Winning CaRet Games with Modular Strategies*

Ilaria De Crescenzo and Salvatore La Torre

Dipartimento di Informatica
Università degli Studi di Salerno

Abstract. Recursive state machines are a well-accepted formalism for modelling the control flow of systems with potentially recursive procedure calls. In the open systems setting, i.e., systems where an execution depends on the interaction of the system with the environment, the natural counterpart is two-player recursive game graphs which essentially are recursive state machines where vertices are split into two sets each controlled by one of the players.
We focus on solving games played on such graphs with respect to winning conditions expressed by a formula of the temporal logic CaRet and such that the protagonist can only use modular strategies (modular CaRet games). In a modular strategy, the protagonist may use as memory only the portion of the play which is local to the current activation of the current module. Therefore, every time a module is entered, the memory used by the protagonist gets reset.
The main motivation for considering modular strategies is related to the synthesis of system controllers. In fact, a modular strategy would correspond to a modular controller for the considered system. Modular strategies have been already studied with winning conditions expressed as reachability, safety, Büchi automata or LTL formulas. In this paper we extend these results to non-regular winning conditions by considering specifications expressed in CaRet. In particular, we show that deciding whether the protagonist has a winning modular strategy in a CaRet game is 2ExpTime-complete, that matches the complexity of deciding modular LTL games.

1 Introduction

The interest for games naturally arises in many contexts. In the formal analysis of systems, games are closely related to the *controller synthesis* problem and to the verification of *open systems*, and are useful tools for solving decision problems such as, for example, the model-checking of the μ-calculus formulas (see [6, 9]).

In controller synthesis, given a description of the system where some of the choices depend upon the input and some represent uncontrollable internal nondeterminism, the goal is to design a *controller* that supplies inputs to the system

* This work was partially funded by the MIUR grants FARB 2009-2010 Università degli Studi di Salerno (Italy).

so that the product of the controller and the system satisfies the correctness specification, that clearly corresponds to computing winning strategies in two-player games. See [10] for a survey.

In the open systems setting, for instance, the *Alternating Temporal Logic* allows specification of requirements such as "module A can ensure delivery of the message no matter how module B behaves" [2]; *module checking* deals with the problem of checking whether a module behaves correctly no matter in which environment it is placed [8].

In this paper, we focus on pushdown systems. Pushdown systems accurately model the control flow in programs of sequential imperative programming languages with recursive procedure calls. A large number of hardware and software systems can be captured by this model, such as programs of object oriented languages, systems with distributed architectures and communication protocols. The study of games on such systems has traditionally focused on determining the existence of a winning strategy, that is a mapping that specifies for each play ending into a controlled state the next move such that each resulting play satisfies the winning conditions [5, 12].

Here, we consider *modular strategies* [4], that are strategies where the next move is determined only looking at the local memory of the current activation of the current module. It is known that modular reachability games are NP-complete [4]. Also, modular strategies have been considered for winning conditions expressed as an ω-regular language, using Büchi, Co-Büchi automata or linear temporal logic formulas, and in particular, modular LTL games are known to be 2ExpTime-complete [3].

We extend the results on modular strategies to a more general class of winning conditions. We allow winning conditions expressed as formulas of the temporal logic CaRet [1]. The logic CaRet combines the temporal modalities of LTL with different kinds of successor (*global, abstract* and *caller*) and can express both regular requirements and a variety of non-regular properties such as partial and total correctness of program blocks or inspection of the stack. By using an automaton theoretic approach, we show that solving the modular CaRet games is decidible within double exponential time. Since modular LTL games are already 2ExpTime-hard and CaRet sintactically includes LTL, we get that also modular CaRet games are 2ExpTime-complete.

2 Preliminary

Recursive game graph. A recursive game graph (RGG) is composed of *game modules* that are essentially two-player graphs (i.e., graphs whose vertices are partitioned into two sets depending on the player who controls the outgoing moves) with *entry* and *exit* nodes and two different kinds of vertices: the nodes and the boxes. A *node* is a standard graph vertex and a *box* corresponds to invocations of other game modules in a potentially recursive manner (in particular, entering into a box corresponds to a module *call* and exiting from a box corresponds to a *return* from a module). Each RGG has a distinct game module

which is called the *start* module. A *state* of an RGG is composed by a call stack and a node. The notion of *run* can be defined analogously to the computation of a standard procedural program (the modules corresponding to the procedures). A *play* of an RGG is a run starting from an entry node of the start module.

