CheAPS: a Checker of Asynchronous Parameterized Systems

Igor V. Konnov
`konnov@cs.msu.su`

Lomonosov Moscow State University

We present CheAPS, the checker of asynchronous parameterized communicating systems. It is a set of tools for verification of parameterized families $\mathcal{F} = \{M_n\}$ of finite-state models against LTL specification φ. Each model M_n from a family $\mathcal{F}$ is composed of a fixed number of control processes and n processes from a fixed set of prototypes. Given a description of a family $\mathcal{F}$ CheAPS generates finite-state models M_n and checks if one of such models can be used as an invariant of the family. As soon as an invariant is detected it is model checked by Spin to verify it 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 parameterized family description. When one has a debugged model with a fixed number of processes it should be rather easy to create a parameterized variant. Therefore, we chose the following way. 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. A parameterized family $\mathcal{F}$ as a set of finite-state models can be viewed as a language of the network grammar G. CheAPS includes the `gen-net-model` tool to automatically generate Promela descriptions 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 models induced by N are successively generated. For two models induced by N `simba` constructs a block simulation relation. 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 I_N of N. In a general case several models induced by N should be simulated by an invariant model I_N. The models vary by application of different grammar rules to N in the last steps. The goal is to find such a model which 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 models induced from N one may apply the `failpath` tool. This tool selects the paths in the models to give an insight on the difference in the behaviour of the models. This tool may be helpful in understanding why such an invariant can not be found easily.

We are going to demonstrate an application of CheAPS to several examples: Chandy-Lamport snapshot algorithm, Awerbuch distributed depth-first search algorithm, Milner's scheduler, and the model of RSVP protocol, where invariants were detected successfully on that models by our tools. The project homepage is `http://lvk.cs.msu.su/~konnov/cheaps/`. It is available under BSD-like license.