The Pushdown Module Checking Saga

Aniello Murano

Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy.

A main distinction in system modeling is between closed systems, whose behavior is totally determined by the program, and open systems, which are systems where the program interacts with an external environment [HP85, Hoa85]. In order to check whether a closed system satisfies a required property, we translate the system into a formal model (such as a transition system), specify the property with a temporal-logic formula (such as *CTL* [CE81], CTL^* [EH86], and μ-calculus [Koz83]), and check formally that the model satisfies the formula. This process is called *model checking* ([CE81, QS81]). Checking whether an open system satisfies a required temporal logic formula is much harder, as one has to consider the interaction of the system with all possible environments.

In this paper, we consider open systems which are modeled in the framework introduced by Kupferman, Vardi, and Wolper. Concretely, in [KV96, KVW01], an open finite-state system is described by an extended transition system called a *module*, whose set of states is partitioned into *system states* (where the system makes a transition) and *environment states* (where the environment makes a transition). Given a module $\mathcal{M}$ describing the system to be verified and a temporal logic formula φ specifying the desired behavior of the system, the problem of model checking a module, called *module checking*, asks whether for all possible environments $\mathcal{M}$ satisfies φ. In particular, it might be that the environment does not enable all the external choices. Module checking thus involves not only checking that the full computation tree obtained by unwinding $\mathcal{M}$ (which corresponds to the interaction of $\mathcal{M}$ with a maximal environment) satisfies the specification φ, but also that every tree obtained from it by pruning children of environment nodes (this corresponds to different environment choices) satisfies φ. For example, consider an ATM machine that allows customers to deposit money, withdraw money, check balance, etc. The machine is an open system and an environment for it is a subset of the set of all possible infinite lines of customers, each with their own plans. Accordingly, there are many different possible environments to consider.

The finite-state system module checking problem for *CTL* and CTL^* formulas has been investigated in [KV96, KVW01, MNP08], while for the propositional μ-calculus ones it has been investigated in [FM07, FMP08]. In all these cases, it has been shown that module checking is exponentially harder than model checking. However, an interesting aspect of those results is that they bear on the corresponding automata-based results for closed systems [KVW00], which gives the hope for practical implementations and applications. The finite-state module checking idea has been also extended to environments with *imperfect information* [KV97]. In this framework, every state of the module is a composition of *visible* and *invisible* variables, where the latter are hidden from the environment.

While a composition of a module $\mathcal{M}$ with an environment with perfect information corresponds to arbitrary disabling of transitions in $\mathcal{M}$, the composition of $\mathcal{M}$ with an environment with imperfect information is such that whenever two computations of the system differ only in the values of invisible variables along them, the disabling of transitions along them coincide. In [KV97], it has been shown that *CTL* and *CTL** module checking with imperfect information is harder than module checking with perfect information.

In this paper, we consider the extension of the module checking idea to open pushdown systems. These are pushdown systems in which the set of configurations are partitioned (in accordance with the control state and the symbol on the top of the stack) into a set of *system configurations* and a set of *environment configurations.* As in the case of finite-state systems, pushdown module checking is much harder than pushdown model checking for both *CTL*, *CTL**, and μ-calculus. Indeed, it turns out that pushdown module checking is 2EXPTIME-complete for *CTL* [BMP05, BMP] and μ-calculus[FMP07, FMP08], and 3EXPTIME-complete for *CTL** [BMP05, BMP]. For the upper bounds, we exploit the standard automata-theoretic approach. while the lower bound for *CTL* (resp., *CTL**) is shown by a technically non-trivial reduction from the word problem for EXPSPACE–bounded (resp., 2EXPSPACE–bounded) alternating Turing Machines.

