2010 11th International Workshop on Microprocessor Test and Verification

Bounded Model Checking of Incomplete Networks of Timed Automata

Christian Miller, Karina Gitina, Christoph Scholl, and Bernd Becker
Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg, Germany
{millerc,gitina,scholl,becker}@informatik.uni-freiburg.de

***Abstract*—Verification of real-time systems – e.g. communication protocols or embedded controllers – is an important task. One method to detect errors is called bounded model checking (BMC). In BMC the system is iteratively unfolded and then transformed into a satisfiability problem. If an appropriate solver finds the *k*-th instance to be satisfiable a counterexample for a given safety property has been found. In this paper we present a first approach to apply BMC to networks of timed automata (that is a system of several interacting subautomata) where parts of the network are unspecified (so called blackboxes). Here, we would like to answer the question of unrealizability, that is, is there a path of a certain length violating a safety property regardless of the implementation of the blackboxes. We provide solutions to this problem for two timed automata communication models. For the simple synchronization model, a BMC approach based on fixed transitions is introduced resulting in a SAT-Modulo-Theory formula. With respect to the use of bounded integer variables for communication, we prove unrealizability by introducing universal quantification, yielding more advanced quantified SAT-Modulo-Theory formulas.**

I. Introduction

Real-time systems appear in many areas of life, such as time critical communication protocols, or in embedded controllers for automobiles. As these systems grow in complexity, verifying their correctness becomes harder, but increasingly more important. One method to prove the presence of errors in complex systems is bounded model checking (BMC) [1], [2]. BMC accomplishes this by iteratively unfolding the system k times until a predefined maximum unfolding depth is reached. After adding the negated property, the BMC instance is converted into a satisfiability problem and then solved by an appropriate solver. If the solver finds the k-th instance satisfiable, a path of length k violating the property has been found. Whereas BMC for conventional discrete circuits result in SAT-problems, BMC instances for real-time systems are encoded into so-called SAT-Modulo-Theory (SMT) formulas, since they are augmented with continuous time constraints over real-valued variables. BMC for timed automata has been studied and improved by several groups [3]–[6]. All this previous work has made the assumption that the entire design is specified.

In contrast, this work focuses on BMC with *incomplete information*, that is designs where parts of the system are not known (so-called blackboxes). For discrete circuits this has been examined extensively in [7]–[9], where the encoding of the BMC instances can yield a SAT or QBF formula, depending on the abstraction level.

In this paper, we describe our approach for analyzing incomplete designs in the continuous domain. We want to answer the question of unrealizability: for a given safety property (w.l.o.g. reachability of locations of the system) is there a counterexample which exists regardless of the implementation of the blackboxes? If so we call this property unrealizable for the network of timed automata. For interaction between the subautomata of the network we focus on two prevalent models, namely *synchronization* and *bounded integer communication*. Depending on which communication model is used, the unknown behavior of the blackboxes has to be modeled in different ways. First experimental results show the efficiency of our methods.

The paper is structured as follows: In Section 2 we present some preliminary information, and our model of timed automata is introduced. Then networks of timed automata as well as both the *synchronization* and *bounded integer communication* interaction models are introduced. Section 3 defines the BMC encoding for a network of timed automata and then extends this encoding for each communication model. Then in Section 4 we introduce blackboxes into networks of timed automata and show our approach for encoding and solving these blackbox BMC problems. After presenting preliminary experimental results in Section 5 we conclude the paper in Section 6.

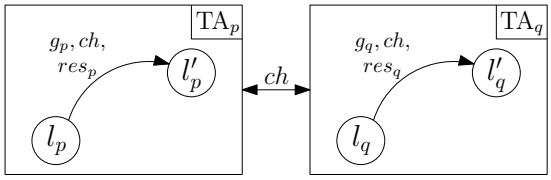
II. Preliminaries

In this paper we focus on timed automata based on the model in [10]. Timed automata are an extension of conventional automata by real-valued clock variables. For these clocks, time is continuous, and transitions and invariants on locations can be labled with constraints over clocks. The set $CC(X)$ of clock constraints over a set of clock variables X is a conjunction of comparisons of clocks to constants and is defined inductively by $g := \mathit{true} \mid x \sim c \mid g_1 \wedge g_2$ with $c \in \mathbb{N}$, $x \in X$, and $\sim \in \{<, \leq, >, \geq\}$. A timed automaton is defined as a tuple $\mathrm{TA} = \langle L, l_0, X, Inv, E \rangle$ with:

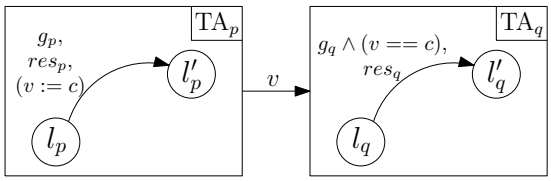
- a set of locations L,
- an initial location $l_0 \in L$,
- a set of real-valued clock variables X,
- a function $Inv : L \rightarrow CC(X)$, which assigns a clock constraint to each location and
- a set of edges $E \subseteq L \times CC(X) \times 2^X \times L$.

