Parameterized Model Checking by Network Invariants: the Asynchronous Case

Igor V. Konnov*, konnov@forsyte.tuwien.ac.at
Institute of Information Systems,
Vienna University of Technology, Vienna, 1040, Austria

1 Introduction

Notwithstanding the significant progress of model checking techniques, the interesting problem of checking a specification φ against a parameterized family $\mathcal{F}$ of finite-state models $\{M_n\}$ is still a challenge. Although for certain kind of systems parameterized model checking is of no practical interest, some systems can be scaled up to unboudedly many communicating processes. It is often the case that one checks an instance M_k of $\mathcal{F}$ and then informally reasons that the results hold true for any model M_i of $\mathcal{F}$. This intuition sometimes can be supported formally (cf. [EN95]), giving a rigorous argument.

It is well-known that the parameterized model checking problem is undecidable in general [AK86], even in the case of rings, communication graphs of which can seem to be simple [EN95]. Nevertheless, sometimes the problem can be solved for certain classes of parameterized families, or by providing a sound but incomplete procedure. In order to verify an infinite family $\mathcal{F}$ one has to capture it by a finite description, for instance, by describing a regular structure of inter-process communication.

The framework of network invariants is an example of such approach [WL90, SG90, MG91, CGJ95, CGJ97]; for further references, see [KZ10]. To apply it one describes the family $\mathcal{F}$ in terms of a network grammar G and then tries to detect invariant models among instances of this family. In this paper we extend the framework for the case when processes respect the asynchronous (interleaving) semantics and communicate by synchronous message passing (rendezvous). This work has been reported previously in [ZK07, Kon10a, KZ10, Kon10b].

2 Network Invariants

Here we give a brief description of network grammars and the framework. One describes a few process prototypes $P_1, \ldots, P_k$ as finite state *labelled transition systems* (LTSs) and defines rules of their parallel composition $\|$. For instance, one may use synchronous composition, where all processes have to make a step together at the same time. In this case processes communicate by means of synchronized actions, i.e., a process can take a step labelled with an action a if there is another process taking a step at the same time labelled with a co-action $\overline{a}$. Remaining processes have to make a silent step τ.

The constraints on communication networks of processes are defined in terms of a *context-free network grammar* $G = (\mathcal{T}, \mathcal{N}, \mathcal{P}, \mathcal{S})$. The process prototypes $\mathcal{T} = \{P_1, \ldots, P_k\}$ play role of *terminals*, basic building blocks with action labels yet to be bound, whereas *non-terminals* from $\mathcal{N}$ present larger clusters of processes with all their action labels bound except for few interface actions. The exact way actions of terminals and of non-terminals are bound is defined

*Supported in part by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) grant PROSEED, and by INTAS Project (2006-2009) 05-1000008-8144.

in *production rules* $\mathcal{P}$. Therefore, each non-terminal $N \in \mathcal{N}$ produces a (possibly infinite) language $\mathcal{L}(N)$ of communication networks that generate LTSs by applying the definition of a parallel composition w.r.t. bound actions. We say that such an LTS M belongs to $\mathcal{L}(N)$ in this case. The language $\mathcal{L}(G)$ of the grammar G is defined as $\mathcal{L}(\mathcal{S})$. The instances of a parameterized family $\mathcal{F}$ are generated by $\mathcal{L}(G)$. Network grammars allow one to generate useful parameterized families, where process communication networks form directed acyclic graphs, for instance: lines, rings, and trees.

Suppose, one introduces an equivalence relation or a preorder $\preceq$ on LTSs such that for any formula φ from a class Φ the conditions $M_1 \preceq M_2$ and $M_2 \models \varphi$ imply $M_1 \models \varphi$ (the relation $\prec$ is called *conservative*). Assume even further, that $\preceq$ is *monotonic*: $M_1 \preceq M_2$ and $M_1' \preceq M_2'$ implies $M_1 \parallel M_1' \preceq M_2 \parallel M_2'$. For example, simulation and bisimulation satisfy these requirements for formulas from ACTL* and CTL* respectively.

The inductive structure of systems defined by network grammars gives a natural way to finding invariants of the family $\mathcal{F}$. For every non-terminal $N \in \mathcal{N}$ one has to reveal a LTS $Inv(N)$ such that $M \preceq Inv(N)$ for any $M \in \mathcal{L}(N)$, i.e., every LTS generated from N is simulated by $Inv(N)$, an *invariant* of N. The latter property can be guaranteed by checking if $M \preceq Inv(N)$ for any M obtained by a production rule $N \to N_1 \parallel \cdots \parallel N_m \parallel P_{j(1)} \parallel \cdots \parallel P_{j(k)}$, where each non-terminal N_i is replaced by its invariant $Inv(N_i)$. Moreover, one can try to detect an invariant of $N \in \mathcal{N}$ among small communication graphs derived from N.

