinization of Rabin automata [4]. The LTL realizability problem is 2ExpTime-Complete and finite-memory strategies suffice to win the realizability game [5,2]. In [6], a so-called Safraless procedure avoids the determinization step by reducing the LTL realizability problem to Büchi games. It has been implemented in the tool Lily [7,8]. Another Safraless approach has been recently given in [9] for the distributed LTL synthesis problem. It is based on a novel emptiness-preserving translation from LTL to safety tree automata. In [10,11,12], a procedure for LTL synthesis problem is proposed and implemented in the tool Unbeast, based on the approach of [9] and symbolic game solving with BDDs.

Our tool Acacia+ is based on several work by some authors of this paper [13,14]. In [13], a construction similar to [9] is proposed for LTL realizability and synthesis by a reduction to *safety games*. In this approach, the formula ϕ is first translated into an equivalent universal coBüchi word automaton, and then into an equivalent universal K-coBüchi automaton provided K is taken large enough (for which any infinite word w is accepted iff all runs labeled by w visit at most K accepting states). The latter automaton can be easily determinized with a variant of the classical subset construction, and the LTL synthesis problem is then solved *on the fly* as a safety game $G(\phi, K)$.

This approach offers very interesting properties in practice. (1) Checking the existence of a winning strategy for Player O in the game $G(\phi, K)$ can be done *incrementally* in the games $G(\phi, k)$, with $k = 0, 1, 2, \dots$ (when ϕ is realizable, $k \leq 5$ is often enough in practice). (2) When ϕ is unrealizable, by the determinacy of ω-regular games [15], Player I has a winning strategy for $\neg\phi$. Therefore checking the existence of a winning strategy for Player O is done incrementally in both games $G(\phi, k)$ and $G(\neg\phi, k)$, with $k = 0, 1, 2, \dots$ (3) The structure of $G(\phi, k)$ presents a partial-order on its states that can be used to represent compactly, with *antichains*, the set of *all* winning strategies. These three observations lead to an efficient antichain-based symbolic algorithm for the LTL realizability and synthesis problems, such that the antichain of the winning strategies for each player is obtained by a *backward fixpoint* computation from the safe configurations of $G(\phi, k)$ [13]. Moreover when ϕ is realizable, the computed antichain allows the construction of a *compact* Moore machine representing a winning strategy for Player O (for Player I when ϕ is unrealizable). This algorithm is called *monolithic* in this paper.

In [13], the authors also propose two *compositional* algorithms for LTL formulas of the form $\phi = \phi_1 \wedge \cdots \wedge \phi_n$. The LTL realizability and synthesis problems are solved by first solving them separately for each conjunct ϕ_i, and then by composing the solutions according to the *parenthesizing* of ϕ. The first algorithm follows a compositional *backward* approach such that at each stage of the parenthesizing, the antichains W_i of the subformulae ϕ_i are computed backward and the antichain of the formula ϕ itself is also computed backward from the W_i's. In this approach, *all* the winning strategies for ϕ (for a fixed k) are computed and compactly represented by the final antichain. This backward approach is optimized by considering relevant input signals only, called *critical signals* [14]. The second algorithm follows a compositional *forward* approach such that at each stage of parenthesizing, antichains are computed backward as explained before, except at the last stage where a forward algorithm seeks for *one* winning strategy by exploring the game arena on the fly in a forward fashion.

While the approaches detailed above ([13,14]) have been first implemented in a Perl prototype [16], we have reimplemented them from scratch in a new tool, Acacia+, now made available to the research community. This tool has been developed in Python and C, with emphasis on modularity, code efficiency, and usability. We hope that this will motivate other researchers to take up the available code and extend it. This new tool is detailed in Sect. 3 and typical scenarios of usage are presented in Sect. 4.

3 Tool Description

Programming Choices. Acacia+ is written in Python and C. The C language is used for all the low level operations, while the orchestration is done with Python. The binding between these two languages is realized by the ctypes library of Python.

This separation presents two main advantages. (1) Due to the reduction to k-coBüchi automata and their determinization, we need to manipulate *counting functions* $Q \to \{-1, 0, \ldots, k, k+1\}$ in a way to know if a state $q \in Q$ of the k-coBüchi automaton is reached or not (value -1), and to know the maximal number of visits to an accepting state of runs that end up in q [13]. These counting functions are implemented as bit arrays, together with specific efficient operations implemented in C. Indeed, our algorithms manipulate antichains of counting functions, and operations like membership or intersection are not standard and cannot be implemented using existing libraries on bit arrays. (2) The simplicity of Python increases scalability and modularity and it reduces the risk of errors. Unfortunately, using Python also presents some drawbacks. Indeed, interfacing Python and C leads to light performance overhead. Nevertheless, we believe that the overhead is a small price to pay in comparison with the gain of simplicity.

