Improved Model Checking of Hierarchical Systems *

Benjamin Aminof[1,**], Orna Kupferman[1], and Aniello Murano[2]

[1] Hebrew University, Jerusalem 91904, Israel.
[2] Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy.

Extended Abstract

In model checking, we verify that a system meets its specification by translating the system to a finite state machine (FSM), translating the specification to a temporal-logic formula, and checking that the FSM satisfies the formula [6]. The translation of a high-level description of a system to an FSM involves a painful blow-up, and the size of the FSM is typically the computational bottleneck in model-checking algorithms.

There are several sources of the blow-up that the translation involves. A well-studied source is the ability of components in the system to work in parallel and communicate with each other, possibly using variables (*concurrent FSMs*))[8]. Another source of the blow-up in the translation of systems to FSMs has to do with the ability of a high-level description of a system to reuse the same component in different contexts (say, by calling a procedure). One way to avoid the later blowup is to consider *hierarchical FSMs*, where some of the states of the FSM are boxes, which correspond to nested FSMs. Since many boxes may refer to the same nested FSM, hierarchical FSM's may be exponentially smaller than regular (i.e., flat) FSM's. Unfortunately, the naive approach to model checking such systems is to "flatten" them by repeatedly substituting references to sub-structures with copies of these sub-structures. However, this results in a flat system that is exponential in the nesting depth of the hierarchical system. In [5], Alur and Yannakakis show that for LTL model checking, one can avoid this blow-up altogether, whereas for CTL, one can trade it for an exponential blow-up in the (often much smaller) size of the formula and the maximal number of exits of sub-structures. In other words, while hierarchical FSMs are exponentially more succinct than flat FSMs [4], in many cases the system complexity of the model-checking problem is not exponentially higher in the hierarchical setting! Thus, even more than with the feature of concurrency, here there is clear motivation not to flatten the FSM before model checking it.

The results in [5] set the stage to further work on model-checking of hierarchical systems. As it so happened, however, this line of research has quickly been focused on *recursive systems*, which allow unbounded nesting of components. Having no bound on the nesting gives rise to infinite-state systems. The emergence of software model checking, the natural association of reusability with (possibly recursive) procedure calls, the challenge and abstraction that the infinite-state setting involves, and the neat connection to pushdown automata, have all put recursive systems in the central stage [1–3], leaving the hierarchical setting as a special case. This work hopes to shift some attention back to the hierarchical setting. We suggest a uniform game-based approach for model checking such systems, and argue that the game-based approach enjoys the versatility and advantages it has proven to have in the flat setting. In particular, the game-based approach leads to improved model-checking algorithms. An important conclusion of our work is that we should not hurry to give up the finite-state nature of the hierarchical setting, as it does lead to simpler algorithms, and better complexities than the recursive setting.

In the flat setting, the *game-based* approach reduces the model-checking problem (does a system $\mathcal{S}$ satisfy a branching temporal logic specification ψ?) to the problem of deciding a *two-player game* obtained by taking the product of $\mathcal{S}$ with an alternating tree automaton $\mathcal{A}_\psi$ for ψ [9]. The game-based approach separates the logic-related aspects of the model-checking problem, which are handled in the translation of the specifications to automata, and the combinatorial aspects, which are handled by the game-solving algorithm. Using the game-based approach, it was

* A longer version of this paper appeared in VMCAI 2010.

** This work was partially done while the author was visiting Università degli Studi di Napoli "Federico II", supported by ESF GAMES project, short visit grant n.2789

possible to tighten the time and space complexity of the branching-time model-checking problem [9]. We describe a unified game-based approach for branching-time model checking of hierarchical systems. We define *two-player hierarchical games*, and reduce model checking to deciding such games. In a hierarchical game, an arena may have boxes, which refer to nested sub-arenas. As in the flat setting, one can take the product of a hierarchical system with an alternating tree automaton for its specification, and model checking is reduced to solving the game obtained by taking this product. Now, however, the hierarchy of the system induces hierarchy in the game.

Having introduced the framework, we turn to the main technical contribution of this paper: a new and improved algorithm for solving hierarchical parity games. We now briefly describe it. Consider a hierarchical game $\mathcal{G}$. The idea behind our algorithm is that even though a sub-arena may appear in different contexts, it is possible to extract information about the sub-arena that is independent of the context in which it appears. Formally, for each strategy of one of the players, we can analyze the sub-arena and extract a *summary function*, mapping each exit of the sub-arena to the best color (of the parity condition) that the other player can hope for, given that the current play eventually leaves the sub-arena through this exit. The summary function is independent of the context and has to be calculated only once. The algorithm for solving the game $\mathcal{G}$ then solves a sequence of flat parity games, obtained by replacing sub-arenas by simple gadgets that implement the summary functions.

Related work. The work since [5] was focused on recursive systems, with some exceptions (e.g., [7, 10]). The closest to our work here is [7], which proved that the model-checking problem for the μ-calculus and hierarchical systems is PSPACE-complete (as opposed to the recursive setting, in which μ-calculus model checking is EXPTIME-complete). However, the μ-calculus model-checking algorithm that our approach induces enjoys several advantages with respect to the one in [7]. The first one is the complexity. Beyond having a polynomial space complexity, the time complexity of our algorithm is usually much better than the one that follows the "flattening" approach, and in all cases it is much better than the one in [7]. Second, recall that we reduce the μ-calculus model-checking to solving hierarchical parity games and our algorithm solves the latter by solving a sequence of (non-hierarchical) parity games. As such, it can benefit from existing and future algorithms and tools for solving parity games. Third, the algorithm presented in [7] does not deal directly with hierarchical systems. Rather, it considers *straight line programs* (SLP). Translating a hierarchical system to an SLP is not hard (indeed, it involves a quadratic blow-up), but it messes-up the direct relationship between the structure of the hierarchical system and the game. This relationship is crucial in understanding the output of the model-checking procedure, by means of counterexamples, and to practically use our hierarchical game algorithm in verification tools.

References

1. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM TOPLAS*, 27(4):786–818, 2005.
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. Springer, 2005.
3. R. Alur, K. Etessami, and M. Yannakakis. Analysis of recursive state machines. In *CAV'01*, LNCS 2102, pages 207–220. Springer, 2001.
4. R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *ICALP'99*, LNCS 1644, pages 169–178. Springer, 1999.
5. R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303, 2001.
6. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
7. S. Göller and M. Lohrey. Fixpoint logics on hierarchical structures. In *FSTTCS'05*, LNCS 3821, pages 483–494. Springer, 2005.
8. D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *J. of Inf. & Comp.*, 173:1–19, 2002.
9. 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.
10. A. Murano, M. Napoli, and M. Parente. Program complexity in hierarchical module checking. In *LPAR'08*, LNCS 5330, pages 318–332. Springer, 2008.