Combined Bounded and Symbolic Model Checking for Incomplete Timed Systems*

Georges Morbé, Christian Miller, Christoph Scholl, and Bernd Becker

Department of Computer Science, University of Freiburg, Freiburg, Germany
{morbe,millerc,scholl,becker}@informatik.uni-freiburg.de

Abstract. We present a hybrid model checking algorithm for incomplete timed systems where parts of the system are unspecified (so-called black boxes). Here, we answer the question of unrealisability, i.e., "Is there a path violating a safety property regardless of the implementation of the black boxes?" Existing bounded model checking (BMC) approaches for incomplete timed systems exploit the power of modern SMT solvers, but might be too coarse as an abstraction for certain problem instances. On the other hand, symbolic model checking (SMC) for incomplete timed systems is more accurate, but may fail due to the size of the explored state space. In this work, we propose a tight integration of a backward SMC routine with a forward BMC procedure leveraging the strengths of both worlds. The symbolic model checker is hereby used to compute an enlarged target which we then try to hit using BMC. We use learning strategies to guide the SMT solver's search into the right direction and manipulate the enlarged target to improve the overall accuracy of the current verification run. Our experimental results show that the hybrid approach is able to verify incomplete timed systems which are out of the scope for BMC and can neither be solved in reasonable time using SMC. Furthermore, our approach compares favourably with UPPAAL-TIGA when considering timed games as a special case of the unrealisability problem.

1 Introduction

Real-time systems appear in many areas of life, such as time-critical communication protocols or embedded controllers for automobiles. Here, in addition to the logical result, the time when the result is produced is relevant. The correctness of timing constraints is even more important for medical devices or for safety-critical systems as they appear in the transportation domain. For this reason it is crucial to perform *formal verification* of safety-critical systems. Moreover, as these systems steadily grow in complexity, verifying their correctness becomes harder and increasingly more important. Nowadays, timed automata (TAs) [2,1], which are an extension of conventional discrete automata by real-valued clock variables, are a common model for real-time systems and have become a standard in industry.

* This work was partly supported 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, http://www.avacs.org/).

E. Yahav (Ed.): HVC 2014, LNCS 8855, pp. 30–47, 2014.

In this work, we focus on the verification of *incomplete* timed systems, i.e., timed systems that contain unknown components (so-called black boxes). The purpose is to add a layer of abstraction if a design is too large to verify in its entirety, or to allow to start the verification process earlier when certain components of the design are only partially completed. Here, we aim to refute the realisability of a property, that is, we tell the designer, no matter how you implement the unknown parts of the system, the property will always fail. To put it in other words, the error is already in the implemented system. More formally, we prove, given an incomplete system T and a safety property ϕ (unsafe states are not reachable), that no matter what the black box BB looks like, the parallel composition of T and BB cannot satify ϕ. If this is the case, then we call the property *unrealisable*.

The unrealisability problem generalises the controller synthesis problem [14,4,11]. Here, the system communicates with an unknown controller which is more powerful than the remaining system in the sense that it may always enforce an immediate interaction with the system. In contrast to the controller synthesis problem, our scenario defines the black box as an equitable part of the system having the same impact as the implemented components. (However, we will see later, that it is also possible to define special black boxes having the same power as the unknown controller in timed games.)

Whereas some approaches to controller synthesis look into properties like LTL [12], TCTL [13] or MTL [6], we restrict our attention to safety properties which state the unreachability of certain discrete states in a timed system (as already mentioned above).

One possible method to prove unrealisability of properties in incomplete timed systems is bounded model checking (BMC). Generally, BMC starts with the initial state, iteratively unfolds the system k times, adds the negated property, and converts the BMC instance into a satisfiability problem which is solved by an appropriate solver. If the k-th instance is satisfiable, a path of length k violating the property has been found. BMC instances for real-time systems are typically encoded into so-called SAT-Modulo-Theory (SMT) formulas, since they are augmented with continuous time constraints over real-valued variables. BMC for incomplete timed systems is studied in [16,17,15]. In this paper, we use their encoding where the verification problem is limited to those transitions which are independent of the behaviour of the black boxes. This approach still yields SMT formulas which typically are easy to solve, however, due to its approximative nature this encoding limits the class of problems which can be verified. A second option for solving the unrealisability problem is symbolic model checking (SMC) for incomplete timed systems. In general, beginning with those states directly violating the property, SMC performs a backwards traversal of the state space using adequate data structures for the symbolic representation and manipulation of the state space and the transition functions. If at some point, the so far explored states include the initial state, there exists a path leading to an unsafe state. For our work, we use the symbolic model checking algorithm presented in [19]. It verifies incomplete finite state machines with time (FSMT) [20,18],

which is a formal model to represent incomplete real-time systems[1]. State sets and transition functions are represented using so-called LinAIGs [9,22,8], a data structure which can hold arbitrary Boolean combinations of Boolean variables and linear constraints over real-valued variables. SMC for incomplete FSMTs is more accurate than BMC for incomplete timed systems, however, it often fails due to the size of the state sets which are generated along the verification task.

In this paper, we adopt the idea of combining BMC and SMC for incomplete discrete systems [21] to the timed world. In the timed world the main challenge is that various (known and unknown) components of the system can influence the time evolution by enforcing and/or preventing certain discrete steps. We present a verification algorithm where we use SMC for incomplete FSMTs to compute an enlarged target which we then try to hit using BMC. This tight integration in combination with learning strategies and on-the-fly manipulations of the enlarged target makes it possible to verify incomplete timed systems, which are out of reach for BMC respectively SMC alone. In other words, our approach makes BMC for incomplete timed systems more accurate and prevents SMC for incomplete timed systems from exploring state sets which are too big to verify. To show the efficacy of our hybrid verification technique, we give preliminary experimental results using multiple parameterized timed benchmarks. Furthermore, our results show that we are able to outperform the state-of-the-art controller synthesis tool UPPAAL-TIGA, when considering timed games as a special case of the unrealisability problem.

The paper is structured as follows. In Section 2, we review incomplete networks of timed automata and BMC on the one hand, and FSMT-based SMC on the other hand. Our novel method is given in Section 3. After presenting experimental results in Section 4 we conclude the paper in Section 5.