For a formal definition of an RGG we refer the reader to [4].

Strategies. Given a player P, a *strategy* is a function that associates a move to every run that ends in a node controlled by P. A *modular strategy* consists of a set of local strategies, one for each game module, that are used together as a global strategy for a player . A local strategy for a module can only refer to the local memory of the module, i.e. the sequence of vertices that correspond to internal nodes, call or returns of the module. This sequence corresponds to the portion of the play concerning to the current invocation of the module.

For detailed comments and formal definitions on recursive game graphs, strategies and modular strategies we refer the reader to [4].

Syntax	$\varphi := p \mid \varphi \vee \varphi \mid \neg\, \varphi \mid \bigcirc^g \varphi \mid \bigcirc^a \varphi \mid \bigcirc^- \varphi \mid \varphi\, \mathcal{U}^g \varphi \mid \varphi\, \mathcal{U}^a \varphi \mid \varphi\, \mathcal{U}^- \varphi$ (where $p \in AP \cup \{\text{call, ret, int}\}$)
Semantics	for a word $\alpha = \alpha_1, \alpha_2, \alpha_3, ..., \alpha_n, ... \in \hat{\Sigma}^\omega$ and n $\in \mathbb{N}$: – $(\alpha, n) \models p$ iff $\alpha_n = (X, d)$ and $p \in X$ or $p = d$ – $(\alpha, n) \models \varphi_1 \vee \varphi_2$ iff $(\alpha, n) \models \varphi_1$ or $(\alpha, n) \models \varphi_2$ – $(\alpha, n) \models \neg\varphi$ iff $(\alpha, n) \not\models \varphi$ – $(\alpha, n) \models \bigcirc^g \varphi$ iff $(\alpha, succ^g_\alpha(n)) \models \varphi$, i.e., iff $(\alpha, n+1) \models \varphi$ – $(\alpha, n) \models \bigcirc^a \varphi$ iff $succ^a_\alpha(n)) \neq \bot$ and $(\alpha, succ^a_\alpha(n)) \models \varphi$ – $(\alpha, n) \models \bigcirc^- \varphi$ iff $succ^-_\alpha(n)) \neq \bot$ and $(\alpha, succ^-_\alpha(n)) \models \varphi$ – $(\alpha, n) \models \varphi_1\, \mathcal{U}^b \varphi_2$ (for any $b \in \{g, a, -\}$) iff there is a sequence of position $i_0, i_1, ..., i_k$, where $i_0 = n, (\alpha, i_k) \models \varphi_2$ and for every $0 \leq j \leq k-1, i_j + 1 = succ^b_\alpha(i_j))$ and $(\alpha, i_j) \models \varphi_1$

Fig. 1. Syntax and semantics of CaRet.

CaRet. Let $\Sigma = 2^{AP}$ where AP is a finite set of atomic propositions. We consider the augmented alphabet of Σ that is $\hat{\Sigma} = \Sigma \times \{call, ret, int\}$.

The syntax and the semantics of CaRet are reported in Fig. 1. We refer the reader to [1] for a detailed definition of CaRet.

In this logic, three different notions of successor are used:

- the global-successor ($succ^g$) which is the usual successor function. It points to next node, whatever module it belongs;
- the abstract-successor ($succ^a$) which, for internal moves, corresponds to the global successor and for calls corresponds to the matching returns;
- the caller successor ($succ^-$) which is a "past" modality that points to the innermost unmatched call.

Typical properties that can be expressed by the logic CaRet are pre and post conditions. An example is the formula $\Box[(call \wedge p \wedge p_A) \rightarrow \bigcirc^a q]$. If we assume that all calls to procedure A are characterized by the proposition p_A, the formula expresses that if the pre-condition p holds when the procedure A is invoked, then the procedure terminates and the post-condition q is satisfied

upon the return. This is the requirement of total correctness. Observe the use of the abstract next operator to refer to the return associated with the call.

Modular CaRet *games.* A modular CaRet game is a pair $\langle G, \varphi \rangle$ where G is a RGG whose vertices (nodes, calls and returns) are labeled with a set of propositions and φ is a CaRet formula. Given a modular CaRet game $\langle G, \varphi \rangle$ we want solve the problem of deciding whether there exists a modular strategy such that the resulting plays are guaranteed to satisfy φ.

