Making the Right Cut in Model Checking Data-Intensive Timed Systems

Rüdiger Ehlers, Michael Gerke, and Hans-Jörg Peter

Reactive Systems Group
Saarland University
66123 Saarbrücken, Germany
{ehlers | gerke | peter}@cs.uni-saarland.de

Abstract. The success of industrial-scale model checkers such as UPPAAL [3] or NuSMV [12] relies on the efficiency of their respective symbolic state space representations. While difference bound matrices (DBMs) are effective for representing sets of clock values, binary decision diagrams (BDDs) can efficiently represent huge discrete state sets. In this paper, we introduce a simple general framework for combining both data structures, enabling a joint symbolic representation of the timed state sets in the reachability fixed point construction. In contrast to other approaches, our technique is robust against intricate interdependencies between clock constraints and the location information. Especially in the analysis of models with only few clocks, large constants, and a huge discrete state space (such as, e.g., data-intensive communication protocols), our technique turns out to be highly effective. Additionally, our framework allows to employ existing highly-optimized implementations for DBMs and BDDs without modifications. Using a prototype implementation, we are able to verify a central correctness property of the physical layer protocol of the FlexRay communication protocol [15] taking an unreliable physical layer into account.

1 Introduction

The verification of asynchronous protocols or *globally asynchronous locally synchronous* (GALS) hardware is a challenging task as it often requires dealing with intricate timing dependencies and huge state spaces at the same time. Manual correctness proofs are relatively hard to perform as special timing cases can be overlooked more easily than in the purely synchronous setting. Consequently, working with automated correctness provers, in particular by performing model checking, is the predominant approach carried out in this field. For keeping track of the timing correctly, a rich theory of timed automata [1] has been developed, which forms the theoretical foundation for model checking tools such as UPPAAL [3], KRONOS [23], or RED [21].

State-of-the-art model checking approaches for timed systems can broadly be classified into two categories: *semi-symbolic* approaches and *fully symbolic* approaches [19]. In semi-symbolic approaches, on the one hand, the discrete part of

the system under consideration is represented explicitly while clock valuations are lumped together into clock zones. These techniques are well-suited for systems with a small discrete state space. Fully symbolic approaches, on the other hand, represent *both* parts of the system in a symbolic way to lower the effect of state space explosion. For settings with a predominant control structure (such as, e.g., data-intensive communication protocols), this is broadly considered to be the more promising way to go as the discrete state space is often too large to be represented explicitly even with modern computers.

Fully symbolic timed model checking is a challenging problem that attracted a lot of interest during the last decade [9,16,4,21,6,19,22]. Some approaches rely on restricting the type of timing of the system in a way that it can be discretized more efficiently [6] or approximating the precise clock values to discrete time steps [9]. In both cases, reduced ordered binary decision diagrams (BDDs) [10,11] have been used as the uniform data structure for both time and the discrete part of the system. For such settings, it has been observed that the BDDs can blow-up significantly due to interdependencies in the timing behavior of the system [9], rendering this approach problematic.

Basically, this leaves us with two ways of solving this problem. The first one builds on using a different symbolic data structure like and-inverter graphs [17] or conjunctive normal form clause sets, thus avoiding this blow-up. Here, canonicity of the representation is renounced, rendering the application of the basic reachability fixed point algorithm difficult. The second way is based on keeping the data structures for clock zones and discrete state sets separate. One promising approach in this direction is to combine difference bound matrices (DBMs) [13] and BDDs. So far, to the best of our knowledge, this combination has only been used for *approximating* the set of reachable states [22].

In this paper, we recall the idea of decoupling the clock zone from the discrete state set representation. We use sets of pairs of DBMs and BDDs to represent reachable states in the classical fixed point computation. A timed state is contained in a state set if the set contains a DBM/BDD pair such that the state's clock valuation satisfies the DBM and the discrete part satisfies the BDD. To the best of our knowledge, this is the first application of this data structure with the standard fixed point algorithm.

Especially in the verification of asynchronous communication protocols, where the timing behavior is usually complicated but the number of distinct arising clock zones is small, our approach turns out to be very efficient since we benefit from the well-suitedness of DBMs for representing intricate timing constraints without cluttering the BDDs with timing dependencies. As the new approach allows the usage of the standard fixed point construction for finding the set of reachable states, it is simpler than the aforementioned techniques. The drawback of having some timed state potentially being present in more than one such pair is easily compensated by the additional possibility to perform a mixed breadth-first and depth-first search in this setting: by preferring discrete steps, we can forward the progress in exploring the discrete state space of the system in one

clock zone to successor clock zones, leading to faster termination of the model checking process.