This work has been partially funded by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, www.avacs.org)

978-0-7695-4354-3/10 $26.00 © 2010 IEEE
DOI 10.1109/MTV.2010.19



(a) Synchronization



(b) Bounded Integer Communication

Fig. 1. Communication Models

A transition $e = \langle l, g, res, l' \rangle \in E$ can be taken if the guard $g \in CC(X)$ is satisfied. If $res \subseteq 2^X$ is not empty then all clock variables in res are reset to 0. Additionally the current assignment (after resetting clocks in res) to the clock variables must not violate the invariant of the successor location l'. A state $s = \langle l, \nu \rangle$ of a timed automaton is given by the current location l and the current assignment ν to the clock variables. Note that taking a transition needs no time. A timed automaton can stay in one location for an arbitrary amount of time unless the invariant of the location is violated. To prevent the invariant from being violated the automaton is forced to take a transition leading out of this location. If several transitions are enabled at the same time, the automaton can choose non-deterministically which one to take.

Usually, the systems we are considering are networks of several timed automata $\mathcal{N} := \{\text{TA}_1, \ldots, \text{TA}_n\}$ which run in parallel and can interact in multiple ways. Note that in networks of timed automata we allow *global* clocks, that is all clock variables can be read and reset by any subautomaton. So, instead of defining X_j for each subautomaton TA_j, one set $X_\mathcal{N}$ for all clock variables is introduced.

A. Synchronization

In the synchronization model we extend the network $\mathcal{N}$ of timed automata by a set of so-called channels $\text{Ch}_\mathcal{N}$. In this extended model, edges in several subautomata can be labeled together in one channel $ch \in \text{Ch}_\mathcal{N}$ if they are to be executed simultaneously. This means that an edge within a channel can only be taken if it happens synchronously with all other edges labeled with the same channel in other subautomata. Of course all guards as well as all invariants of the successor states have to be satisfied for this to happen. In UPPAAL [11] a similar synchronization model is used, where exactly two automata synchronize on one channel (so-called binary channels). Figure 1(a) illustrates a synchronization of two subautomata TA_p and TA_q via the channel ch. If these subautomata are in the locations l_p and l_q, respectively, and both guards g_p and g_q are satisfied, then either both transitions can be taken simultaneously, or none of them is taken.

B. Bounded Integer Communication

In many real-world designs bounded integer variables are used for communication of timed automata. These variables can be set by arithmetic expressions as an action of a transition, and can be compared to constants as a guard of a transition. They can be read and written to by any timed automaton of the network, and will keep their value until they are re-assigned. Bounded integer variables have a minimum and a maximum value that they can be assigned. Furthermore, they must be initialized to a certain value (default is the lower bound).

Let $V_\mathcal{N} := \{v_1, \ldots, v_m\}$ be the set of global integer variables for the network of timed automata $\mathcal{N}$, with each v_k, $1 \leq k \leq m$ having a lower bound $v_{\text{k,min}}$ and an upper bound $v_{\text{k,max}}$ and an initial value $v_{\text{k,init}}$. For the sake of convenience in this work we limit assignments and comparisons of bounded integer variables to constants. That is, assignments to bounded integer variables resulting from the actions of a transistion are written as $v_k := c$. Comparisons, which guard the transitions are defined like $v_k == c$ for a constant $c \in \mathbb{N}$ with $v_{\text{k,min}} \leq c \leq v_{\text{k,max}}$. Figure 1(b) shows an example for a bounded integer variable communication. First the transition in automaton TA_p has to be taken and thus, the variable v is assigned to a constant c. Then, an arbitrary amount of time may pass until the transition on automaton TA_q is taken.

Using these definitions, we will now describe how we encode our BMC problems.

III. Encoding of the Bounded Model Checking Problem

The BMC procedure iteratively searches for a counterexample, starting at length 0, and up to a predefined length k. Using the initial state I^0 (in the following an upper index always denotes the unfolding depth), the system is unfolded by repeated conjunction of the transition relation $T^{i,i+1}$ and the negated property $\neg P^k$. This formula $I^0 \wedge T^{0,1} \wedge \ldots \wedge T^{k-1,k} \wedge \neg P^k$ is a satisfiability problem, which is satisfied if there is a path of length k leading to a state violating the property.