Our implementation does not use BDDs, as they do not seem to be well adapted in this context, and might be outperformed by the use of antichains [17,18]. We have instead developed a library with a generic implementation of antichains that can easily be reused in another context.

Tool Download and User Interface. Acacia+ can be downloaded at `http://lit2.ulb.ac.be/acaciaplus`. It can be installed under a single command-line version working both on Linux and MacOsX, or used directly via a web interface. The source is licensed under the GNU General Public License. The code is open and can be used, extended and adapted by the research community. For convenience, a number of examples and benchmarks have been pre-loaded in the web interface.

Execution Parameters. Acacia+ offers many execution parameters, fully detailed in the web interface helper.

Formula. Two inputs are required: an LTL formula ϕ and a partition of the atomic signals into the sets I and O. The formula can be given as a single specification, or as a conjunction $\phi_1 \wedge \cdots \wedge \phi_n$ of several specifications (for the compositional approach). Acacia+ accepts both the Wring and LTL2BA input formats, whatever the tool used to construct the automata.

Method. Formulas ϕ are processed in two steps: the first step constructs a universal co-Büchi automaton from ϕ, the second step checks for realizability (synthesis follows when ϕ is realizable). The automaton construction can be done either monolithically (a single automaton is associated with ϕ), or compositionally if the formula is given as a conjunction $\phi_1 \wedge \cdots \wedge \phi_n$ (an automaton is then associated with every ϕ_i). The realizability check can be either monolithic, or compositional (only if ϕ is a conjunction of ϕ_i's). When the automaton construction is compositional and the realizability step is monolithic, the latter starts with the union of all automata obtained from each ϕ_i.

The user can also choose between backward or forward algorithms for solving the underlying safety game. In the case of a compositional realizability check with forward option enabled, each intermediate subgame is solved backward and the whole game is solved forward. The way of parenthesizing ϕ is totally flexible: the user can specify his own parenthesizing or use predefined ones. This parenthesizing enforces the order in which the results of the subgames are combined, and may influence the performances.

Output. The output of the execution indicates if the input formula ϕ is realizable, and in this case proposes a winning strategy for the system. A winning strategy for the environment can also be returned when ϕ is unrealizable (only in case of a monolithic automaton construction). The output strategies are written in Verilog. When they are small ($\leq$ 20 states), the corresponding Moore machines are also drawn in PNG using PyGraphviz. Many statistics about the execution are also output.

Options. For the automaton construction, the user can choose either LTL2BA [19] or Wring [20][1]. Both tools present advantages and drawbacks: LTL2BA works faster whereas Wring provides smaller automata. The user can also choose the starting player for the realizability game[2]. We recall that the implemented algorithms are incremental; an upper bound can be imposed on the values of $k = 0, 1, 2, \ldots$ used in the safety games $G(\phi, k)$. The user can also choose between either realizability check, or unrealizability check, or both in parallel. Finally Acacia+ includes several optimizations like surely losing states detection on the automata, limitation to critical signals, aso . . .. All of them are enabled by default, but can be turned off.

[1] In the latter case, our tool uses the Wring module included in Lily.

[2] In the introduction, the realizability game has been described such that Player O, the system, plays first. A variant is to let Player I, the environment, play first.

4 Application Scenarios

In this section, we describe three typical scenarios of usage of Acacia+. More details and examples can be found on the website of Acacia+.

Controller Synthesis from LTL Specifications. A first classical use of Acacia+ is to construct finite-state controllers that enforce LTL specifications. Such specifications are usually specified by a set of LTL assumptions on the environment, and a set of LTL guarantees to be fulfilled by the controller. Several benchmarks of synthesis problems are available for comparison with other tools: the test suite included in the tool Lily [7,8], a generalized buffer controller from the IBM RuleBase tutorial [21], and the load balancing system provided with the tool Unbeast [10,12]. The performances of Acacia+ are better or similar to other tools, with the advantage of generating compact solutions. As an example, for the load balancing system with 4 clients, Acacia+ first builds a universal coBüchi word automaton with 187 states, and then outputs a winning strategy as a Moore machine with 154 states. This is in contrast with the worst-case complexity analysis announcing a size exponential in 187, and with the winning strategy extracted by Unbeast, whose nuSMV representation is a file of 30MB. As mentioned in [11], extracting small strategies is a challenging problem.

Debugging of LTL Specifications. Writing a correct LTL specification is error prone [22]. Acacia+ can help to debug unrealizable LTL specifications as follows. As explained in Sect. 2, when an LTL specification ϕ is unrealizable for Player O, then its negation $\neg\phi$ is realizable for Player I. A winning strategy of Player I for $\neg\phi$ can then be used to debug the specification ϕ. Again, Acacia+ often offers the advantage to output readable compact (counter) strategies that help the specifier to correct his specification.