2 Preliminaries

2.1 Timed Automata

Real-time systems are often modelled using timed automata (TAs) [1,2], an extension of conventional automata by a set X of real-valued clock variables to represent the continuous time. The set of clock constraints $\mathcal{C}(X)$ contains atomic constraints of the form $(x_i \sim d)$ and $(x_i - x_j \sim d)$ with $d \in \mathbb{Q}$, $x_i, x_j \in X$ and $\sim \in \{<, \leq, =, \geq, >\}$. We consider TAs extended with bounded integer variables. Let Int be a set of bounded integer variables each having a fixed lower and upper bound. Let $Assign\,(Int)$ be the set of assignments to integer variables. Let $\mathcal{C}(Int)$ be a set of constraints of the form $(int_i \sim d)$ and $(int_i \sim int_j)$ with $d \in \mathbb{Z}$, $\sim \in \{<, \leq, \geq, >\}$ and $int_i, int_j \in Int$. Let $\mathcal{C}_c(X, Int)$ be the set of conjunctions over clock constraints and constraints from $\mathcal{C}(Int)$. Using this information we define a timed automaton as follows:

Definition 1 (Timed Automaton). *A timed automaton (TA) is a tuple* $\langle L, l_0, X, Act, Int, lb, ub, E\rangle$ *where* L *is a finite set of locations,* $l_0 \in L$ *is an*

[1] Note that networks of timed automata can easily be transformed into FSMTs.

initial location, $X = \{x_1, \ldots, x_n\}$ *is a finite set of real-valued clock variables,* $Act = Act_{nu} \cup Act_u$ *with* $Act_{nu} \cap Act_u = \emptyset$*,* Act_{nu} *is a finite set of non-urgent actions and* Act_u *is a finite set of urgent actions,* $Int = \{int_1, \ldots, int_m\}$ *is a finite set of integer variables,* $lb : Int \to \mathbb{Z}$ *and* $ub : Int \to \mathbb{Z}$ *assign lower and upper bounds to each* $int_i \in Int$ *with* $lb(int_i) \leq ub(int_i)$ *for* $1 \leq i \leq m$*,* $E \subseteq L \times \mathcal{C}_c(X, Int) \times (Act \cup \{\epsilon_u, \epsilon_{nu}\}) \times 2^X \times 2^{Assign(Int)} \times L$ *is a set of transitions with* $E = E_{nu} \cup E_u$*.* $E_{nu} = \{(l, g_e, a, r_e, Assign_e, l') \in E \mid a \in Act_{nu} \cup \{\epsilon_{nu}\}\}$ *is the set of non-urgent transitions and* $E_u = \{(l, g_e, a, r_e, Assign_e, l') \in E \mid a \in Act_u \cup \{\epsilon_u\}\}$ *is the set of urgent transitions. If for* $e = (l, g_e, a, r_e, Assign_e, l') \in E$ *it holds that* $a \in Act$*, then we call* e *a transition with an (urgent or non-urgent) synchronising action, if* $a \in \{\epsilon_u, \epsilon_{nu}\}$ *then we call* e *an (urgent or non-urgent) transition without synchronising action.*

A state $s_i = \langle l_i, \nu_i, \mu_i \rangle$ of a TA is a combination of a location l_i and a valuation ν_i of the clock variables and a valuation μ_i of the integer variables. A TA may perform a continuous transition, that is, all clock variables evolve over time with the same rate and neither the location nor the values of the integer variables change. Discrete transitions describe the change of the location. A discrete transition happens instantaneously and can only be taken, if its guard is satisfied, that is, if the transition is *enabled*. We consider networks of timed automata having an interleaving semantics, however, transitions labelled with the same action have to be taken simultaneously. If several transitions without action are enabled at the same time it is chosen non-deterministically which one is taken.

A discrete transition may be declared as urgent. Whenever an urgent transition labelled by ϵ_u is enabled, the current state must be left without any time delay. Analogously, whenever in all components containing a^u-transitions with $a^u \in Act_u$ a transition labelled with a^u is enabled, then there must not be any time delay before taking any transition. In literature, location invariants are used to enforce discrete transitions. However, in [18,19] the authors showed that TAs having closed invariants can be converted into semantically equivalent TAs using urgency. The translation consists in adding a supplementary urgent transition once the upper limit of the invariant is reached. A similar technique is used in the context of timed games where "forced transitions" labelled with upper limits of invariants are added in order to prevent one player from forcing the system into a timelock [3].

2.2 Incomplete Networks of Timed Automata

In this paper, we focus on *incomplete* networks of TAs where parts of the system are not specified (black box (BB)), however, the interface to the remaining system (white box (WB)) is defined. Here, we aim to prove the *unrealisability* of a safety property. We call a property unrealisable if there exists no replacement of the BB such that the property holds for the resulting overall design.

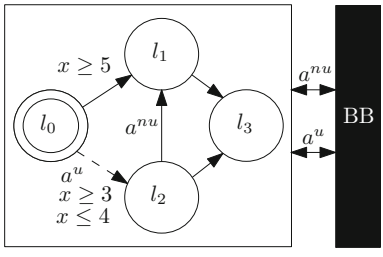

Fig. 1. Incomplete Timed System

The interface of BB components and WB components consists of non-urgent and urgent synchronisation actions. Since the behaviour of the BB is unknown it is unclear when the BB is ready to synchronise. In fact, in case of an urgent action, the BB is even able to stop time evolution of the whole system until the synchronisation takes place or until the conditions enabling the synchronising transition are not fulfilled anymore. In other words, allowing the above mentioned communication methods to define the interface of the BB, the unknown parts are able to affect the discrete behaviour of the WB and, in case of urgency, also may influence their timing behaviour. In the following, we distinguish three kinds of discrete transitions: *f-transitions* (also referred to as fixed transitions) are not labelled with any action synchronising with the BB, *nu-transitions* are labelled with a non-urgent action synchronising with the BB, and *u-transitions* are labelled with an urgent action synchronising with the BB.

Example 1. Consider the incomplete timed system shown in Fig. 1. The BB uses a non-urgent action a^{nu} and an urgent action a^u to interact with the implemented system. The nu-transition from l_2 to l_1 can only be taken if it synchronises with the BB via a^{nu}, and thus, the BB is able to influence the behaviour of the system in that way. Being in location l_0, the BB has the power to enforce the u-transition labelled with a^u leading to l_2 (dashed arrow) when $3 \leq x \leq 4$. In that case, time evolution is stopped and a discrete transition has to be taken, instantaneously. All remaining transitions in the system are f-transitions which can not be affected through the BBs behaviour.