We encode the BMC problem for a network of timed automata $\mathcal{N}$ according to an interleaving semantic similar to [10]. Additionally, we allow parallel transitions unless writing conflicts occur. Preserving the conventional interleaving semantic can be achieved by a slight modificiation of the formulas described in this section. In the following the encoding of the initial state I^0, the transition relation $T^{i,i+1}$ and a safety property P^k are described. For the locations of each subautomaton TA_p of the network, we introduce new boolean variables $\vec{at_p}$ which represent its binary encoding (in the following we abbreviate this binary encoding by simply writing $at_q = l$, $l \in L_q$).

In the initial state of the network all subautomata are in their initial location and all clocks are set to zero:

$$I^0 \quad := \quad \bigwedge_{j=1}^{n} (at_j^0 = l_{j,0}) \wedge \bigwedge_{x \in X_\mathcal{N}} (x^0 = 0)$$

For the transition relation we have to differentiate between a discrete step $T^{i,i+1}_{\text{jump}}$, which describes all possibitities of

changing a location, and a continuous step $T^{i,i+1}_{\text{flow}}$ describing the passing of time.

We introduce boolean variables $r_{x,j}$ which are *true* if clock x is reset in subautomaton TA_j, and *false* otherwise. In this way we can later define a *noreset*-formula which ensures that un-reset clocks keep their values. So, for each edge $e = \langle l, g, res, l' \rangle$ in a subautomaton TA_j we encode the transition as follows:

$$\begin{aligned} T^{i,i+1}_{\text{jump}}(e,j) \quad := \quad & (at^i_j = l) \wedge g^i \wedge (at^{i+1}_j = l') \wedge Inv^{i+1}(l') \\ & \wedge \bigwedge_{x \in res} \big(r^i_{x,j} \wedge (x^{i+1} = 0)\big) \\ & \wedge \bigwedge_{x \in X_{\mathcal{N}} \setminus res} \neg r^i_{x,j} \end{aligned}$$

We also have to consider the case that one subautomaton does not perform a transition, that is, it stays in its current location and no clocks are reset:

$$\text{fix}^{i,i+1}(j) \quad := \quad (at^{i+1}_j = at^i_j) \wedge \bigwedge_{x \in X_{\mathcal{N}}} \neg r^i_{x,j}$$

For each clock $x \in X_{\mathcal{N}}$ it has to be checked whether x has been reset in any subautomaton. If not, the clock keeps its value:

$$\text{noreset}^{i,i+1} \quad := \quad \bigwedge_{x \in X_{\mathcal{N}}} \Big(\big(\bigwedge_{j=1}^{n} \neg r^i_{x,j} \big) \implies (x^{i+1} = x^i) \Big)$$

The discrete step $T^{i,i+1}_{\text{jump}}$ of the whole network finally is encoded as follows:

$$\begin{aligned} T^{i,i+1}_{\text{jump}} \quad := \quad & \bigwedge_{j=1}^{n} \Big(\bigvee_{e \in E_j} T^{i,i+1}_{\text{jump}}(e,j) \vee \text{fix}^{i,i+1}(j) \Big) \\ & \wedge \text{noreset}^{i,i+1} \end{aligned}$$

A continuous step of the network simply states that all subautomata stay in their locations and time passes equally for all clocks while no invariants are violated:

$$\begin{aligned} T^{i,i+1}_{\text{flow}} \quad := \quad & \bigwedge_{j=1}^{n} \Big((at^{i+1}_j = at^i_j) \\ & \qquad \wedge \bigwedge_{l \in L_j} \big((at^{i+1}_j = l) \implies Inv^{i+1}(l) \big) \Big) \\ & \wedge (\lambda^i > 0) \wedge \bigwedge_{x \in X_{\mathcal{N}}} (x^{i+1} = x^i + \lambda^i) \end{aligned}$$

Since we restrict ourselves to the reachability of locations of the subautomata, we allow properties P defined by

$$P := l \mid \neg P \mid P_1 \vee P_2$$

with $l \in \bigcup_{i=1}^{n} L_i$ which then are simply encoded with the corresponding at variables.