As a proof of concept, we demonstrate the applicability of our technique by verifying the physical layer protocol of the FlexRay communication protocol [15] which follows the GALS paradigm: the setting can be described using only two clocks, each modeling the local pulsing in the discrete circuits of the sender and the receiver. Our approach is thus ideally suited for this interesting verification task as it exploits the model's restricted timing behavior, which enables the use of BDDs to tackle the huge arising discrete state space.

In Sect. 2, we begin our presentation with some preliminary definitions. Section 3 then describes the general approach, followed in Sect. 4 by an explanation of how example traces certifying the satisfaction of a reachability property are generated. Afterwards, we describe our FlexRay model which is then used for an experimental evaluation of the approach in Sect. 5. Finally, Sect. 6 concludes with an outlook on the possible evolution of this approach.

Related work. Developing fully symbolic timed state space representations has been an active field of research in the last decade. Møller et al. introduced *difference decision diagrams* (DDDs) [16], a BDD-like data structure in which each diagram node is labeled with a difference constraint. Here, the Boolean constraints, represented as special differences, are interleaved with the clock constraints in the diagram structure. Unfortunately, there is no implementation of their prototype model checker available. Based on DDDs, Behrmann et al. proposed *clock difference diagrams* (CDDs) [4] featuring deterministic interval-based branching at each node level. However, a combination of CDDs with BDDs enabling a fully symbolic state space exploration was only briefly discussed but has not been thoroughly investigated yet. As a further extension, Wang proposed *clock restriction diagrams* (CRDs) [21] where the branching decision depends on overlapping upper bounds and unrestricted constraints are omitted. Experiments with the CRD-based model checker RED suggest that the approach works well on standard timed automata benchmarks having many clocks and causing only a moderate discrete blow-up. However, in our experiments, RED runs out of memory on the FlexRay case study.

Based on *closed timed automata*, a restricted form of classical timed automata where only nonstrict clock constraints are allowed, Beyer introduced an integer semantics where clock values and location configurations can be represented jointly in a single BDD [6]. Similarly, Bozga et al. approximated the precise clock values to discrete time steps, also resulting in a pure discrete semantics allowing a state space representation using a single BDD [9]. Besides the loss of expressivity in the modeling of timed systems, in both approaches, it has been observed that the BDDs can blow-up significantly due to interdependencies in the timing behavior of the system.

Seshia and Bryant solved the TCTL model checking problem by representing sets of states by difference logic formulas which are, in turn, represented as BDDs using a binary encoding [19]. The clock differences that need to be tracked in the fixed-point computation are encoded in so-called transitivity constraints,

which are added on-the-fly during the model checking process. Even though they added some specialized optimizations for this case, the experimental results are inconclusive.

The idea of combining DBMs and BDDs was independently developed by Yamane and Nakamura [22] for implementing an *abstraction* technique proposed by Dill and Wong-Toi [14]. Our approach, however, uses DBM/BDD combinations for a fully symbolic state space representation in the *precise* computation of the reachable states of a timed system.

Previous correctness proofs of the physical layer protocol of the FlexRay communication protocol [15] were obtained in a deductive way. In this line of research, a fully reliable physical layer without any bit flips is assumed [7]. Our correctness proof, which is obtained via model checking, bases upon a more realistic setting taking an unreliable physical layer into account.

2 Preliminaries

2.1 Timed Systems

Timed Automata. The components of a timed system are represented by *timed automata*. A timed automaton [1] is a tuple $\mathcal{A} = (L, l_0, I, \Sigma, \Delta, X)$, where L is a finite set of (control) locations, $l_0 \in L$ is the initial location, $I : L \to \mathcal{C}(X)$ maps each location to an invariant, Σ is a finite set of actions, $\Delta \subseteq (L \times \Sigma \times \mathcal{C}(X) \times 2^X \times L)$ is a transition relation, X is a finite set of real valued clocks, and $\mathcal{C}(X)$ is the set of clock constraints over X. A clock constraint $\varphi \in \mathcal{C}(X)$ is of the form

$$\varphi = \mathbf{true} \mid x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2,$$

where x is a clock in X and c is a constant in $\mathbb{N}_0$. We say that a timed automaton is invariant-free if $I(l) = \mathbf{true}$ for all locations $l \in L$. A *clock valuation* $\boldsymbol{t} : X \to \mathbb{R}_{\geq 0}$ assigns a nonnegative value to each clock and can also be represented by a $|X|$-dimensional vector $\boldsymbol{t} \in \mathcal{R}$, where $\mathcal{R} = \mathbb{R}^X_{\geq 0}$ denotes the set of all clock valuations.