We further extend the pushdown module checking problem by considering environments with imperfect information about the system's control state and pushdown store content. Like in the finite-state case, the control states are assignments to Boolean *control variables*, some of which are visible and some of which are not. Similarly, symbols of the pushdown store are assignments to Boolean visible and invisible *pushdown store variables.* In presence of imperfect information, it turns out that *CTL* pushdown module-checking becomes undecidable, and that the undecidability relies upon hiding information about the pushdown store [AMV07]. Indeed, *CTL* pushdown module checking with imperfect state information but visible pushdown store is decidable and 2EXPTIME-complete. The decidability also holds for more expressive logics. Indeed, pushdown module-checking is 2EXPTIME-complete w.r.t the propositional μ-calculus and the graded μ-calculus [KSV02][1] and 3EXPTIME-complete w.r.t *CTL** [ALMS08]. All the above lower bounds follow from the known perfect information case. For the upper bounds, it is possible to use an automata-theoretic approach by reducing the problem to the emptiness problem of a *semi-alternating pushdown tree automaton*. These are alternating pushdown tree automata that behave deterministically on the pushdown store content.

Acknowledgments. The results reported in this extended abstract have been achieved in the last three years in collaboration with several colleagues, which I wish to thank. In particular, I wish to thank Moshe Y. Vardi for having introduced me to the module checking framework.

[1] The graded μ-calculus extends the propositional μ-calculus by allowing graded modalities, which enable statements about the number of successors of a state (see also [BLMV06, BLMV08, BMM08]).

References

[ALMS08] B. Aminof, A. Legay, A. Murano, and O. Serre. μ–calculus pushdown module checking with imperfect information. In *IFIP TCS'08*, volume 273 of *IFIP series*, 2008.

[AMV07] B. Aminof, A. Murano, and M.Y. Vardi. Pushdown module checking with imperfect information. In *CONCUR '07*, LNCS 4703, pages 461–476. Springer-Verlag, 2007.

[BLMV06] P.A. Bonatti, C. Lutz, A. Murano, and M.Y. Vardi. The complexity of enriched μ-calculi. In *ICALP'06*, LNCS 4052, pages 540–551, 2006.

[BLMV08] P.A. Bonatti, C. Lutz, A. Murano, and M.Y. Vardi. The Complexity of Enriched μ-Calculi. *LMCS*, 4(3:11):1–27, 2008.

[BMM08] Alessandro Bianco, Fabio Mogavero, and Aniello Murano. Graded Computation Tree Logic. Technical Report 21, University of Naples "Federico II", Italy, April 2008.

[BMP] Laura Bozzelli, Aniello Murano, and Adriano Peron. Pushdown module checking. *Formal Methods in System Design*, To appear.

[BMP05] Laura Bozzelli, Aniello Murano, and Adriano Peron. Pushdown module checking. In *LPAR'05*, LNCS 3835, pages 504–518. Springer-Verlag, 2005.

[CE81] E.M. Clarke and E.A. Emerson. Design and verification of synchronization skeletons using branching time temporal logic. In *Proceedings of Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *J. of the ACM*, 33(1):151–178, 1986.

[FM07] A. Ferrante and A. Murano. Enriched μ–calculus module checking. In *FOSSACS'07*, volume 4423 of *LNCS*, page 183197, 2007.

[FMP07] A. Ferrante, A. Murano, and M. Parente. Enriched μ–calculus pushdown module checking. In *LPAR'07*, volume 4790 of *LNAI*, pages 438–453, 2007.

[FMP08] A. Ferrante, A. Murano, and M. Parente. Enriched μ-Calculi Module Checking. *LMCS*, 4(3), 2008.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 477–498. Springer-Verlag, 1985.

[Koz83] D. Kozen. Results on the propositional mu–calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KSV02] O. Kupferman, U. Sattler, and M.Y. Vardi. The complexity of the graded μ-calculus. In *CADE'02*, LNAI 2392, pages 423–437, 2002.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *CAV'96*, LNCS 1102, pages 75–86. Springer-Verlag, 1996.

[KV97] O. Kupferman and M. Y. Vardi. Module checking revisited. In *Proc. 9th International Computer Aided Verification Conference*, LNCS 1254, pages 36–47. Springer-Verlag, 1997.

[KVW00] 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.

[KVW01] O. Kupferman, M.Y. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.

[MNP08] Aniello Murano, Margherita Napoli, and Mimmo Parente. Program complexity in hierarchical module checking. In *LPAR*, volume 5330 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 2008.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in Cesar. In *Proceedings of the Fifth International Symposium on Programming*, LNCS 137, pages 337–351. Springer-Verlag, 1981.