While the authors of [WL90] introduced basic network invariants for ring networks and asynchronous parallel composition w.r.t. trace inclusion, in [SG90, MG91, CGJ95, CGJ97] general frameworks for network grammars and synchronous composition are given. Furthermore, in the latter the global properties of systems, sometimes called *multi-index*, are considered, for instance, mutual exclusion. Thus, the invariant detection condition is given with respect to a specification-based abstraction h, i.e., $h(M) \preceq h(Inv(N))$.

3 Asynchronous Case

In our work [ZK07, Kon10a, KZ10] we extend the framework to the case when a system is constructed as a set of processes respecting interleaving semantics and communicating by pairwise synchronized actions. We are interested in checking properties of certain fixed processes in any instance M of the family $\mathcal{L}(G)$, i.e., in checking *single-index* properties.

The crucial point in this setting is that usually strong simulation and strong bisimulation do not help to detect an invariant. The intuition for this is simple: For example, in the synchronous case $P \parallel P \preceq P$ may co-exist with $P \parallel P \parallel P \preceq P$, whereas in the asynchronous setting it is not usually the case. This is due to the step-to-step correspondence in simulation (bisimulation), i.e., for H to be a strong simulation relation ($M \preceq M'$) one requires that $(s_1, t_1) \in H$ iff for any $s_2 : s_1 \xrightarrow{a} s_2$ there is $t_1 : t_1 \xrightarrow{a} t_2$ such that $(s_2, t_2) \in H$; plus, the observed propositions coincide $L(s_1) \cap AP_0 = L(t_1) \cap AP_0$. Under interleaving semantics $P \parallel P \parallel P$ has longer path fragments that can not be matched by shorter path fragments of $P \parallel P$.

To this end we tried to use *block simulation*, a non-symmetric version of block bisimulation introduced in [EN95]. Unlike strong simulation it requires that any finite path $s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_k \xrightarrow{a} s_{k+1}$ is matched by a finite path $t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_m \xrightarrow{a} t_{m+1}$ such that $(s_{k+1}, t_{m+1}) \in H$ and $(s_i, t_j) \in H$ for any $1 \le i \le k$, $1 \le j \le m$. Furthermore, any infinite path $s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_k \xrightarrow{\tau} \cdots$ must be matched by an infinite path $t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_m \xrightarrow{\tau} \cdots$ such that $(s_i, t_j) \in H$ for any $i \ge 1$, $j \ge 1$. Block simulation appeared to be conservative w.r.t. ACTL*-X. However, it is not monotonic and thus it can not be used in the framework of network invariants.

In order to apply block simulation in the framework we generalized it to *quasi-block simulation*, that is both monotonic and conservative w.r.t. ACTL*-X. In quasi-block simulation matching paths can be split further into equal number of partitions that should relate to each other (see [KZ10] for the exact definition). This allows one to construct new quasi-block simulation when additional processes are added and thus monotonicity is preserved. Quasi-block simulation seems to be hard for computation. Thus, it is used to prove the soundness of the framework while block simulation could be still applied for invariant detection. In particular, from $M_1 \preceq^b M_2$ if follows that $M_1 \parallel P \preceq^{qb} M_2 \parallel P$.

Finally, we found another kind of simulation that co-exists with block simulation, i.e. $M_1 \preceq^b M_2$ iff $M_1 \preceq^{sb} M_2$. As it closely resembles relation between van Glabbeek's semi-branching bisimulation and branching simulation, we call it *semi-block simulation*. To ensure that H is semi-block simulation one has to ensure that any finite path $s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_k \xrightarrow{a} s_{k+1}$ is matched by a finite path $t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_m \xrightarrow{a} t_{m+1}$ such that $(s_{k+1}, t_{m+1}) \in H$ and $(s_1, t_n) \in H$ if $n > 1$. Furthermore, any infinite path $s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_k \xrightarrow{\tau} \cdots$ must be matched by an infinite path $t_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} t_m \xrightarrow{\tau} \cdots$ such that $(s_1, t_\ell) \in H$ for some $\ell \geq 1$. Thus, one can check much simpler conditions on paths when constructing simulation.