In each transition step of the network, a discrete step or a continuous step can be performed. So we construct the final formula $BMC(k)$ for the k-th unfolding of the BMC problem as follows:

$$BMC(k) \quad := \quad I^0 \wedge \bigwedge_{i=0}^{k-1} (T^{i,i+1}_{\text{jump}} \vee T^{i,i+1}_{\text{flow}}) \wedge \neg P^k$$

A. Encoding of the Synchronization Model

For the synchronization model we have to modify the discrete transition. Additionally, in our model of timed automata an edge can be labeled with one synchronization mark $ch \in \text{Ch}_j$ (to keep it simple we allow edges only to be labeled with at most one channel):

$$e = \langle l, g, res, ch, l' \rangle.$$

Synchronized transitions are encoded by introducing a new boolean variable for each channel. We then simply extend $T^{i,i+1}_{\text{jump}}(e,j)$ by setting the boolean variable corresponding to ch to *true* and the variables corresponding to all other channels occuring in TA_j to *false*:

$$T^{i,i+1}_{\text{jump,sync}}(e,j) \quad := \quad T^{i,i+1}_{\text{jump}}(e,j) \wedge ch^i \wedge \bigwedge_{d \in \text{Ch}_j \setminus \{ch\}} \neg d^i$$

In the case that no transition is taken in TA_j this subautomaton does not synchronize on any channel:

$$\text{fix}^{i,i+1}_{\text{sync}}(j) \quad := \quad (at^{i+1}_j = at^i_j) \wedge \bigwedge_{x \in X_{\mathcal{N}}} \neg r^i_{x,j} \wedge \bigwedge_{d \in \text{Ch}_j} \neg d^i$$

Using these extensions it is ensured that all transitions within ch are executed simultaneously. The synchronization model does not have any impact on $T^{i,i+1}_{\text{flow}}$.

B. Encoding of the Bounded Integer Communication Model

In the communication model using bounded integer variables, edges are extended by comparisons (as guards of transitions) and assignments (as actions of transitions) of bounded integer variables to constants. The clock-guards are simply combined with the corresponding comparisons. Since bounded integer variables are global variables we can encode assignments in the same manner as the reset of clock variables by introducing new boolean variables z ($z_{v,j}$ is *true*, if the bounded integer variable v was assigned to in subautomaton TA_j). Similar to the *noreset*-formula we define a formula for bounded integer variables which do not change their values:

$$\text{noassign}^{i,i+1} \quad := \quad \bigwedge_{v \in V_{\mathcal{N}}} \Big(\big(\bigwedge_{j=1}^{n} \neg z^i_{v,j} \big) \implies (v^{i+1} = v^i) \Big)$$

Next, we add the following formula to encode the bounds of the integer variables:

$$\text{range}^{i,i+1} \quad := \quad \bigwedge_{v \in V_{\mathcal{N}}} \big((v^{i+1} \geq v_{\text{min}}) \wedge (v^{i+1} \leq v_{\text{max}}) \big)$$

Let A_e be the set of bounded integer variables which are assigned to on edge e and c_v the corresponding constant for each $v \in A_e$, then we adapt $T^{i,i+1}_{\text{jump}}(e,j)$ and $\text{fix}^{i,i+1}(j)$ as follows:

$$\begin{aligned} T^{i,i+1}_{\text{jump,int}}(e,j) \quad := \quad & T^{i,i+1}_{\text{jump}}(e,j) \\ & \wedge \bigwedge_{v \in A_e} \big(z^i_{v,j} \wedge (v^{i+1} = c_v) \big) \\ & \wedge \bigwedge_{v \in V_{\mathcal{N}} \setminus A_e} \neg z^i_{v,j} \\ \text{fix}^{i,i+1}_{\text{int}}(j) \quad := \quad & \text{fix}^{i,i+1}(j) \wedge \bigwedge_{v \in V_{\mathcal{N}}} \neg z^i_{v,j} \end{aligned}$$

So we finally obtain a discrete step with bounded integer communication:

$$T^{i,i+1}_{\text{jump,int}} := \bigwedge_{j=1}^{n} \Big(\bigvee_{e \in E_j} T^{i,i+1}_{\text{jump,int}}(e,j) \vee \text{fix}^{i,i+1}_{\text{int}}(j) \Big) \wedge \text{noreset}^{i,i+1} \wedge \text{noassign}^{i,i+1} \wedge \text{range}^{i,i+1}$$

The initial state of the network I^0 is extended by setting all bounded integer variables to their initial values by adding $\bigwedge_{v \in V_{\mathcal{N}}} (v^0 = v_{\text{init}})$. We add $\bigwedge_{v \in V_{\mathcal{N}}} (v^{i+1} = v^i)$ to $T^{i,i+1}_{\text{flow}}$ to assure that the bounded integer variables keep their values during a continuous step.

IV. Bounded Model Checking of Incomplete Networks of Timed Automata

In this section we introduce blackboxes for networks of timed automata and introduce a method to solve the unrealizability problem for the two communication models. First, we analyze synchronization channels as an interface between the blackboxes and the implemented system, where BMC can be performed based on so-called fixed transitions, i.e. transitions which do not depend on the behavior of the blackboxes. Second, we allow bounded integer variables to be used as part of the blackboxes interface. In this case, one possibility to solve the unrealizability problem is by introducing universal quantification.