2.3 Bounded Model Checking of Incomplete Networks of Timed Automata

Generally, the BMC procedure [5,7] starts with the initial state I^0 (superscript numeral denotes the unfolding depth), iteratively unfolds a system k times by adding a conjunction of transition relations $T^{i,i+1}$, and connects the negated property $\neg P^k$. Finally, the BMC instance is converted into a satisfiability problem. If an appropriate solver finds the k-th problem instance satisfiable, a path of length k violating the property has been found.

When proving unrealisability using BMC, the unknown behaviour of the BB needs an adequate modelling. Taking incomplete networks of TAs into account, BMC based on fixed transitions [16,17] is one option to solve the unrealisability problem as long as the system does not contain any u-transitions. The idea is that the BB can prevent the system from taking the nu-transitions on a path to an error state (by disabling those transitions). Thus, nu-transitions are omitted in the search for an error path, which does not depend on the BB behaviour, and the transition relation is reduced to f-transitions. To encode the transition relation, we differentiate between a discrete step $T^{i,i+1}_{\text{jump}}$, which describes all possibilities of changing a location via f-transitions, and a continuous step $T^{i,i+1}_{\text{flow}}$, where all subautomata stay in their locations, and time passes equally for all clocks (for a detailed description of the BMC encoding please refer to [16,17]). Finally, the

k-th BMC instance is constructed as follows

$$BMC(k) := I^0 \wedge \bigwedge_{i=0}^{k-1} \begin{cases} T_{\text{jump}}^{i,i+1} \text{ if } i \text{ is even,} \\ T_{\text{flow}}^{i,i+1} \text{ otherwise} \end{cases} \wedge \neg P^k \tag{1}$$

using an alternation of discrete and continuous steps and then passed to an arbitrary SMT solver, which supports the theory of linear arithmetic for integers and reals. If $BMC(k)$ is satisfiable, there is a run $r = \langle s_0, s_1, \ldots, s_k \rangle$ of length k with $s_i = \langle l_i, \nu_i, \mu_i \rangle$, $0 \leq i \leq k$ and l_i being a location, ν_i a clock valuation, and μ_i being an integer valuation which does not depend on the BB behaviour and leads to a state s_k that violates the property P.

Whereas this encoding yields easy-to-solve SMT formulas in many cases, the model in [16] assumes that the BB may only enable and disable transitions in the WB, but never may *enforce* transitions in the WB. Thus the BB may not synchronise over urgent actions, i.e., the implemented system may not contain any u-transitions. This is a really strong restriction and limits the class of problems that can be verified.

Example 2. In this example, we show that unrealisability proofs based on f-transitions may be wrong, if u-transitions are present in the WB. We consider the incomplete timed system given in Fig. 1 and the property that the location l_1 can never be reached. The BMC procedure as defined above confines the consideration to f-transitions and it would find a path leading to l_1 (e.g. $\langle l_0, x = 0 \rangle \rightarrow \langle l_0, x = 5 \rangle \rightarrow \langle l_1, x = 5 \rangle$). However, due to the u-transition which is enabled in location l_0 once x reaches the value 3, this error path is not valid for *all* possible black box implementations, since the BB may stop time evolution and force the network to synchronise via a^u. If afterwards, the BB never synchronises via a^{nu}, then l_1 can never be reached, i.e., there is a BB-implementation which fulfills the property and unrealisability does not hold.

2.4 Symbolic Backward Model Checking Based on FSMTs

The symbolic methods we will use are based on finite state machines with time (FSMTs) [20,18] which are symbolic representations of networks of TAs well suited for fully symbolic model checking algorithms. Basically, an FSMT consists of a set of Boolean location variables Y, a set of real-valued clock variables X, a set of Boolean input variables I, a predicate *init* describing the set of initial states, and a predicate *urgent* indicating when an urgent transition is enabled. Each location variable $y_i \in Y$ is determined by a transition function δ_i, and reset conditions $reset_{x_i}$, which deterimine when each clock variable x_i is reset. A state of an FSMT is a valuation of the clock variables and the location variables. An FSMT performs a discrete step depending on the current state and the input variables. Here, the location variable y_i is set to 0 (1) iff δ_i evaluates to 0 (1) and the clock x_i is reset iff $reset_{x_i}$ evaluates to 1. In a continuous step time may pass unless the *urgent* predicate evaluates to 1. FSMTs communicate by reading each other's location variables, clock variables, and shared input

variables. In [20] the authors show how to translate timed systems into FSMTs (integer communication is encoded using Boolean variables which are included into Y). In [18,19] this translation is extended to incomplete timed systems. Here, we additionally define a predicate $urgent_{BB}$, which evaluates to 1 if any u-transition is enabled.

The symbolic backward model checking algorithm for incomplete timed systems presented in [19] starts with the negation of a given safety property and computes predecessor state sets until the initial state is reached. The generated state sets contain those states from which the negated property is reachable regardless of the behaviour of the BB. Similarly to BMC, we define two kinds of predecessor operators. Starting from a state set ϕ the *discrete predecessor operator* computes a state set $Pre_d(\phi)$ containing only states from which ϕ is reachable taking a discrete transition in the WB independently from any BB behaviour. The BB can not prevent the WB from taking f-transitions and thus, only those are considered for the computation of the discrete predecessor (u-transitions and nu-transitions can be blocked when the BB is not sending the appropriate synchronisation action). The *continuous predecessor operator* computes a state set $Pre_c(\phi)$, which contains only states from which ϕ is reachable regardless of the behaviour of the BB when performing continuous transitions. A state s is added to $Pre_c(\phi)$ if it is possible to reach ϕ from s through time evolution which may not be blocked by the BB. Here we have to account for the fact that the BB is able to influence the timing behaviour of the WB by sending urgent actions and thus enabling u-transitions which stop time evolution. Consider the case that a state s_ϕ in ϕ is reachable through time evolution starting in s, but there is another state t on the way between s and s_ϕ which is the source of an enabled u-transition labelled with the urgent action a^u. Now we have to consider two cases in order to decide whether s can be included into $Pre_c(\phi)$:

(1) Assume that all enabled u-transitions labelled with a^u and starting from t lead to some state which is not in ϕ. By sending the action a^u the BB can cause time evolution to stop and it can impede the WB from reaching ϕ starting in s. Thus we may not include s into $Pre_c(\phi)$.
(2) Assume that there is some enabled u-transition that starts in t, is labelled with a^u, and leads to a state $t'' \in \phi$. In that case, the choice of the BB is irrelevant, since in both cases (synchronising via a^u gives the WB the opportunity to move to $t'' \in \phi$, not synchronising leads to ϕ through time evolution) the resulting state is included in ϕ. Thus, in this case, and if there is no other state $\tilde{t}$ on the way from s to s_ϕ fulfilling the condition of case (1), we may include s into $Pre_c(\phi)$ and, since we are only interested in the question from which states we may reach ϕ regardless of the BB behaviour (and not how), we do not need to consider stopping time evolution at t due to urgent synchronisation with the BB.

