Synthesis of a Hierarchical System from a Library

Benjamin Aminof, Fabio Mogavero, and Aniello Murano

Hebrew University, Jerusalem 91904, Israel.
Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy.
benj@cs.huji.ac.il {mogavero, murano}@na.infn.it

GAMES 2011
September 2011

Extended Abstract[1]

Synthesis is the automated construction of a system from its specification. The basic idea is simple and appealing: instead of developing a system and verifying that it is correct w.r.t. its specification, we use instead an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [8]; the modern approach to this problem was initiated by Pnueli and Rosner who introduced linear temporal logic (LTL) synthesis [25], later extended to handle branching-time specifications, such as μ-calculus [10].

In spite of the rich theory developed for system synthesis in the last two decades, little of this theory has been reduced to practice. In fact, the main approaches to tackle synthesis in practice are either to use heuristics (e.g., [12]) or to restrict to simple specifications (e.g., [24]). Some people argue that this is because the synthesis problem is very expensive compared to model-checking [17]. There is, however, something misleading in this perception: while the complexity of synthesis is given with respect to the specification only, the complexity of model-checking is given also with respect to a program, which can be very large. A common thread in almost all of the works concerning synthesis is the assumption that the system is to be built "from scratch". Obviously, real-world systems are rarely constructed this way, but rather by utilizing many preexisting reusable components, i.e., a library. Using standard preexisting components is sometimes unavoidable (for example, access to hardware resources is usually under the control of the operating system, which must be "reused"), and many times has other benefits (apart from saving time and effort, which may seem to be less of a problem in a setting of automatic - as opposed to manual - synthesis), such as maintaining a common code base, and abstracting away low level details that are already handled by the preexisting components. Another important reason for the limited use of formal synthesis in practice is the fact that synthesized systems are usually monolithic and look very unnatural from the system designer's point of view. Indeed, in classical synthesis algorithms, one usually creates a "flat" system, i.e., a system in which sub-systems may be repeated many times. On the contrary, real-life software and hardware systems are hierarchical (or even recursive) and repeated sub-systems (such as sub-routines) are described only once. While hierarchical systems may be exponentially more succinct than flat ones, it has been shown that the cost of solving questions about them (like model-checking) are in many cases not exponentially higher [4, 5, 11, 6, 23]. Hierarchical systems can also be seen as a special case of recursive systems [1, 2, 7, 13, 14], where the nesting of calls to sub-systems is bounded. However, having no bound on the nesting of calls gives rise to infinite-state systems, and this results in a higher complexity.

[1] A full version of this paper can be found in [3]

In this work we provide a uniform algorithm, for different temporal logics, for the synthesis of hierarchical systems (or, equivalently, *transducers*) from a library of hierarchical systems, which mimics the "bottom-up" approach to system design, where one builds a system by constructing new modules based on previously constructed ones[2]. More specifically, the synthesis process starts by providing the algorithm with a library of available hierarchical components (as well as atomic ones). Then, the system designer provides a specification formula φ of the desired hierarchical component, which is then automatically synthesized using the currently available components as possible sub-components. We show that while hierarchical systems may be exponentially smaller than flat ones, the problem of synthesizing a hierarchical system from a library of existing hierarchical systems is EXPTIME-complete for μ-calculus, and 2EXPTIME-complete for LTL. Thus, this problem is not harder than the classical synthesis problem of flat systems "from scratch". Furthermore, we show that this is true also in the case where the synthesized system has incomplete information about the environment's input. Observe that our algorithm can be used for synthesis of a hierarchical system in many rounds, when at each round the system designer provides the specification of the currently desired module, which is then automatically synthesized using the initial library and the modules constructed in previous iterations. We discuss this application of our algorithm and suggest possible approaches to deal with some of the issues that may arise in this setting.

The problem of automatic synthesis from reusable components has received less attention in the formal verification literature than that given to the issues of specification and correctness of modularly designed systems. Examples of important work on the subject are [9, 20, 21, 26]. To solve our synthesis problem, we use an automata-theoretic approach[17, 19, 15, 18, 16, 22]. However, unlike the classical approach of [25], we build an automaton whose input is not a computation tree, but rather a system description in the form of a *connectivity tree* (inspired by the "control-flow" trees of [21]), which describes how to connect library components in a way that satisfies the specification formula. Taken by itself, our single-round algorithm extends the "control-flow" synthesis work from [21] in four directions. **(i)** We consider not only LTL specifications but also the modal μ-calculus. Hence, unlike [21], where co-Büchi tree automata were used, we have to use the more expressive parity tree automata. Unfortunately, this is not simply a matter of changing the acceptance condition. Indeed, in order to obtain an optimal upper bound, a widely different approach, which makes use of the machinery developed in [5] is needed. **(ii)** We need to be able to handle libraries of hierarchical transducers, whereas in [21] only libraries of flat transducers are considered. **(iii)** A synthesized transducer has no top-level exits (since it must be able to run on all possible input words), and thus, its ability to serve as a sub-transducer of another transducer (in future iterations of a multiple-rounds algorithm) is severely limited (it is like a function that never returns to its caller). We therefore address the problem of synthesizing exits for such transducers.

