SAT-based Unbounded Model Checking of Timed Automata
(Extended Abstract)

Wojciech Penczek       Maciej Szreter
Institute of Computer Science, PAS
Ordona 21, 01-237 Warsaw, Poland
{penczek, mszreter}@ipipan.waw.pl

Introduction. Symbolic model checking, based mostly on BDD graphs, is a standard technology nowadays. It has been implemented in several tools like Uppaal, Kronos, RED, or Cadence FORMALCHECK. In the last decade, very efficient implementations of SAT solvers have been provided. Thanks to that SAT-based Bounded Model Checking (BMC) and Unbounded Model Checking (UMC) became feasible. The idea of UMC [4] consists in encoding the states of a model, where a temporal formula holds, by propositional formulas in conjunctive normal form (called blocking clauses). Unfortunately, the number of these clauses can be exponential. There are several methods aiming at improving the above deficiency by using generalized blocking clauses [6], or for instance circuit cofactoring [3]. In this paper we define and use timed generalized blocking clauses in UMC of timed automata for untimed temporal properties expressed in $\text{CTL}_X$.

Timed automata and Unbounded Model Checking. The standard formalism of networks of timed automata (TA) is used for specifying systems. Time flow in TA is modeled by real time variables, called clocks. A timed automaton is a transition system where locations (associated with invariants) are connected by transitions equipped with guards, resets of clocks, and action labels. Clock constraints are conjunctions of arithmetic expressions in which a clock (or a difference of clocks) is related to a constant. Clock constraints are used for representing guards and invariants.

Our approach to UMC for TA consists of the following steps. The detailed regions are states in our finite abstraction of the concrete state space of TA. The transition relation is defined as a composition of the future projection with action transitions. The temporal logic $\text{CTL}_X$ is interpreted in the above model. The clock valuations are discretized and the whole model is encoded in Propositional Logic in the standard way, like in BMC [5]. The fixpoints characterizing the temporal operators $\text{AG}$ and $\text{AU}$ are computed. This is based on the quantifier elimination and pre-image computation. Optimizing this step is crucial for the overall efficiency and it is the main contribution of this paper.

Efficient SAT-based quantifier elimination. Binary Decision Diagrams (BDDs) are typically used for representing formulas in model checking. Despite numerous advantages, the exponential size of these diagrams for some formulas, and the high cost of computing the quantifier elimination calls for searching for new methods.

An alternative approach, proposed in [4], consists in using a syntactic formula representation and applying a SAT-solver to eliminating quantifiers. The idea is based on the concept of blocking clauses: the input formula is translated to an equisatisfiable formula in Conjunctive Normal Form. The quantified variables are removed from all the blocking clauses. This method was improved in [6] by introducing generalized blocking clauses, consisting not only of propositional variables, but also of literals encoding more general subformulas. This approach can reduce the number of blocking clauses in some cases. Another contribution of [6] consists in providing a new method for generating blocking clauses. Instead of exploring internal structures of the SAT-solver (as proposed in [4]), the input formula is searched with respect to the current blocking assignment in order to find subformulas implying its false value. Two sets consisting of the formulas over quantified and not quantified variables are identified. The negated elements of latter define blocking clauses. The general optimized method of the quantifier elimination was applied [6] to verification of untimed systems. For every blocking clause, the quantification is restricted to the components in which the blocked action occurs (this is explained below). However, for timed systems another optimization is needed in order to deal with discretized clock valuations. The next section gives more details in application to timed systems.
Efficient SAT-based quantifier elimination applied to timed systems. The optimized method described above can be applied directly to an abstract discretized model. Unfortunately, the experiments have shown that this is not effective. The main reason is that there is no way of representing concisely the time constraints in the blocking clauses. This means that each blocking clause blocks usually exactly one detailed region and thus only one transition. The key idea of timed generalized blocking clauses is as follows. First, the transition relation of the discretized abstract model is encoded. The propositional variables and the Boolean operators of the input formula are handled in the straightforward way. So let’s focus on the pre-image computation of the formula $AG \varphi$. The formula $\neg \varphi$ is encoded. Then, a SAT solver finds assignments corresponding to transitions leading to the states in which $\neg \varphi$ holds. The source state of each such transition needs to be blocked. This is done by a blocking clause. A generalized blocking clause is to block not only the current blocking transition, but also some other transitions which would be potentially blocking, but will not have to be enumerated by the solver explicitly. This way the number of blocking clauses can be significantly reduced.

A timed generalized blocking clause consists of the location part (encoding a set of locations) and the timed part (encoding a set of discretized clock valuations). The general idea of generating the clauses by the formula search consists in the following steps. First, a blocking action is identified. For the location part, after finding a blocking assignment the quantification range is restricted to the variables encoding the components in which the blocking action occurs. Only the restricted class of automata is considered, i.e., these without invariants including the operator $<$ or $\leq$. This is because of efficiency reasons - the search may find some components irrelevant for the current transition and if the time can flow there, they can be abstracted away.

For the timed part, respective clock constraints are identified, and then the time pre-image with respect to the blocked action is computed by using the conventional (not logic-based) technique of Difference Bound Matrices (DBMs), which provides an efficient representation of clock constraints with the necessary operations. The obtained DBM is encoded as a Boolean formula and added to the resulting blocking clause.

Experimental results In order to verify the effectiveness of the presented algorithm, a standard benchmark of the Fischer’s mutual exclusion has been examined for $n$ processes (see Fig. 1 for $n = 2$). The mutex property is satisfied when $\delta > \Delta$. In Table 1 our results are compared with these obtained from the tools: Uppaal [1] and RED [7]. Our approach is particularly effective for the case when the mutex property holds.

### Table 1. Tests for $EF(\bigvee_{1 \leq i,j \leq n, i \neq j} crit_i \land crit_j)$

<table>
<thead>
<tr>
<th>Mutex parameters</th>
<th>Time [s]</th>
<th>Upaal</th>
<th>RED</th>
<th>Verics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 12, \Delta = 1, \delta = 2$ (true)</td>
<td>580</td>
<td>304</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>$n = 13, \Delta = 1, \delta = 2$ (true)</td>
<td>-</td>
<td>657</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>$n = 20, \Delta = 1, \delta = 2$ (true)</td>
<td>-</td>
<td>-</td>
<td>491</td>
<td></td>
</tr>
<tr>
<td>$n = 11, \Delta = 3, \delta = 4$ (true)</td>
<td>125</td>
<td>133</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>$n = 10, \Delta = 2, \delta = 1$ (false)</td>
<td>7</td>
<td>49</td>
<td>97</td>
<td></td>
</tr>
</tbody>
</table>

Conclusions A new UMC method for $CTL_\infty$ properties of TA has been presented. The method exploits generalized blocking clauses and efficient SAT-based quantifier elimination. The implementation is a part of the tool Verics [2]. Experimental results look very promising.

References