In order to know whether the WB is able to force u-transitions into some state in ϕ or not, which is crucial for the continuous predecessor operator, SMC has to be able to compute a special discrete predecessor state set of ϕ only over u-transitions (in the following named *pre-urgent (PU)* respectively $Pre_d^u(\phi)$). Put

in other words, a state s is included into $Pre_d^u(\phi)$ if for all urgent actions on enabled outgoing transitions from s, there is at least one enabled u-transition in the WB synchronising over this action, which leads to ϕ. For details on the exact computation of $Pre_d(\phi)$, $Pre_c(\phi)$, and $Pre_d^u(\phi)$ we refer to [19].

Example 3. As an example consider again the incomplete timed system of Fig. 1 and a state set ϕ_1 with $\langle l_1, x \geq 0\rangle \in \phi_1$. Then the state $s_1 = \langle l_2, x \geq 0\rangle$ is no discrete predecessor of ϕ_1, $s_1 \notin Pre_d(\phi_1)$, since it is backwards reachable from ϕ_1 only over the nu-transition (l_2, l_1), which can be disabled by the BB not sending the action a^{nu}. State $s_2 = \langle l_0, x \geq 5\rangle$, reachable via the f-transition (l_0, l_1) is the only discrete predecessor of ϕ_1, $Pre_d(\phi_1) = \{s_2\}$, as it is the only state reachable over f-transitions.

Consider now another state set ϕ_2 with $\langle l_2, x \geq 0\rangle \in \phi_2$ and $\langle l_0, x \geq 5\rangle \in \phi_2$. We ask the question whether the state $s_0 = \langle l_0, x = 0\rangle$ can be included into $Pre_c(\phi_2)$. When the BB does not interfere by sending the urgent action a^u, $\langle l_0, x \geq 5\rangle$ and thus ϕ_2 is reachable from s_0 through normal time passing. By sending a^u in any state $s_3 = \langle l_0, x = c\rangle$ with $3 \leq c \leq 4$, the BB can stop time evolution. However, in that case the WB can take the enabled u-transition (l_0, l_2) leading to the state $\langle l_2, x = c\rangle$ which is in ϕ_2 (since $\langle l_2, x = c\rangle \in \phi_2$, s_3 is in pre-urgent of ϕ_2, i.e., $s_3 \in Pre_d^u(\phi_2)$). Hence state set ϕ_2 is reachable starting in s_0 independently from the BB behaviour and thus, $s_0 \in Pre_c(\phi_2)$.

3 Hybrid Verification of Incomplete Real-Time Systems

In this section we present a hybrid BMC/SMC algorithm to prove unrealisability in incomplete networks of TAs which (1) makes it possible for BMC to handle u-transitions and (2) avoids full SMC runs possibly exceeding resources.

To this end, we extend the BMC encoding given in Section 2.3 by the possibility to handle urgency. We modified the initial state and the transition relations in a way that time evolution is blocked immediately (i.e. the length of the time evolution is enforced to be 0) whenever either an urgent transition is enabled or an urgent synchronisation within WB components can take place. We call such a state where time evolution has to be blocked immediately an *urgent state*. Furthermore, time evolution started in a non-urgent state is stopped when an urgent state is reached.[2] BMC as defined above computes a path based on f-transitions without considering the timing constraints imposed by u-transitions. In that way, BMC *over-approximates* the set of possible runs leading to an error state, however, our novel approach excludes spurious error paths by using SMC methods.

The idea of the overall algorithm combining BMC and SMC is as follows: We use SMC to compute an enlarged target, that is a symbolic representation of states from which there exists a path to the negated property no matter how the BB is implemented. We then try to hit this target by searching a path via f-transitions which starts in the initial state of the incomplete network and

[2] Since we need a well-defined starting point in time for the urgent state, we forbid constraints with '$>$' instead of '$\geq$' guarding any urgent transition.

finally ends in one of the states of the enlarged target. Whenever along this path there is a state s with an enabled u-transition, we additionally test whether for all urgent actions on enabled outgoing transitions from s, there is at least one enabled u-transition in the WB, synchronising over this action, which leads to the enlarged target. In this case we say "the WB can force the u-transitions into the enlarged target". Then the decision of the BB is irrelevant and the error state is reached in every case. If not, we have to extend the BMC problem by additional information learnt from SMC.

In the following, we describe the algorithm in general and then give a detailed description of the interaction with the enlarged target and the pre-urgent state set, respectively.

3.1 Overall Algorithm

Algorithm 1 shows the procedure to prove unrealisability of a property in an incomplete network of TAs, and consists of two steps, in the first step (lines 5 to 17) BMC is used to search for a fixed path, reaching ET, based on f-transitions, and in the second step, using SMC methods, it is checked whether the WB can force all enabled u-transitions along this error path candidate into ET (lines 20 to 31). To this end, we compute the set $Pre_d^u(ET)$ of all states which have the property that the WB can force all outgoing u-transitions into the enlarged target and check whether all states on the error path candidate, which have enabled u-transitions, are included in $Pre_d^u(ET)$. We call this test the "PU inclusion check". The enlargement of ET and the necessitated manipulations of the SMT solver are described in Algorithm 2.

We introduce a set Π_{ET} holding the conflict clauses, which are generated after an unsuccessful inclusion check of the enlarged target. Next, we store in a set Π_{PU} additional constraints resulting from the PU inclusion check, which restrict the continuous transitions of future BMC runs. Furthermore, let BMC be an SMT formula representing the current BMC instance and nr_of_fixed_paths a counter, storing the number of so far explored paths based on f-transitions. Lastly, we use a predefined number K which limits the number of BMC steps. In informal words, K defines the influence of BMC in the combined approach. As a special case consider $K = 0$ where a pure SMC run is performed.