References

[1] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4):786–818, 2005.

[2] While for systems built from scratch, a top-down approach may be argued to be more suitable, we find the bottom-up approach to be more natural when synthesizing from a library.

[2] R. Alur, S. Chaudhuri, K. Etessami, and P. Madhusudan. On-the-fly reachability and cycle detection for recursive state machines. In *TACAS'05*, LNCS 3440, pages 61–76, 2005.

[3] B. Aminof, F. Mogavero, and A. Murano. Synthesis of hierarchical systems. In *FACS'11*, LNCS 7253 pages 42–59, 2011.

[4] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303, 2001.

[5] B. Aminof, O. Kupferman, and A. Murano. Improved model checking of hierarchical systems. *VMCAI'10*, LNCS 5944, pages 61-77, 2010

[6] B. Aminof, O. Kupferman, and A. Murano. Improved Model Checking of Hierarchical Systems. *International Journal of Information and Computaion*, To appear in 2012.

[7] L. Bozzelli, A. Murano, and A. Peron. Pushdown module checking. *Formal Methods in System Design*, 36(1):65–95, 2010.

[8] A. Church. Logic, arithmetics, and automata. In *Proc. International Congress of Mathematicians, 1962*, pages 23–35. institut Mittag-Leffler, 1963.

[9] L. de Alfaro and T. A. Henzinger. Interface-based design. In *Engineering Theories of Software-intensive Systems. NATO Science Series: Mathematics, Physics, and Chemistry*, 195, pages 83–104. Springer, 2005.

[10] E.A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chap. 16, pg. 997–1072. Elsevier, MIT Press, 1990.

[11] S. Göller and M. Lohrey. Fixpoint logics on hierarchical structures. In *FSTTCS'05*, LNCS 3821, pages 483–494. Springer, 2005.

[12] D. P. Guelev, M. D. Ryan, and P. Y. Schobbens. Synthesising features by games. *Electr. Notes Theor. Comput. Sci.*, 145:79–93, 2006.

[13] A. Ferrante, A. Murano, and M. Parente. Enriched μ-calculus pushdown module checking. In *LPAR'07*, LNCS 4790, pages 438–453. 2007.

[14] A. Ferrante, A. Murano, and M. Parente. Enriched μ-calculi module checking. *Logical Methods in Computer Science*, 4(3:1):1–21, 2008.

[15] O. Kupferman, G. Morgenstern, and A. Murano. Typeness for ω-regular automata. In *ATVA'04*, LNCS 3299, pages 324–333, 2004.

[16] O. Kupferman, G. Morgenstern, and A. Murano. Typeness for ω-regular automata. *International Journal of Foundations of Computer Science*, 17(4):869–883, 2006.

[17] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. of the ACM*, 47(2):312–360, 2000.

[18] S. La Torre, A. Murano, and M. Napoli. Weak muller acceptance condition for tree automata. *Theoretical Computer Science*, 332(1–3):233–250, 2005.

[19] S. La Torre and A. Murano. Reasoning about co-büchi tree automata. In *ICTAC'04*, LNCS 3407, pages 527–542, 2005.

[20] R. Lanotte, A. Maggiolo-Schettini, and A. Peron. Structural model checking for communicating hierarchical machines. In *MFCS*, pages 525–536, 2004.

[21] Y. Lustig and M. Y. Vardi. Synthesis from component libraries. In *FOSSACS'09*, LNCS 5504, pages 395–409, 2009.

[22] A. Murano. *Decision Problems on Tree Automata and Synthesis of Open Timed Systems*. PhD thesis, Universitá degli Studi di Salerno, February 2003.

[23] A. Murano, M. Napoli, and M. Parente. Program complexity in hierarchical module checking. In *LPAR'08*, LNCS 5330, pages 318–332, 2008.

[24] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive designs. In *VMCAI'06*, LNCS 3855, pages 364–380. Springer, 2006.

[25] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL'89*, pages 179–190. ACM Press, 1989.

[26] J. Sifakis. A framework for component-based construction extended abstract. In *SEFM'05*, pages 293–300. IEEE Computer Society, 2005.