4 Implementation

The framework is implemented as a toolset CheAPS [Kon10b]. Each model M_n from a family $\mathcal{F}$ is composed of a fixed number of control processes and of n processes from a fixed set of prototypes. Given a network grammar G and finite-state process prototypes CheAPS generates finite-state models M_n of n processes and checks if one of such models can be used as an invariant of the family. As soon as invariants of all non-terminals are detected, $Inv(\mathcal{S})$ (and probably a few smaller LTSs) is checked by Spin against a specification φ. If Spin completes the verification successfully, then all the models of $\mathcal{F}$ satisfy φ.

CheAPS is designed to use existing non-parameterized models as a source of the parameterized family description. When one has a debugged model of a fixed number of processes, it should be considerably easy to create a parameterized version of it. Therefore, we chose the following approach. The process prototypes are described in a subset of Promela. The communication structure of the models from $\mathcal{F}$ is described by means of a network grammar G. The terminals of G stand for process prototypes whereas non-terminals of G are used to generate subnets. The rules of this grammar are annotated with channel bindings to provide a correct connection of prototype processes to the network. CheAPS includes the `gen-net-model` tool to automatically generate Promela code of models M_n from a network grammar G and prototype descriptions.

The core component of CheAPS is the `simba` tool intended for checking block simulation between finite-state models. For each non-terminal N of the grammar G, the tools sequentially generate LTSs induced by N. For two LTSs induced by N `simba` constructs an initial semi-block simulation. In the simple case if a larger model is proved to be simulated by a smaller one, then the smaller one is declared to be an invariant $Inv(N)$ of N. In a general case several models induced by N should be simulated by an invariant model $Inv(N)$. The models vary by application of different grammar rules to N in the last steps. The goal is to find a model that simulates all the models derived from N.

As state-spaces in model checking grow rapidly with increase of the number of communicating processes `simba` has several state storage implementations and search strategies. State storage implementations are as follows: `std`, `dfa`, `dfafile`. The first one is a standard `C++`

implementation of a set, which works well only on relatively small state spaces. The second one uses the representation of state set by a minimized DFA, which is implemented in Spin. The last one is a mixed representation by a minimized DFA and a sequential file. While DFA is utilized to check set membership, a file keeps "unstable" states, which should be explored on the next iteration. Thus, `dfafile` keeps the balance between memory consumption and performance. Along with forward search strategy `simba` provides forward-then-back search strategy, which propagates negative results.

If `simba` cannot find an invariant for "reasonably" large LTSs induced from N one may apply the `failpath` tool. This tool selects the paths of the LTSs to give an insight on the difference in their behaviour. This tool may be helpful in understanding why such an invariant can not be found easily, though it is not guaranteed to find a good representative counter-example.

The tool has been successfully applied to several examples: Chandy-Lamport snapshot algorithm (line topology), Awerbuch distributed depth-first search algorithm (tree topology), and the model of RSVP protocol (ring topology), where invariants were detected successfully by our tools. The project homepage is `http://lvk.cs.msu.su/~konnov/cheaps/`.

References

[AK86] K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 15:307–309, 1986.

[CGJ95] E. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In Insup Lee and Scott Smolka, editors, *CONCUR '95: Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 395–407. Springer Berlin / Heidelberg, 1995.

[CGJ97] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks. *ACM Trans. Program. Lang. Syst.*, 19(5):726–750, September 1997.

[EN95] E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '95, pages 85–94, New York, NY, USA, 1995. ACM.

[Kon10a] I. V. Konnov. On application of weaker simulations to parameterized model checking by network invariants technique. *Automatic Control and Computer Sciences*, 44(7):378–386, 2010.

[Kon10b] Igor V. Konnov. CheAPS: a checker of asynchronous parameterized systems. In *Third International Workshop on Invariant Generation (WING 2010)*, 2010.

[KZ10] Igor V. Konnov and Vladimir A. Zakharov. An invariant-based approach to the verification of asynchronous parameterized networks, 2010.

[MG91] R. Marelly and O. Grumberg. Gormel — grammar oriented model checker. Technical Report 697, The Technion, Haifa, Israel, 1991.

[SG90] Ze'ev Shtadler and Orna Grumberg. Network grammars, communication behaviors and automatic verification. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 151–165. Springer Berlin / Heidelberg, 1990.

[WL90] Pierre Wolper and Vinciane Lovinfosse. Verifying properties of large sets of processes with network invariants. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 68–80. Springer Berlin / Heidelberg, 1990.

[ZK07] Vladimir Zakharov and Igor Konnov. An invariant-based approach to the verification of asynchronous parameterized networks. In *International Workshop on Invariant Generation (WING'07)*, pages 41–55, 2007.