After initialising the procedure (lines 1–2), the enlarged target (ET) is computed (line 3) by performing a predefined number ($\#\text{steps}_{\text{init}}$) of continuous and discrete backward steps (Pre_c and Pre_d) as described in Section 2.4, using the negated property as a starting point. In lines 5 to 17 the algorithm searches for a path based on f-transitions. When constructing the BMC formula we omit the negated property, and thus, the pure BMC instance (i.e. the BMC instance without any additional conflict clauses) is naturally satisfiable. The state (assignment to location variables, clocks variables and integer variables) computed by the SMT solver at the end of the last unfolding is checked for inclusion in ET (line 8). If this check is successful, a path based on f-transitions has been found and we continue with the next step of the algorithm. However, in the negative case a conflict clause, which forbids the current assignment to the state variables

Algorithm 1. Hybrid Algorithm BMC–SMC

```
1: Π_ET = ∅; Π_PU = ∅; k = 0; BMC = I^0; nr_of_fixed_paths = 0;
2: add_transition_relation_to_BMC();
3: compute_ET(#steps_init);
4: while true do
5:     fixed_path_found = false;
6:     while !fixed_path_found do
7:         if SMT_solve(BMC) == SAT then
8:             if is_in_ET(state_vars(k)) then
9:                 fixed_path_found := true; nr_of_fixed_paths++;
10:            else
11:                π = generate_ET_cc; Π_ET = Π_ET ∪ π; add_to_solver(π, k);
12:        else
13:            k = k + 1;
14:            if k < K then
15:                add_transition_relation_to_BMC();
16:                remove_from_solver(Π_ET, k − 1); add_to_solver(Π_ET, k);
17:                if k is continuous transition then add_to_solver(Π_PU, k);
18:            else
19:                enlarge_ET_and_reset(#steps);
20:    fixed_path_valid = true;
21:    while fixed_path_valid && untested continuous transition s^i → s^{i+1} exists
       along fixed path do
22:        if !check_PU(s^i, s^{i+1}) then
23:            fixed_path_valid = false;
24:            if nr_of_fixed_paths < max_fixed_paths then
25:                π = generate_PU_constraint;
26:                for j=0...k do
27:                    if j is continuous transition then
28:                        add_to_solver(π, j);
29:                Π_PU = Π_PU ∪ π;
30:            else
31:                enlarge_ET_and_reset(#steps);
32:    if fixed_path_valid then return Unrealisability proven
```

at the end of the last unfolding is generated, added to the BMC problem and thus, prevents the solver to explore the same path of length k again (see Section 3.2 for details). The solver is invoked again and the new solution for the state variables of the last unfolding is tested, etc.. This procedure is repeated until either a path of length k leading into ET has been found, or the BMC instance including all generated conflict clauses gets unsatisfiable. In the latter case, there exists no path of length k into ET and we continue the search for $k + 1$. The generated conflict clauses exclude states which are not part of ET, and thus, they contain valuable information for future unfolding depths. Since the index of the last unfolding has increased by 1, prior to the new search, we

Algorithm 2. enlarge_ET_and_reset(#steps)

1: enlarge_ET(#steps);
2: **if** $I \cap ET \neq \emptyset$ **then return** `Unrealisability proven`
3: **else if** fixed_point(ET) **then return** `Unrealisability not proven`
4: reduce_k = min(#steps $-$ 1, k); $k = k -$ reduce_k;
5: remove_transition_relations(reduce_k);
6: update_ET_cc(Π_{ET}); add_to_solver(Π_{ET}, k);
7: remove_PU_constraints_from_solver(); $\Pi_{PU} = \emptyset$;
8: nr_of_fixed_paths=0;

remove all conflict clauses having time index k and add them again with the incremented time index $k + 1$ (line 16).

If it is not possible to find a fixed path within K steps (line 18), the enlarged target is further extended by '#steps' backward steps and the combined procedure is reset (see line 19 and Algorithm 2).

Once a fixed path leading to ET is found (line 9), Algorithm 1 continues to check whether the BB is able to force the implemented system to leave that path by sending urgent actions (lines 20 to 31). That is, for each continuous transition along the path, we test whether there is any u-transition enabled during this time step, and – in a positive case – whether the WB can force all enabled u-transitions into ET (see Section 3.3 for details on the interaction with this state set). If this test is successful for all continuous transitions, the current fixed error path is valid and unrealisability of the system has been proven. Once this test fails for some continuous transition (line 22), the BB is able to enforce a transition leaving the current fixed error path. To avoid this situation in the future, a constraint is generated and added to Π_{PU} (lines 25 to 29). Since the current fixed error path is not valid, the algorithm starts searching for a new fixed path taking all conflict clauses included in Π_{ET} and all constraints in Π_{PU} into account.

It may be the case that the current ET is too small to lead to a result (if the BB is able to enforce transitions leaving the current fixed error path into states not in ET). Therefore from time to time (when a predefined amount of fixed paths using the same ET has been explored) we enlarge the target further (line 31). In that case, the overall accuracy is increased by expanding the ET which in turn necessitates a restart of the procedure.

Algorithm 2 performs a target enlargement (line 1) followed by necessary manipulations of the BMC instance. After the target enlargement we check in line 2 whether the new ET already contains some initial states which proves unrealisability (line 2). A fixed point check determines whether new states could be added to the enlarged target. If not, it is clear that we will never be able to prove unrealisability (line 3). However, in case new states could be added, the algorithm is restarted using the new ET as a basis after removing the last $\#steps - 1$ transition relations from the BMC problem in lines 4–5 (since it has been proven that there exists no fixed paths at previous unfolding depths). In

order to keep as much learnt information as possible for the next search of a fixed path, we update Π_{ET} in update_ET_cc() as follows: For each conflict clause in Π_{ET} (which describes a state set not belonging to the old ET) we test whether it still describes a state set not belonging to the extended ET. If the outcome of the test is negative, we have to remove the conflict clause from Π_{ET}. At the end, the updated ET conflict clauses again describe state sets which should be excluded from the solvers search and the conflict clauses are added to the last time frame k (line 6). In contrast, the so far generated PU constraints limit the timing behaviour of continuous transitions in BMC based on the old ET and thus, they might prevent the solver from finding valid error paths in future runs (after extending ET). Hence, we remove all PU constraints from the solver and set $\Pi_{PU} = \emptyset$ (line 7).