Detailed comments and formal definitions for modular CaRet games can be found in [13].

3 Solving modular CaRet games

In this section, we briefly sketch our solution to CaRet games. For the omitted details, we refer the interested reader to [13].

Our solution consists of three main steps.

Let $\langle G, \varphi \rangle$ be a modular CaRet game.

The first step consists of constructing from $\langle G, \varphi \rangle$ an equivalent game $\langle G', color \rangle$ with parity winning conditions such that $|G'| = O(2^{|\varphi|})$. We build on the top of the construction given in [1] for model checking CaRet formulas. The main differences are that we apply the construction to the negation of the formula φ instead of to the formula φ directly, and introduce fresh nodes to ensure the correct semantics of the interaction of the two players. Negating the formula is needed to use correctly the construction from [1]. We recall that this construction, such as all the constructions which are tableau based, are nondeterministic and therefore cannot be directly combined with a game graph in a cross product. Starting from the negated formula, we can apply the same tableau construction but now interpreting the nondeterminism as universality, and thus dualizing also the winning conditions we obtain a game graph which is equivalent to the starting one with respect to the considered decision problem. The resulting winning condition is the conjunction of a Büchi condition with a generalized co-Büchi one, that can be simplified to a single pair Rabin condition and thus to an equivalent parity condition.

Also, notice that both G and φ can define context-free languages, and problems such as inclusion and emptiness of intersection are undecidable for context-free languages [7]. Therefore, translating φ into an automaton and then intersecting it with the recursive game graph does not seem feasible.

The second step consists of constructing from the parity game $\langle G', color \rangle$ a two-way alternating parity tree automaton A_{win} similarly to what is done in [3]. The resulting automaton accepts a strategy tree iff it corresponds to a winning modular strategy.

For the last step of our solution, we observe that by using [11] we can convert A_{win} to a one-way nondeterministic tree automaton and take its intersection with the A_{strat} that is a tree automaton accepting strategy trees. Thus, we get a one-way nondeterministic automaton A' which accepts a tree iff it corresponds

to a winning strategy tree. Checking the emptiness of this automaton takes polynomial time in the number of states and exponential in the number of colors in the parity condition [10].

Since the constructed recursive game graph G' is doubly exponential in the formula φ and linear in the recursive game graph G, *color* is constant and LTL games are already 2ExpTime-hard, we get:

Theorem 1. *Deciding modular* CaRet *games is* 2ExpTime*-complete.*

References

1. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Proc. of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'04*, LNCS 2988, pages 467–481. Springer, 2004.
2. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):1–42, 2002.
3. R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for infinite games on recursive graphs. In *Proc. of the 15th International Conference on Computer Aided Verification, CAV'03*, LNCS 2725, pages 67-79. Springer, 2003.
4. R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for recursive game graphs. In *Proc. 9th Intern. Conf. on Tools and Algorithms for the Construction and the Analysis of Systems, TACAS'03*, LNCS 2619, pages 363–378. Springer, 2003.
5. T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *Automata, Languages and Programming, 29th Int'l Coll., ICALP, Malaga, Spain, July 8-13, 2002, Proceedings*, LNCS 2380, pages 704–715. Springer.
6. E. A. Emerson. Model checking and the mu-calculus. In N. Immerman and P. Kolaitis, editors, *Proceedings of the DIMACS Symposium on Descriptive Complexity and Finite Models*, pages 185–214. American Mathematical Society Press, 1997.
7. J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.
8. O. Kupferman, M. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164(2):322–344, 2001.
9. W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.
10. W. Thomas. Infinite games and verification. In *Proceedings of the International Conference on Computer Aided Verification CAV'02*, LNCS 2404, pages 58–64. Springer, 2002.
11. M. Vardi. Reasoning about the past with two-way automata. In *Proc. 14th Intern. Coll. on Automata, Languages, and Programming, ICALP'98*, LNCS 1443 , pages 628–641. Springer, 1998.
12. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, January 2001.
13. Giochi su Grafi Pushdown rispetto a Strategie Modulari. *http://www.dia.unisa.it/professori/latorre/ilaDec/tesi2011.pdf*, 2011