The states of a timed automaton are pairs $(l, \boldsymbol{t})$ of locations and clock valuations. Timed automata have two types of transitions: *timed transitions*, where only time passes and the location remains unchanged, and *discrete transitions*. In a timed transition, denoted by $(l, \boldsymbol{t}) \xrightarrow{a} (l, \boldsymbol{t} + a \cdot \mathbf{1})$, the same nonnegative value $a \in \mathbb{R}_{\geq 0}$ is added to all clocks such that, for each $0 \leq d \leq a$, $\boldsymbol{t} + d$ satisfies the location invariant $I(l)$. A discrete transition, denoted by $(l, \boldsymbol{t}) \xrightarrow{a} (l', \boldsymbol{t}')$ for some $a \in \Sigma$, is a transition $\delta = \langle l, a, \varphi, \lambda, l' \rangle$ of Δ such that $\boldsymbol{t}$ satisfies the clock constraint φ of δ, and $\boldsymbol{t}' = \boldsymbol{t}[\lambda := 0]$ is obtained from $\boldsymbol{t}$ by setting the clocks in λ to 0 and satisfies the location invariant $I(l')$.

We say that a finite sequence $a_1 \dots a_n \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$ of transitions is in the *language of* $\mathcal{A}$ ($a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$) if there is a path $s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_n$ such that for all $1 \leq i \leq n$, the individual $s_i = (l_i, \boldsymbol{t}_i)$ are states of the automaton, s_0

is an initial state (that is, l_0 is the initial location and $\boldsymbol{t}_0 = \mathbf{0}$ is the zero vector), and $s_{i-1} \xrightarrow{a_i} s_i$ are transitions of $\mathcal{A}$. We write $s_0 \longrightarrow^* s_n$ for the existence of a finite sequence $a_1 \dots a_n \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$ of transitions with $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$, and call a state s *reachable* iff there is an initial state s_0 with $s_0 \longrightarrow^* s$.

Composition. Timed automata can be composed to networks, in which the automata run in parallel and synchronize on shared actions. For two timed automata $\mathcal{A} = (L_1, l_0^1, I_1, \Sigma_1, \Delta_1, X_1)$ and $\mathcal{A}' = (L_2, l_0^2, I_2, \Sigma_2, \Delta_2, X_2)$ with disjoint clock sets $X_1 \cap X_2 = \emptyset$, the *parallel composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is the timed automaton $(L_1 \times L_2, (l_0^1, l_0^2), I, \Sigma_1 \cup \Sigma_2, \Delta, X_1 \cup X_2)$, where $I(l_1, l_2) = I_1(l_1) \wedge I_2(l_2)$ for all $l_1 \in L_1$ and $l_2 \in L_2$, and Δ is the smallest set that contains

- for $a \in \Sigma_1 \cap \Sigma_2$, $\langle (l_1, l_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (l_1', l_2') \rangle$ if $\langle l_1, a, \varphi_1, \lambda_1, l_1' \rangle \in \Delta_1$ and $\langle l_2, a, \varphi_2, \lambda_2, l_2' \rangle \in \Delta_2$,
- for $a \in \Sigma_1 \setminus \Sigma_2$, $\langle (l_1, l_2), a, \varphi_1, \lambda_1, (l_1', l_2) \rangle$ if $\langle l_1, a, \varphi_1, \lambda_1, l_1' \rangle \in \Delta_1$, and
- for $a \in \Sigma_2 \setminus \Sigma_1$, $\langle (l_1, l_2), a, \varphi_2, \lambda_2, (l_1, l_2') \rangle$ if $\langle l_2, a, \varphi_2, \lambda_2, l_2' \rangle \in \Delta_2$.

In the following, we only consider the *global timed automaton* that is obtained from the composition of the system's component automata. Note that control-related concepts such as synchronization, parallel composition, or integer variables are just technicalities in the construction of the symbolic discrete transition relation; they do not have to be considered in the actual model checking procedure.

Finite Semantics. The decidability of the reachability problem of timed automata relies on the existence of the *region equivalence relation* [1] on $\mathcal{R}$ which has a finite index.