3.2 Enlarged Target Inclusion Check and Conflict Clause Generation

One connection point of BMC and SMC is a test whether a state $s_k = \langle l_k, \nu_k, \mu_k \rangle$ of the last BMC unfolding is included in ET (line 8 of Algorithm 1). If the test fails, a conflict clause is generated and passed to the SMT solver to prevent the search from exploring the same branch again. The inclusion check and the conflict clause generation is performed in three steps: First, we test whether there exists any state in ET having location l_k. This can be reduced to an SMT check fixing in ET the location variables $y_1, \ldots, y_l$ to the valuation ξ_k, which represents l_k. If not, it is clear that s_k cannot be part of ET as well and another possible error path has to be found. As a conflict clause, it would be sufficient to exclude only ξ_k, however, we can exclude a larger part of the search space when lifting the location variables $y_1, \ldots, y_l$: A location variable y_i can be removed from the conflict clause if the assignment to y_i is irrelevant for the SMT check. As a result, the conflict clause contains only those location variables which are essential to exclude l_k and additionally excludes further states which are not part of ET. However, if ET contains states having the location l_k, the second step tests for the integer valuation μ_k. In a negative case, the conflict clause forbids $\langle l_k, \mu_k \rangle$. Otherwise, a third test includes the valuation ν_k of the clock variables. If this SMT check succeeds as well, s_k is part of ET and a fixed error path hitting ET has been found. If not, instead of s_k, we can exclude $\langle l_k, Z, \mu_k \rangle \notin ET$ with Z being a clock zone which contains ν_k. We obtain Z by a conjunction of all linear clock constraints of ET which are satisfied by ν_k and the negations of all clock constraints which are not satisfied by ν_k.

3.3 Pre-Urgent Inclusion Check and Conflict Constraint Generation

In contrast to the ET inclusion check, where one specific state $s_k = \langle l_k, \nu_k, \mu_k \rangle$ was considered, the check whether a *u-transition* leads to ET needs to be performed for *all* states in which a u-transition is enabled. The PU inclusion check takes as an input a complete continuous BMC step of length λ from the state

$s^i = \langle l, \nu, \mu \rangle$ to its successor state $s^{i+1} = \langle l, \nu + \lambda, \mu \rangle$ along the current path based on f-transitions. Using the $urgent_{BB}$ predicate of the incomplete FSMT, SMC computes the set of states ϕ^u in which a u-transition is enabled along this continuous transition. If ϕ^u is empty, the BB is not able to enforce the WB to leave this path, and the procedure simply returns *true*. If not, the procedure checks whether ϕ^u is completely included in $Pre_d^u(ET)$, that is, the WB is able to force all u-transitions emerging from states in ϕ^u into ET. In a positive case the procedure returns *true* and continues with the succeeding continuous transition of the candidate error path. However, if there is some state s in ϕ^u, such that the WB is *not* able to force the u-transitions starting in s into ET, we generate a constraint which prevents the SMT solver to take the same continuous transition in future BMC runs. To this purpose, through SMT solving, we pick one single state $\tilde{s} \in \phi^u \setminus Pre_d^u(ET)$, and similarly to the ET conflict clause generation, we compute a state set $\tilde{\phi}$ with $\tilde{\phi} \cap Pre_d^u(ET) = \emptyset$, and furthermore restrict this set to states which are source of a u-transition by using the $urgent_{BB}$ predicate again. In this way, we obtain a *critical state set* $\langle l, \bigwedge_{i=1}^{n} (a_i \leq x_i \leq b_i), \mu \rangle$ which must be avoided in future *continuous* steps. To avoid the critical state set, we add to each continuous time frame of the BMC problem a constraint which is a conjunction of the following two conditions:

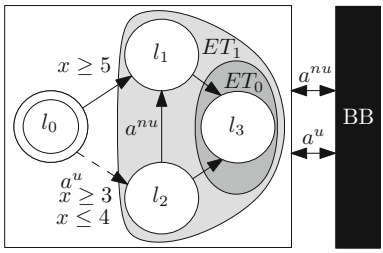

Fig. 2. Example

(a) If we are in location l with integer valuation μ and the valuation of at least one clock variable x_j of the starting point of a continuous transition is lower than a_j, then time passing has to be stopped before all clock variables x_k obtain valuations greater than a_k. In this way, reaching the critical state set through time evolution is prevented.

$$\Big[(s^i = l) \wedge (int^i = \mu) \wedge \big(\bigvee_{j=1}^{n} (x_j^i < a_j)\big)\Big] \implies \big(\bigvee_{j=1}^{n} (x_j^{i+1} \leq a_j)\big)$$

(b) It might be possible that a state of the critical state set is reached through a *discrete transition. In this case, we have to ensure that time must not proceed, that is, another discrete transition must be taken immediately.*

$$\Big[(s^i = l) \wedge (int^i = \mu) \wedge \big(\bigwedge_{j=1}^{n} (a_j \leq x_j^i \leq b_j)\big)\Big] \implies \big(\bigwedge_{j=1}^{n} (x_j^{i+1} = x_j^i)\big)$$

We consider the example shown in Figure 2 and the property that location l_3 can never be reached. Assume that initially the enlarged target (ET_0) contains only the location l_3. Next, BMC finds a path based on f-transitions leading into ET_0 as follows:

$$p_0^{\mathrm{fix}} = \langle l_0, x = 0 \rangle \rightarrow \langle l_0, x = 5 \rangle \rightarrow \langle l_1, x = 5 \rangle \rightarrow \mathrm{ET}_0$$

During the time step, the BB may enforce the u-transition e^u from l_0 to l_2 when x has a value between 3 and 4. To check whether p_0^{fix} is still valid in this situation we perform the PU inclusion check for the time step $\langle l_0, x = 0\rangle \rightarrow \langle l_0, x = 5\rangle$ which tests whether $\langle l_0, 3 \leq x \leq 4\rangle$ is included in $Pre_d^u(\text{ET}_0) = \emptyset$. Obviously, this is not the case. To prevent time steps from touching $\langle l_0, 3 \leq x \leq 4\rangle$, we add the constraint

$$\Big[(s^i = l_0) \wedge (x^i < 3) \implies (x^{i+1} \leq 3)\Big] \wedge \Big[(s^i = l_0) \wedge (3 \leq x^i \leq 4) \implies (x^{i+1} = x^i)\Big]$$

to the time step of any BMC unfolding depth. However, using these constraints, BMC is not able to find another fixed path leading to ET_0 and thus, the procedure is rerun using the expanded enlarged target ET_1, containing l_1, l_2, and l_3. The new fixed path found by the SMT-solver is

$$p_1^{\text{fix}} = \langle l_0, x = 0\rangle \rightarrow \langle l_0, x = 5\rangle \rightarrow \text{ET}_1$$

Again, being in location l_0 with $x = 3$ there is a u-transition e^u which the BB can enforce to be taken. However, using ET_1, the pre-urgent inclusion check succeeds since $\langle l_0, 3 \leq x \leq 4\rangle$ is included in $Pre_d^u(\text{ET}_1) = \{\langle l_0, 3 \leq x \leq 4\rangle\}$. To put it in words, no matter whether the BB synchronises on a^u or not, the enlarged target can always be reached either using p_1^{fix} or e^u and, as a consequence, the unrealisability has been proven.