From LTL to Deterministic Büchi automata. As suggested to us by R. Ehlers, following an idea proposed in [6], LTL synthesis tools can be used to convert LTL formulas into an equivalent deterministic Büchi automaton (when possible). The idea is as follows. Let φ be an LTL formula over a set of signals Σ and σ be a new signal not in Σ. Let $I = \Sigma$ and $O = \{\sigma\}$. Then the formula $\phi = (\varphi \leftrightarrow GF\sigma)$ is realizable iff there exists a deterministic Büchi automaton equivalent to φ. Indeed if ϕ is realizable, then the Moore machine M representing a winning strategy for Player O can be transformed into a deterministic Büchi automaton equivalent to φ, by declaring accepting the states of M with output σ. Conversely, if there exists a deterministic Büchi automaton equivalent to φ, this automaton, outputting σ on accepting states, realizes ϕ.

Therefore one can use Acacia+ to construct deterministic automata from LTL formulas (if possible). In [23], the author provides an automated method (together with a prototype) for the NP-complete problem of minimizing Büchi automata [24]. This method is based on a reduction to the SAT problem, and it is benchmarked on several automata obtained from a set of LTL formulas. We use those formulas to benchmark Acacia+ on the LTL to deterministic Büchi automata problem. We obtain very short execution times and the size of the constructed automata is very close to that of a minimal deterministic Büchi automata. The minimum size is indeed reached for 18 among 26 formulas. This shows again that Acacia+ is able to synthesize compact strategies. Finally, let us mention that a similar technique can be used to convert LTL formula into equivalent deterministic parity automata with a fixed number of priorities [6].

References

1. Acacia+, `www.lit2.ulb.ac.be/acaciaplus/`
2. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Principles of Programming Languages, POPL, pp. 179–190. ACM (1989)
3. Abadi, M., Lamport, L., Wolper, P.: Realizable and Unrealizable Specifications of Reactive Systems. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
4. Safra, S.: On the complexity of ω-automata. In: Foundations of Computer Science, FOCS, pp. 319–327. IEEE Computer Society (1988)
5. Pnueli, A., Rosner, R.: On the Synthesis of an Asynchronous Reactive Module. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 652–671. Springer, Heidelberg (1989)
6. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Foundations of Computer Science, FOCS, pp. 531–542. IEEE Computer Society (2005)
7. Lily, `www.iaik.tugraz.at/content/research/design_verification/lily/`
8. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Formal Methods in Computer-Aided Design, FMCAD, pp. 117–124. IEEE Computer Society (2006)
9. Schewe, S., Finkbeiner, B.: Bounded Synthesis. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 474–488. Springer, Heidelberg (2007)
10. Unbeast, `www.react.cs.uni-sb.de/tools/unbeast/`
11. Ehlers, R.: Symbolic bounded synthesis. Formal Methods in System Design 40, 232–262 (2012)
12. Ehlers, R.: Unbeast: Symbolic Bounded Synthesis. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 272–275. Springer, Heidelberg (2011)
13. Filiot, E., Jin, N., Raskin, J.F.: Antichains and compositional algorithms for LTL synthesis. Journal of Formal Methods in System Design 39, 261–296 (2011)
14. Filiot, E., Jin, N., Raskin, F.: Exploiting structure in LTL synthesis. International Journal on Software Tools for Technology Transfer, 1–21 (2012)
15. Martin, D.: Borel determinacy. Annals of Mathematics 102, 363–371 (1975)
16. Acacia, `www.lit2.ulb.ac.be/acacia/`
17. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: A New Algorithm for Checking Universality of Finite Automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 17–30. Springer, Heidelberg (2006)
18. Doyen, L., Raskin, J.-F.: Improved Algorithms for the Automata-Based Approach to Model-Checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 451–465. Springer, Heidelberg (2007)
19. LTL2BA, `www.lsv.ens-cachan.fr/~gastin/ltl2ba/`
20. Wring, `www.iaik.tugraz.at/content/research/design_verification/wring/`
21. IBM RuleBase Tutorial, `www.haifa.ibm.com/projects/verification/rb_homepage/tutorial3`
22. Könighofer, R., Hofferek, G., Bloem, R.: Debugging unrealizable specifications with model-based diagnosis. In: Raz, O. (ed.) HVC 2010. LNCS, vol. 6504, pp. 29–45. Springer, Heidelberg (2010)
23. Ehlers, R.: Minimising Deterministic Büchi Automata Precisely Using SAT Solving. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 326–332. Springer, Heidelberg (2010)
24. Schewe, S.: Beyond hyper-minimisation - Minimising DBAs and DPAs is NP-complete. In: Theory and Applications of Satisfiability Testing, FSTTCS. LIPIcs, vol. 8, pp. 400–411. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)