A. Incomplete Networks of Timed Automata

In a network of timed automata we model whole subautomata as blackboxes. Let $\mathcal{N}^{\text{bb}} \subset \mathcal{N}$ be the set of blackboxed automata. While the behavior of these subautomata is unknown, the interface to the remaining network is defined. In the case of the synchronization model, we know all channels of the blackboxes. In the bounded integer communication case, we know all the bounded integer variables which possibly could be assigned in the blackboxes. In the following we only build the BMC formula for $\mathcal{N} \setminus \mathcal{N}^{\text{bb}}$. Similarly, if there is a set of clocks $X^{\text{bb}} \subseteq X_{\mathcal{N}}$ only occurring in the blackboxed automata we can simply ignore them. Note that clocks which are read in the implemented system must not be reset in blackboxes. Otherwise, this would lead to a universal quantification of real-valued variables which is beyond the scope of this paper. Furthermore, we would like to avoid the case that blackboxes are able to eliminate an error path simply by enforcing a timelock [12]. Therefore we require that the initial locations of all possible blackbox implementations are not labeled with an invariant.

B. Blackbox Bounded Model Checking with Synchronization

Our first approach for BMC of incomplete networks of timed automata is restricted to the blackbox interfaces with only synchronization labels. To prove unrealizability it must be possible to find a counterexample which is **independent** of the implementation of the blackboxes. The only way to achieve this, is to find a path into the bad states using transitions which are not labeled with a blackboxed channel. We call these transitions **fixed**. All other transitions may be blocked by a corresponding behavior of the blackboxes. Thus, when a counterexample based only on fixed transitions can be found, the property is not realizable, and vice versa. Let $\text{Ch}^{\text{bb}} \subseteq \text{Ch}_{\mathcal{N}}$ be the set of channels the blackboxes may synchronize on. For any edge $e = \langle l, g, res, ch, l' \rangle$ of the remaining implemented automata we adapt $T^{i,i+1}_{\text{jump,sync}}(e,j)$ as follows:

$$T^{i,i+1}_{\text{jump,sync,bb}}(e,j) := \begin{cases} \mathit{false}, & \text{if } ch \in \text{Ch}^{\text{bb}} \\ T^{i,i+1}_{\text{jump,sync}}(e,j), & \text{else} \end{cases}$$

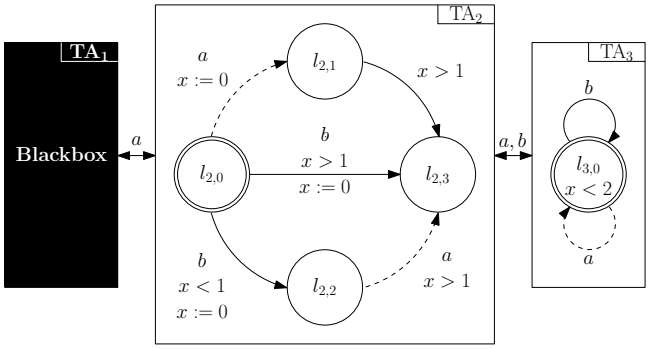


Fig. 2. Incomplete network with synchronization

Example: Figure 2 shows a network of three timed automata which synchronize over two channels a and b. There is one clock x, which is reset only in TA_2. Property P states that location $l_{2,3}$ of TA_2 can never be reached. Assume TA_1 is a blackbox, and that we only know the synchronization channel a that it uses. Now, only those transitions not labeled with a are considered when performing BMC:

$$\begin{aligned}
I^0 &= (at^0_2 = l_{2,0}) \wedge (at^0_3 = l_{3,0}) \wedge (x^0 = 0) \\
T^{i,i+1}_{\text{flow}} &= (at^{i+1}_2 = at^i_2) \wedge (at^{i+1}_3 = at^i_3) \wedge \\
&\quad \big((at^{i+1}_3 = l_{3,0}) \implies (x^{i+1} < 2)\big) \wedge \\
&\quad (\lambda^i > 0) \wedge (x^{i+1} = x^i + \lambda^i) \\
T^{i,i+1}_{\text{jump}} &= \Big[\big[(at^i_2 = l_{2,0}) \wedge (x^i > 1) \wedge (at^{i+1}_2 = l_{2,3}) \wedge \\
&\qquad r^i_{x,2} \wedge (x^{i+1} = 0) \wedge b^i\big] \\
&\quad \vee \big[(at^i_2 = l_{2,0}) \wedge (x^i < 1) \wedge (at^{i+1}_2 = l_{2,2}) \wedge \\
&\qquad r^i_{x,2} \wedge (x^{i+1} = 0) \wedge b^i\big] \\
&\quad \vee \big[(at^i_2 = l_{2,1}) \wedge (x^i > 1) \wedge (at^{i+1}_2 = l_{2,3}) \wedge \\
&\qquad \neg r^i_{x,2} \wedge \neg b^i\big] \\
&\quad \vee \big[(at^{i+1}_2 = at^i_2) \wedge \neg r^i_{x,2} \wedge \neg b^i\big]\Big] \\
&\quad \wedge \Big[\big[(at^i_3 = l_{3,0}) \wedge (at^{i+1}_3 = l_{3,0}) \wedge \neg r^i_{x,3} \wedge b^i\big] \\
&\quad \vee \big[(at^{i+1}_3 = at^i_3) \wedge \neg r^i_{x,3} \wedge \neg b^i\big]\Big] \\
&\quad \wedge \Big[(\neg r^i_{x,2} \wedge \neg r^i_{x,3}) \implies (x^{i+1} = x^i)\Big] \\
P^i &= \neg(at^i_2 = l_{2,3})
\end{aligned}$$

Obviously, the BMC formula for step 2 shown next is satisfied.

$$BMC(2) = I^0 \wedge (T^{0,1}_{\text{jump}} \vee T^{0,1}_{\text{flow}}) \wedge (T^{1,2}_{\text{jump}} \vee T^{1,2}_{\text{flow}}) \wedge \neg P^2$$

One possible solution is the following run of the system (with $\langle at_2, at_3, x \rangle$):

$$\langle l_{2,0}, l_{3,0}, 0 \rangle \xrightarrow{\text{flow}} \langle l_{2,0}, l_{3,0}, 1.5 \rangle \xrightarrow{\text{jump}} \langle l_{2,3}, l_{3,0}, 0 \rangle$$

Since no single transition depends on the synchronization channel a, this run can occur regardless of the implementation of the blackbox (thus, the property is unrealizable). Furthermore, this encoding of fixed transitions yields a conventional SMT formula and can be solved by state-of-the-art SMT solvers like Yices [13] or MathSAT [14].

Next, we discuss communication over bounded integer variables and present an encoding for solving the unrealizability problem based on universal quantification. Note, that these problems cannot be solved with the conventional SMT solvers mentioned above.

C. Blackbox Bounded Model Checking with Bounded Integer Communication

We now extend the blackboxes interface by bounded integer variables. This allows bounded integer variables to be written to inside a blackbox. Thus, we don't know the value these variables hold at any particular time. Since we want to know, whether a safety property is violated for **all** possible implementations of the blackboxes, we have to find a path into the bad states for all possible values of the bounded integer variables. This can be achieved by universally quantifying these variables. First, we determine the set $V^{\text{bb}} \subseteq V_{\mathcal{N}}$ of bounded integer variables which the blackboxes can write to. Similarly to [8], [9] we then build a quantifier prefix for the BMC formula as follows:

$$\underbrace{\exists at^0 x^0 \; \forall v^0_{\text{bb}} \; \exists v^0 \lambda^0 r^0_x z^0_v \overbrace{at^1 x^1}^{T^{\text{jump}}_{1,2} \vee T^{\text{flow}}_{1,2}}}_{T^{\text{jump}}_{0,1} \vee T^{\text{flow}}_{0,1}} \; \forall v^1_{\text{bb}} \; \exists v^1 \lambda^1 r^1_x z^1_v at^2 x^2 \ldots$$

To put it in words, this quantifier prefix asks, whether there exists a state of the system at depth 0 ($\langle at^0 x^0 \rangle$) such that for all blackbox outputs at depth 0 (v^0_{bb}) there is an assignment to the v^0, λ^0, r and z variables leading to a successor state at depth 1 ($\langle at^1 x^1 \rangle$) ..., such that the BMC formula $BMC(k)$ holds.