Theorem 1. *An error path π found by Algorithm 1 is valid for all possible BB implementations.*

Proof (sketch). π is based on fixed transitions and each u-transition along π is leading to ET. Assume that the SMC methods are correct [19], then ET contains only states which lead to states violating the property for all implementations of the BB. The BB cannot influence fixed transitions, and thus, the only way the BB can enforce the system to leave π is by synchronising over u-transitions. However, since the WB can force all u-transitions which are enabled along π into ET, a state violating the property is reached regardless of the behaviour of the BB, and thus, the property is unrealisable.

Thrm. 1 proves the soundness of the approach. Note that Algorithm 1 is only complete, if we assume that (1) the safety property does not contain disjunctions and (2) the BB has the ability to make different decisions depending on the state of the WB. For this the BB has to be able to read the state of the WB or to infer the state of the WB (e.g. by reading synchronization actions which are internal to the WB). (This assumption is implicitly made in classical controller synthesis approaches as well [4,11].)

4 Experiments

To evaluate our hybrid approach for the unrealisability proof in incomplete timed systems we combined the BMC tool `timemachine` [16,15] which uses Yices [10]

as underlying SMT solver and the SMC tool `fsmtMC` [20,19] which is based on LinAIGs [9,22,8] as the core data structure. We extended `timemachine` by the encoding of urgency and implemented the ET and PU routines using the methods provided by `fsmtMC`.

We use extended versions of the parameterizable *arbiter* [20,19] and *cpp* [19] benchmarks. The *arbiter* benchmark models n processes which try to access a shared resource and are controlled by an erroneous distributed arbiter and one counter module yielding $2n + 1$ components in total. We model all but two processes as a BB and prove the unrealisability of the property that the two processes cannot access the shared resource at the same time. The *cpp* benchmark models a ring of n parallel processes where each component communicates with its neighbours. We model $\frac{1}{2}n$ successive components as a BB and prove the unrealisability of the property that the first two components never enter an unsafe location at the same time.

We compare our hybrid approach with pure SMC. Since all tested benchmarks use urgent synchronisation *in their BB interface*, `timemachine` was not able to prove the unrealisability of any benchmark instance. It is neither possible to solve the unrealisability problem of our benchmark set using controller synthesis methods, since timed games are a special case of our scenario where the unknown controller has more power than the remaining components. However, we are able to modify the arbiter benchmark (by changing the power of the individual components within the network) making it possible to construct a semantically equivalent timed game. Using this modified benchmark, we are able to compare the results of our combined method to those obtained using the state-of-the-art controller synthesis tool UPPAAL-TIGA.

Table 1. Results

		fsmtMC	*hyVer*				
	TA	time	time	BMC	SMC	ET/CC	PU/C
orig. Arbiter	5	161.0	53.9	3	10	34/28	5/5
	10	2064.5	55.3	3	10	42/36	5/5
	12	5932.3	26.6	7	8	33/28	5/4
	13	-	32.0	7	8	32/27	5/4
	15	-	28.3	7	8	32/27	5/4
	25	-	75.4	3	10	40/34	5/5
	30	-	87.9	3	10	39/33	5/5
CPP	4	3.0	1.3	5	2	7/5	1/1
	5	-	2.4	5	2	7/5	1/1
	6	232.6	3.5	5	2	7/5	1/1
	7	-	5.5	5	2	7/5	1/1
	23	-	156.8	5	2	7/5	1/1
	24	-	372.1	5	2	6/4	1/1

To test our prototype implementation we used an AMD Opteron processor running on 2.3 GHz and having 64 GB RAM. We put a time limit of 2 CPU hours and a memory limit of 2 GB.

Table 1 compares the runtime of pure SMC (fsmtMC) with our hybrid approach (hyVer). The number of instantiated components of the respective benchmark is given in Column 1 (TA) followed by the runtime of fsmtMC in Column 2. Additionally to the runtime of hyVer (time), we report the number of forward steps (BMC) and backward steps (SMC) in order to successfully find an error path which proves the unrealisability. Furthermore, the table gives the number of ET inclusion checks and the resulting conflict clauses (ET/CC), analogously we give the number of PU inclusion checks and resulting constraints (PU/C).

The hybrid approach outperforms pure SMC for both the *orig. arbiter* and the *cpp* benchmark sets. Using SMC methods, the *arbiter* benchmark ran into timeouts for 13 processes and beyond, however `hyVer` was able to complete the verification task for up to 30 processes in a reasonable amount of time. A similar picture is valid for the *cpp* benchmark set. Here, SMC fails to prove unrealisability for instances having more than 6 processes whereas `hyVer` is able to complete the task within the given timeout for up to 24 processes.

In this setting, we used the negated property as the initial ET (that is, $\#\text{steps}_{\text{init}} = 0$). Later on, ET is always expanded by one discrete symbolic step followed by one continuous symbolic step (#steps = 2). max_fixed_paths is set to 1. We pick out three examples to explain the detailed results of our combined approach. First, consider the arbiter benchmark with 5 instantiated components. For this example, 5 fixed paths are discarded, since a continuous transition on the fixed path passes through a state in which a u-transition not leading to ET is enabled. The sixth candidate path is a valid error path, since it does not contain any state which is the source of a u-transition. In total, 28 ET conflict clauses are needed to find the candidate paths (6 ET inclusion checks are successful). The length of the valid path is the sum of the number of BMC steps and the number of SMC steps (in this example the length is 13). For the original arbiter benchmark having 12 instantiated components, the algorithm discards only 4 fixed paths and the fifth PU inclusion check is successful. In this case the final path contains a state which is the source of a u-transition, but using SMC it is proved that this transition leads into the ET.

For the cpp benchmark all paths found by BMC do not contain any states which are sources of a u-transition, i.e., the effect of u-transitions is handled by SMC only. Two additional SMC backward steps (discrete and continuous) are enough to find a valid fixed error path.