For a timed automaton $\mathcal{A} = (L, l_0, I, \Sigma, \Delta, X)$, we call the value of a clock $x \in X$ *maximal* if it is strictly greater than the highest constant c_{max} any clock is compared to.[1] We say that two clock valuations $\boldsymbol{t}_1, \boldsymbol{t}_2 \in \mathcal{R}$ are in the same *clock region*, denoted $\boldsymbol{t}_1 \sim_R \boldsymbol{t}_2$, if

- the set of clocks with maximal value is the same in $\boldsymbol{t}_1$ and in $\boldsymbol{t}_2$ ($\forall x \in X : \boldsymbol{t}_1(x) > c_{max} \Leftrightarrow \boldsymbol{t}_2(x) > c_{max}$), and
- $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ agree (1) on the integer parts of the clock values, (2) on the relative order of the noninteger parts of the clock values, and (3) on the equality of the noninteger parts of the clock values with 0. That is, for all clocks x and y with nonmaximal value, it holds that (1) $\lfloor \boldsymbol{t}_1(x) \rfloor = \lfloor \boldsymbol{t}_2(x) \rfloor$, (2) $\widehat{\boldsymbol{t}}_1(x) \leq \widehat{\boldsymbol{t}}_1(y) \Leftrightarrow \widehat{\boldsymbol{t}}_2(x) \leq \widehat{\boldsymbol{t}}_2(y)$, and (3) $\widehat{\boldsymbol{t}}_1(x) = 0$ if, and only if, $\widehat{\boldsymbol{t}}_2(x) = 0$, where $\widehat{\boldsymbol{t}}_i(x) = \boldsymbol{t}_i(x) - \lfloor \boldsymbol{t}_i(x) \rfloor$ for $i \in \{1, 2\}$.

We denote with $[\boldsymbol{t}]_R = \{\boldsymbol{t}' \in \mathcal{R} \mid \boldsymbol{t} \sim_R \boldsymbol{t}'\}$ the clock region $\boldsymbol{t}$ belongs to. We say that two states $s_1 = (l_1, \boldsymbol{t}_1)$ and $s_2 = (l_2, \boldsymbol{t}_2)$ of $\mathcal{A}$ are *region-equivalent*, denoted by $s_1 \sim_R s_2$, if their locations are the same ($l_1 = l_2$) and the clock valuations are in the same clock region ($\boldsymbol{t}_1 \sim_R \boldsymbol{t}_2$), and denote with $[(l, \boldsymbol{t})]_R = \{(l, \boldsymbol{t}') \in L \times \mathcal{R} \mid \boldsymbol{t} \sim_R \boldsymbol{t}'\}$ the equivalence class of region-equivalent states that $(l, \boldsymbol{t})$ belongs to.

[1] c_{max} is sometimes called the clock ceiling.

Regions are a suitable semantics for the abstraction of timed automata because they essentially preserve the language: if there is a discrete transition $s \xrightarrow{a} s'$ from a state s to a state s' of a timed automaton, then there is, for all states r with $r \sim_R s$, a state r' with $r' \sim_R s'$ such that $r \xrightarrow{a} r'$ is a discrete transition with the same label. For timed transitions, a slightly weaker property holds: if there is a timed transition $s \xrightarrow{t} s'$ from a state s to a state s', then there is, for all states r with $r \sim_R s$, a state r' with $r' \sim_R s'$ such that there is a timed transition $r \xrightarrow{t'} r'$ (but possibly with $t' \neq t$).

The *finite semantics* of a timed automaton $\mathcal{A} = (L, l_0, I, \Sigma, \Delta, X)$ is the finite graph $[\![\mathcal{A}]\!] = (S, s_0, T)$ where

- the symbolic state set $S = \{[(l, \boldsymbol{t})]_R \mid (l, \boldsymbol{t}) \in L \times \mathcal{R}\}$ of $[\![\mathcal{A}]\!]$ is the set of equivalence classes of region-equivalent states of $\mathcal{A}$, with
- the initial state $s_0 = [(l_0, \boldsymbol{t}_0)]_R$, and
- the set $T = \{(s, s') \in S \times S \mid \exists r \in s,\ r' \in s', a \in \Sigma \cup \mathbb{R}_{\geq 0}.\, r \xrightarrow{a} r'\}$ of transitions.

The finite semantics is reachability-preserving:

Lemma 1. *[1] For a timed automaton $\mathcal{A} = (L, l_0, I, \Sigma, \Delta, X)$ there is a finite path from a state $(l, \boldsymbol{t})$ to a state $(l', \boldsymbol{t}')$ if, and only if, there is a finite path from $\big[(l, \boldsymbol{t})\big]_R$ to $\big[(l', \boldsymbol{t}')\big]_R$ in $[\![\mathcal{A}]\!]$.*

Clock Zones. A coarser finite representation of $\mathcal{R}$ can be obtained by considering *clock zones*. A clock zone z is represented by a conjunction of clock difference constraints of the form $x - y \prec_{x,y} c_{x,y}$, where $x, y \in X \cup \{x_0\}$, for an $x_0 \notin X$, $\prec_{x,y} \in \{\leq, <\}$, and $c_{x,y} \in \mathbb{Z} \cup \{\infty\}$. A clock valuation $\boldsymbol{t}$ satisfies z, written as $\boldsymbol{t} \in z$, if $\boldsymbol{t}' = \boldsymbol{t} \cup \{x_0 \mapsto 0\}$ satisfies each constraint $x - y \prec_{x,y} c_{x,y}$ from z: $\boldsymbol{t}'(x) - \boldsymbol{t}'(y) \prec_{x,y} c_{x,y}$. We define $\mathcal{Z}$ as the set of all clock zones.

