Finding the Right Order in Compositional Aggregation

Background

The theory of communicating processes [4, 6] allows us to implicitly specify how hierarchical systems behave, by defining how their components behave. The explicit model of the behavior of a system can be found by combining the behaviors of its components. The size of the resulting model, however, grows exponentially with the number of components. This problem is know as the state space explosion problem.

Large models can often be replaced by smaller, equivalent, models. Of course it is then necessary to define what we mean by "equivalent". Usually we differentiate between observable and unobservable behavior and use as our definition of equivalence the concept of *observational equivalence* [6]. In general we call two models equivalent based only on how we observe their behaviors. We do not care about differences in internal behvaior. However, it is often expensive to compute the minimized (or aggregated) equivalent of a model, so it is imperative to avoid the construction of these large models altogether. If a system is defined as the composition of some component models we can use the technique of *compositional aggregation*. In compositional aggregation we alternate between combining models and minimizing them to find the minimized model of the entire system in an iterative way. But for purely communicating systems this technique often fails as is described in [5].

In [2] Hermanns introduces interactive Markov chains (IMCs), which combine communicating processes with stochastic behavior. This formalism can be used to find the *stochastic* behavior of systems, i.e. the resulting model also contains information about timing. It turns out that for stochastic models compositional aggregation can be very successful in avoiding large state spaces [3, 1]. However, the effectiveness of compositional aggregation is determined largely by the order in which the components of the system are combined. For instance, in a three-component system we can ask the question: should we first combine components A and B and then add component C or should we instead start by combining B and C? So far, the question of how we should order the compositions has usually been answered by the researchers themselves. In [1] for instance, the authors used intuitive heuristics and trial-and-error to find the best composition orders. In order for the compositional aggregation method to be fully automatic the composition order must be found in a mechanical way. In [7] various formal heuristics are proposed to find good composition orders for communicating finite state machines (CSFM).

The Assignment

The assignment is to define formal composition-order heuristics for input/output interactive Markov chains (I/O-IMCs, a variation on IMCs) based on the heuristics proposed by Tai and Koppol for CSFMs [7] and implement them in the tool

chain of Boudali, Crouzen and Stoelinga [1]. The programming language used in this tool is C. The heuristics must then be applied to a number of case studies and compared to the results of intuition-based heuristics. Finally, the student will draw conclusions from this comparison and will, possibly, suggest ideas for other composition-order heuristics.

Prerequisites

Required Experience in programming with C or C++.

Optional Some knowledge of automata theory.

Optional Some knowledge of interactive processes (labelled transition systems).

Optional Some knowledge of stochastic modelling (Markov chains)

References

[1] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *International Conference on Dependable Systems and Networks*, 2007.

[2] H. Hermanns. *Interactive Markov Chains*. Springer, 2002.

[3] H. Hermanns and J.-P. Katoen. Automated compositional markov chain generation for a plain-old telephone system. *Science of Computer Programming*, 36:97–127, 2000.

[4] C.A.R. Hoare. *Communicating Sequential Processes*. 1985.

[5] K.G. Larsen and R. Milner. Verifying a protocol using relativized bisimulation. In *Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, 1987.

[6] R. Milner. *A Calculus of Communicating Systems*. 1989.

[7] K.-C. Tai and P.V. Koppol. An incremental approach to reachability analysis of distributed programs. In *International Workshop on Software Specifications & Design*. IEEE, 1993.