For every depth k the corresponding successor state for each v^k_{bb} can result either from a discrete step or from a continuous step. Note, that we also have to modify the BMC formula in some way. First, all assignments to the bounded integer variables in V^{bb} are removed from the initial state I^0, and the transition relations $T^{i,i+1}_{\text{flow}}$ and $T^{i,i+1}_{\text{jump}}$. This must be done because the blackboxes can assign any value to these variables at any time. This implies that we have to find a path into the bad states for all values within the bounds. Further, for the universal quantified bounded integer variables, we are only interested in values which are within the bounds of these variables. So for $k > 0$ we use $BMC_{\text{int,bb}}(k)$ defined as follows:

$$BMC_{\text{int,bb}}(k) := \Big(\bigwedge_{j=0}^{k-1} \text{range}(V^{\text{bb},j}) \Big) \implies BMC(k)$$

Example: Figure 3 shows an example of a network of three timed automata TA_1, TA_2 and TA_3 with one clock x and two bounded integer variables v and w. Assume that TA_1 is modeled as a blackbox. Thus, variable v can be written to by the blackbox. Furthermore, in this example we have the additional constraint that v and w are within the range of 1 and 2. Now, to verify that we can reach location $l_{2,2}$ regardless of

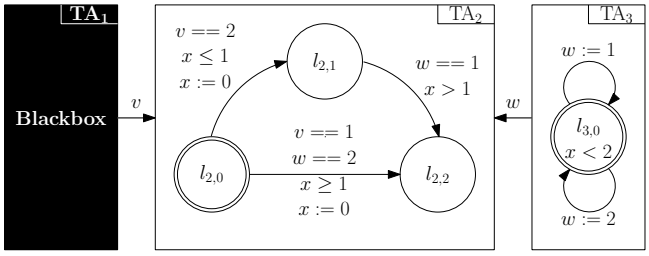


Fig. 3. Incomplete network with bounded integer communication

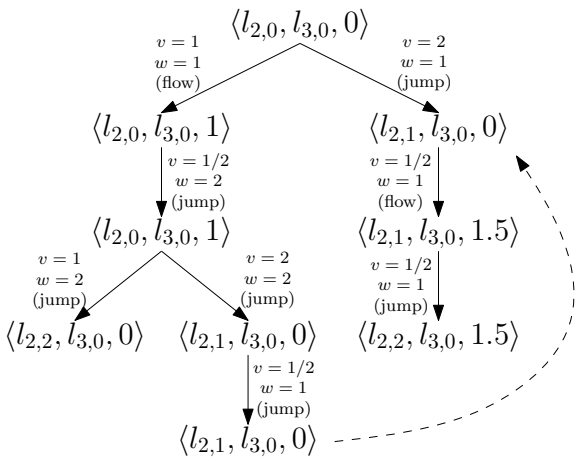


Fig. 4. One possible solution tree for example in Fig. 3

the implementation of the blackbox, we would build the BMC formula analog to the example in Fig. 2 and let the solver try to find solutions for quantified SMT formulas as follows:

$$\begin{aligned}
k = 0: \quad & \exists at^0_2, at^0_3, x^0 : BMC(0) \\
k = 1: \quad & \exists at^0_2, at^0_3, x^0 \forall v^0 \exists w^0, \lambda^0, r^0_{x,2/3,}, z^0_{w,2/3,} \\
& \exists at^1_2, at^1_3, x^1 : BMC_{\text{int,bb}}(1) \\
k = 2: \quad & \exists at^0_2, at^0_3, x^0 \forall v^0 \exists w^0, \lambda^0, r^0_{x,2/3,}, z^0_{w,2/3,} \\
& \exists at^1_2, at^1_3, x^1 \forall v^1 \exists w^1, \lambda^1, r^1_{x,2/3,}, z^1_{w,2/3,} \\
& \exists at^2_2, at^2_3, x^2 : BMC_{\text{int,bb}}(2) \\
& \vdots
\end{aligned}$$

Figure 4 shows one possible solution tree for this example at depth 6.

V. Experimental Results

We implemented a prototype BMC tool to verify incomplete networks of timed automata. It uses the UTAP parser to read benchmarks in the UPPAAL file format and Yices, MathSAT and LIRA [15] as back-end solvers. To evaluate our BMC tool for incomplete networks of timed automata with the synchronization model we used three instances of the Fischer mutual exclusion protocol with 10, 15 and 20 processes [16]. The communication of the processes relies on a synchronization automata which determines which process may enter the critical section. We inserted an error into each process by

processes in total	processes in property (n)	k	complete system	incomplete system
10	2	12	0.23	0.05
	3	26	57.65	2.55
	4	40	9622.51	555.83
	5	-	-	-
15	2	12	0.65	0.10
	3	26	104.87	2.61
	4	40	19774.76	539.53
	5	-	-	-
20	2	12	1.41	0.16
	3	26	330.91	2.69
	4	40	29805.93	633.47
	5	-	-	-

TABLE I
RESULTS FISCHER PROTOCOL

changing a clock constraint consisting of a strict inequality to a non-strict inequality and tested whether a network state can be reached where the first n processes are in the critical section at the same time. Table I gives the results for $n = 2, \ldots, 5$. Note, as was introduced in [17] we use a slightly modified BMC formula alternating T^{jump} and T^{flow}, leading to an additional speedup of the whole process. In the first column one can find the total number of processes of the instance followed by the number of processes checked by the property in the second column. The third column denotes the depth k where the counterexample was found. For the complete case and the incomplete case where all processes not occurring in the property are blackboxed, the times are given in seconds. All experiments were performed on an AMD Opteron processor running at 2.4 GHz with 4 GB of main memory and a timeout of 36,000 seconds.