Table 2 shows the results of the modified arbiter benchmark when using fsmtMC, TIGA, and hyVer. Again, TA denotes the number of instantiated components, however, in this setting we vary the number of black boxed components (BB). Column TIGA reports the runtime of UPPAAL-TIGA, the remaining columns of the table are structured as before. We slightly changed the setting for this set of experiments.

Table 2. Modified Arbiter Benchmark

		fsmtMC	*TIGA*	*hyVer*				
TA	BB	time	time	time	BMC	SMC	ET/CC	PU/C
5	3	1052.1	0.1	1.0	0	12	1/0	0/0
5	2	-	0.2	1.4	3	8	2/1	0/0
5	1	-	4.1	2.8	3	12	2/1	0/0
6	4	1707.6	0.1	1.3	0	14	1/0	0/0
6	3	-	0.2	1.5	3	8	2/1	0/0
6	2	-	4.3	4.7	0	20	1/0	0/0
6	1	-	761.3	7.4	0	20	1/0	0/0
7	5	1502.9	0.1	2.5	0	22	1/0	0/0
7	4	-	0.2	2.2	0	16	1/0	0/0
7	3	-	4.4	2.4	5	10	11/10	0/0
7	2	-	901.9	25.5	5	8	28/26	1/1
7	1	-	-	149.6	5	10	12/12	1/1
8	6	2952.9	0.1	2.1	6	22	12/11	1/0
8	5	-	0.2	2.9	0	22	1/0	0/0
8	4	-	4.6	2.6	5	10	11/10	0/0
8	3	-	1010.1	41.9	7	8	194/192	1/1
8	2	-	-	127.7	3	10	125/123	1/1
8	1	-	-	1280.1	15	10	234/231	2/2

In particular, as a further optimisation in the initial target enlargement procedure, we only perform continuous steps, if performing discrete steps does not add new states to ET. In that way, for many instances the paths found by BMC do not contain any states which are sources of a u-transition. fsmtMC is only able to solve benchmarks where the maximum number of components (that is, all but 2) is abstracted. TIGA can solve more benchmarks than fsmtMC, however, the runtime increases dramatically for more than 4 WB components. hyVer is the only tool which is able to solve the whole benchmark set within the given timeout. It also completes the verification task in significant less time than TIGA for benchmarks which are solvable by both tools.

We also implemented a version of `hyVer` where ET is converted into an SMT formula and then directly connected to the BMC formula to avoid possibly multiple inclusion checks in order to generate one fixed error path. However, this version was not competitive compared to the procedure depicted in Algorithm 1 and justifies to perform inclusion checks and the usage of conflict clauses.

5 Conclusion

We presented a hybrid model checking algorithm to prove unrealisability for incomplete real-time systems. We use backward SMC methods to compute an enlarged target which we then try to hit using SMT-based forward BMC procedures. In order to accelerate the verification process we apply learning strategies and manipulate the enlarged target along the verification run to improve the overall accuracy. Our combined approach makes it possible to verify incomplete timed systems, which can neither be solved using pure BMC due to its inaccuracy nor using pure SMC due to the state space explosion problem. Finally, we showed the efficacy using parameterized incomplete timed benchmark sets. Our results show advantages compared to UPPAAL-TIGA when we consider timed games as a special case of the unrealisability problem.

References

1. Alur, R.: Timed automata. Theoretical Computer Science (1999)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science (1994)
3. Behrmann, G., Cougnard, A., David, R., Fleury, E., Larsen, K.G., Lime, D.: Uppaal tiga user-manual
4. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-tiga: Time for playing games! In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 121–125. Springer, Heidelberg (2007)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without bDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
6. Bouyer, P., Bozzelli, L., Chevalier, F.: Controller synthesis for MTL specifications. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 450–464. Springer, Heidelberg (2006)

7. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Form. Methods Syst. Des (2001)
8. Damm, W., Dierks, H., Disch, S., Hagemann, W., Pigorsch, F., Scholl, C., Waldmann, U., Wirtz, B.: Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. Sci. Comput. Program. (2012)
9. Damm, W., Disch, S., Hungar, H., Jacobs, S., Pang, J., Pigorsch, F., Scholl, C., Waldmann, U., Wirtz, B.: Exact state set representations in the verification of linear hybrid systems with large discrete state space. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 425–440. Springer, Heidelberg (2007)
10. Dutertre, B., de Moura, L.: The YICES SMT solver. Tech. rep., Computer Science Laboratory, SRI International (2006)
11. Ehlers, R., Mattmüller, R., Peter, H.-J.: Combining symbolic representations for solving timed games. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 107–121. Springer, Heidelberg (2010)
12. Faella, M., La Torre, S., Murano, A.: Automata-theoretic decision of timed games. In: Cortesi, A. (ed.) VMCAI 2002. LNCS, vol. 2294, pp. 94–108. Springer, Heidelberg (2002)
13. Faella, M., La Torre, S., Murano, A.: Dense real-time games. In: LICS (2002)
14. Maler, O., Pnueli, A., Sifakis, J.: On the synthesis of discrete controllers for timed systems (an extended abstract). In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 229–242. Springer, Heidelberg (1995)
15. Miller, C., Gitina, K., Becker, B.: Bounded model checking of incomplete real-time systems using quantified smt formulas. In: Proc. of MTV (2011)
16. Miller, C., Gitina, K., Scholl, C., Becker, B.: Bounded model checking of incomplete networks of timed automata. In: Proc. of MTV (2010)
17. Miller, C., Scholl, C., Becker, B.: Verifying incomplete networks of timed automata. In: Proc. of MBMV (2011)
18. Morbé, G., Scholl, C.: Fully symbolic tctl model checking for incomplete timed automata. In: Proc. of AVOCS (2013)
19. Morbé, G., Scholl, C.: Fully symbolic TCTL model checking for complete and incomplete real-time systems. Reports of SFB/TR 14 AVACS 104, SFB/TR 14 AVACS (September 2014), `http://www.avacs.org`
20. Morbé, G., Pigorsch, F., Scholl, C.: Fully symbolic model checking for timed automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 616–632. Springer, Heidelberg (2011)
21. Nopper, T., Miller, C., Lewis, M.D.T., Becker, B., Scholl, C.: Sat modulo bdd – a combined verification approach for incomplete designs. In: MBMV (2010)
22. Scholl, C., Disch, S., Pigorsch, F., Kupferschmid, S.: Computing optimized representations for non-convex polyhedra by detection and removal of redundant linear constraints. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 383–397. Springer, Heidelberg (2009)