On the Minimization of Hybrid Automata

Erika Ábrahám[1] Ulrich Loup[1] Ralf Wimmer[2]
Joost-Pieter Katoen[1]

[1]RWTH Aachen University, Germany
{abraham,loup,katoen}@informatik.rwth-aachen.de

[2]University of Freiburg, Germany
wimmer@informatik.uni-freiburg.de

Abstract

Minimization is a widely used technique for the abstraction of systems with large or infinite state spaces. The result of the minimization is a finite partitioning of the concrete state space and an induced abstract transition system with the partitions as abstract states. This final abstract transition system is bisimilar to the concrete system with respect to the reachable states and the observable actions.

In this work we discuss the minimization of hybrid systems and introduce a modified minimization algorithm. It computes an abstraction that is in general not bisimilar to the concrete system, but it provides enough information for reachability analysis. Furthermore, besides the well-known forward approach for minimization, we discuss as an alternative a backward minimization method.

1 Introduction

The *analysis* of different kinds of systems—discrete, timed, or hybrid systems—builds an active research area in computer science. Finite-state systems can be model checked. However, if the state space is too large, model checking, suffering from the state space explosion problem, may be not applicable.

For the analysis of systems with large or even infinite state spaces *minimization* can be used to generate an abstraction of the concrete system. The minimization gives us a partitioning of the state space into a finite number of *regions*. The partitioning induces a finite *abstract transition system* with the regions as abstract states. The concrete and the abstract system are *bisimulation-equivalent*, and the abstraction can be model checked instead of the concrete system.

Minimization is frequently used for discrete systems, for which it is complete, i. e., it always terminates. A minimization algorithm for *linear hybrid automata* was introduced in [1] for model checking reachability properties. As the problem is undecidable, the algorithm is not complete. However, if the algorithm terminates, it gives us a finite abstraction that is bisimulation-equivalent to the concrete system and that can be used to check the reachability problem.

In this work we discuss different improvements for the above method. Though we are mainly interested in hybrid automata reachability analysis, our improvements can also be applied to the minimization of discrete and timed systems.

2 Minimization for Linear Hybrid Automata

Due to lack of space we do not formally introduce *linear hybrid automata*. Important in our context is that the state space of a hybrid automaton has some discrete and some continuous dimensions. *Discrete steps* $\xrightarrow{e}$ may change the values of both discrete and continuous variables according to a discrete transition e, whereas *time steps* $\xrightarrow{\delta}$ only affect the continuous ones. We denote by $\sigma \rightarrow \sigma'$ the fact that σ' can be reached from σ via a (possibly stutter) discrete and a subsequent (possibly zero-length) time step.

Given a set R, we use $\langle R\rangle^{\nearrow}$ to denote the set of states that are reachable from R via some time step, and $\langle R\rangle^{\swarrow}$ to denote the region containing all states from which a state in R is reachable via some time step. Analogously, $post[R]$ denotes the set of those states that are reachable from R via a discrete step, and $pre[R]$ the set of states from which a state in R is reachable via a discrete step. For linear sets R the above sets are constructible.

The basic structure of the minimization algorithm for linear hybrid automata from [1] is as follows. It defines a stepwise refined sequence of state space partitionings. Each partitioning induces a finite state transition system, with the partitions being the abstract states called regions. Regions containing at least one concrete initial state are initial regions. There is an abstract transition $R \to R'$ between the regions R and R' iff a there are $\sigma \in R$ and $\sigma' \in R'$ with $\sigma \to \sigma'$, i. e., if $pre[\langle R'\rangle^{\swarrow}] \cap R \neq \emptyset$.