The results in Table I show the efficiency of abstracting all processes not occurring in the property. By using blackboxes we are able to prove unrealizability for all solvable instances in significant less time. On all benchmarks we are one to two orders of magnitude faster when verifying the incomplete system.

For bounded integer communication, our first approach for BMC based on universal quantification of bounded integer variables relies on current SMT solvers, as they offer an integer data type in their input language. Additionally, they allow universal and existential quantification of variables making it possible to form a quantifier prefix for the resulting SMT formula. But even for the very small example shown in Fig. 3 it was not possible to obtain a result: With conventional SMT solvers we get the answer *UNKNOWN*, since they are not capable of resolving the universal quantifiers due to incompleteness. On the other hand, it also was not possible to prove unrealizability with a complete solver like LIRA, since the computational resources (out of memory) were already exceeded at depth three.

A. Discussion and Future Work

The results show that quantified SMT formulas resulting from the communication model based on bounded integer variables are more challenging. Either the solver is not able to give a result due to incompleteness of resolving quantifiers, or for complete solvers, it exceeds the computational resources available. At the moment we are working on transition relation methods for networks of timed automata where the bounded integer variables are binary encoded. In this case, only boolean variables would be universally quantified allowing us to use adequate data structures such as linAIGs [18] which can efficiently handle universally quantified boolean variables. Alternatively, we are working on a combination of a search-based QBF solver with a conventional SMT solver as back-end. This combination shows great promise in handling the problem class of quantified SMT formulas. So far, we have only allowed blackboxes to manipulate channels and bounded integer variables. One can also think about global clocks which could be reset in several subautomata as well as in each blackbox implementation. In this case, we would have to universally quantify over real-valued variables.

VI. CONCLUSION

In this paper we introduced BMC for incomplete networks of timed automata. For the two communication models *synchronization* and *bounded integer communication* we presented approaches to prove unrealizability. BMC based on fixed transitions shows very promising preliminary results. For solving the class of BMC problems based on universal quantification modifications of current solvers are necessary.

REFERENCES

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, 1999.

[2] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, 2001.

[3] M. Sorea, "Bounded model checking for timed automata," in *ENTCS*, 2002.

[4] P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, O. Maler, and N. Jain, "Verification of timed automata via satisfiability checking," in *Formal Techniques in Real-Time and FaultTolerant Systems FTRTFT*, 2002.

[5] B. Wozna, W. Penczek, and A. Zbrzezny, "Checking reachability properties for timed automata via sat," *Fundamenta Informaticae*, 2002.

[6] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani, "Bounded model checking for timed systems," in *Int'l Conf. on Formal Techniques for Networked and Distributed Systems*, 2002.

[7] M. Herbstritt and B. Becker, "On SAT-based Bounded Invariant Checking of Blackbox Designs," in *MTV*, 2005.

[8] M. Herbstritt, B. Becker, and C. Scholl, "Advanced SAT-techniques for bounded model checking of blackbox designs," in *MTV*, 2006.

[9] M. Herbstritt and B. Becker, "On Combining 01X-Logic and QBF," in *Int'l Conf. on Computer Aided Systems Theory*, 2007.

[10] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.

[11] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *Software Tools for Technology Transfer*, 1997.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[13] B. Dutertre and L. D. Moura, "The yices SMT solver," SRI, Tech. Rep., 2006.

[14] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani, "The mathsat 4 smt solver," in *Int'l Conf. on Computer Aided Verification*, 2008.

[15] B. Becker, C. Dax, J. Eisinger, and F. Klaedtke, "LIRA: Handling constraints of linear arithmetics over the integers and the reals," in *Int'l Conf. on Computer Aided Verification*, 2007.

[16] L. Lamport, "A fast mutual exclusion algorithm," 1986.

[17] E. Ábrahám, B. Becker, F. Klaedtke, and M. Steffen, "Optimizing bounded model checking for linear hybrid systems," in *Int'l Conf. on Verification, Model Checking and Abstract Interpretation*, 2005.

[18] W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz, "Exact state set representations in the verification of linear hybrid systems with large discrete state space," in *ATVA*, 2007.