Assume a linear hybrid automaton and a property whose invariance we want to check. The initial partitioning defines two regions, one containing all "good" states satisfying the property and a second one containing the "bad" states violating it. This initial abstraction gets refined stepwise until it becomes bisimilar to the concrete system as follows: Regions R with $R \to R'$ for some R' such that not all states in R have a successor state in R' are called *non-stable*. They get split into two parts $R_1 = pre[\langle R'\rangle^{\swarrow}] \cap R$ and $R_2 = R \backslash R_1$. The region R_1 contains those states of R from which R' is reachable and R_2 the rest. The splitting is restricted to (in the abstraction) reachable regions in a forward manner: First initial regions are marked as reachable. Only regions that are marked as reachable get split. If they become stable, their successor regions also get marked as reachable.

If there are no such regions, the algorithm terminates. The concrete system is safe iff no "bad" region with states violating the property is reachable in the abstraction.

The correctness of this minimization algorithm is easy to show: On the one hand, the abstraction over-approximates the concrete system's behavior, i. e., if no "bad" region is reachable in the abstraction, then the concrete system is correct. On the other hand, if a "bad" region is reachable, then there is a path $R_0 \to \ldots \to R_n$ from an initial region R_0 to a "bad" region R_n. Each region on the path contains at least one concrete reachable state by induction: The initial region R_0 contains at least one initial state. For each abstract step $R_i \to R_{i+1}$ if R_i contains at least one concrete reachable state σ_i, then, since *all* states in R_i have a successor in R_{i+1}, we can reach a state in R_{i+1} from σ_i.

3 Improved Minimization

We only describe the two main improvements. Though these contributions can be also used for the analysis of discrete systems, we focus on the application to hybrid systems and restrict ourselves to reachability problems.

3.1 A Coarser Minimization

We define a minimization that generates a finite abstraction that is *not bisimulation-equivalent* to the concrete system, but which still assures that each reachable region contains at least one concrete reachable state, such that the abstraction can be used to model check the safety property. An advantage of this modified method is that in general it defines a smaller abstraction, and thus minimization and model checking are both faster. Furthermore, the abstraction is coarser than the one defined by the original method, increasing the chance for termination.

In the original method the minimization algorithm splits regions that are marked as reachable until they become stable, in which case also their successors get marked as reachable. However, if there is a reachable region R with a successor R' such that R is *stable for* R', i. e., if for all states $\sigma \in R$ there is a $\sigma' \in R'$ such that $\sigma \to \sigma'$, then we can already conclude that R' contains at least one reachable concrete state and can mark it as reachable. Since we are only interested in reachability, we need to split non-stable regions R only for successor regions for which R is non-stable *and* which are not yet marked as reachable. The stability of R is only needed to make a conclusion about the reachability of its successor regions. If a successors R' of R is already marked as reachable, then we do not need the stability of R to make conclusions about the reachability of R'.

We make a depth-first search for reachable regions instead of a breadth-first search: When a region R gets marked as reachable, we can also check for successors R' of R if R is stable for R'. If it is the case, we can also mark such R' as reachable. This allows to reduce the number of splittings in the minimization.

3.2 Backward Minimization

All minimization algorithms we are aware of use *forward reachability* analysis for the abstraction refinement. However, backward reachability analysis is sometimes more efficient than the forward analysis. Therefore, dually to forward minimization, we define a *backward minimization method*.

In the backward minimization we do not split regions according to the reachability of successors but according to predecessors. We mark regions that are predecessors of "bad" regions as backward reachable. Given a region R marked as backward reachable and a predecessor region R' of R such that not all states of R can be reached from R', we split R into two halfs, one that is reachable from R' and one that is not. Formally, we split R into $R_1 = \langle post[R'] \rangle^{\nearrow} \cap R$ and $R_2 = R \backslash R_1$. If there is no such predecessor region of R then we mark all predecessors of R as backward reachable.

The algorithm terminates if no region that is marked as backward reachable can be split. If an initial region is marked as backward reachable then the system does not satisfy the property, otherwise it does so.

In the backward method, for the splitting we need to compute time successors, in contrast to the forward method that computes the predecessors. This is more advantageous for more complex hybrid automata for which the time predecessor is not always easy to determine.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.