A data structure for representing clock zones are difference bound matrices (DBMs) [13], which allow, for two clock zones $z, z' \in \mathcal{Z}$, an efficient implementation of the operations (1) intersection $z \wedge z'$, (2) clock reset $z[\lambda := 0]$, and (3) elapsing of time $z^{\Uparrow}$ (see [5] for an overview). Note that, in order to ensure termination of the forward analysis, we implicitly apply a maximal constant extrapolation after executing a time elapse. We denote z_0 as the clock zone that only comprises the initial clock valuation $\mathbf{0}$.

2.2 Binary Decision Diagrams

For representing sets of locations symbolically we use *reduced ordered binary decision diagrams* (BDDs) [10,11], which represent functions $f : 2^V \to \mathbb{B}$ for some finite set of variables V. Since they are well-established in the context of formal verification, we do not describe their details here but rather treat them on an abstract level and only state the important operations (see [11] for an overview). Given two BDDs (or more generally, two binary functions, abbreviated as BF) f and f', we define their conjunction and disjunction as

Algorithm 1 Least fixed point construction for computing the set of reachable states R.

$R_0 := \{\text{initial states}\}$
$i := 0$
repeat
  $i := i + 1$
  $R_i := R_{i-1} \cup \mathsf{post}(R_{i-1})$
until $R_i = R_{i-1}$
$R := R_i$

$(f \wedge f')(x) = f(x) \wedge f'(x)$ and $(f \vee f')(x) = f(x) \vee f'(x)$ for all $x \subseteq V$. The negation of a BF is defined similarly. Given some set of variables $V' \subseteq V$ and a BF f, we define $\exists V'.f$ as the function that maps all $x \subseteq V$ to **true** for which there exists some $x' \subseteq V'$ such that $f(x' \cup (x \setminus V')) = \mathbf{true}$.

2.3 Reachability Model Checking

Model checking reachability properties is carried out by computing the set of reachable states and testing whether some goal state is contained in this set. In this paper, w.l.o.g., we only consider properties of the form $\exists\Diamond\phi$, that is, "is there an execution of the system such that ϕ is eventually reached", where ϕ is a Boolean state predicate defining the goal states. The classical fixed point construction for this task is given in Algorithm 1. It relies on the existence of an efficiently computable post operator for computing all successor states of a given set of states.

When using BDDs for storing state sets in this algorithm, usually it is beneficial to pre-compute a Boolean encoding of the *transition relation* of the system for usage in the post operator. It contains precisely the pairs of states (s, s') for which there exists a transition from s to s'. For a comprehensive overview of building such a relation and using it in the post operator, see [2].

3 Fully Symbolic Real-Time Model Checking

In this section, we present the basic building blocks of our approach, namely the timed state set representation that is used and how the basic fixed point algorithm for computing the set of reachable states can be extended in order to be applicable to this representation. For a clear separation of concerns, we describe our representation in a general way and abstract from the actual choices of data structures for representing clock zones (CZ) and discrete location sets (LS). While for our actual implementation of the approach (as described in Sect. 5), DBMs and BDDs are used, the general idea is applicable to all suitable data structure types (such as, e.g., CDDs [4]). Thus, future alternatives for storing clock zones and location sets can also be used with our approach.

We start with a presentation for invariant-free timed automata and demonstrate the application of our algorithm on an example network. We then show

how to extend the idea to include support for invariants. The section closes with some remarks on optimizations to the algorithm.

The starting point for our approach are sets of CZ/LS pairs which permit representing the timed and discrete parts of sets of states separately. That means, sets of states $\mathcal{S}$ in the fixed point computations are defined as partial functions $\mathcal{S} : \mathcal{Z} \rightharpoonup 2^L$. For such a so-called *clock zone map* (CZM), a state $(l, \boldsymbol{t})$ is contained in S if for some $z \in \mathcal{Z}$, $\boldsymbol{t} \in z$ and $l \in \mathcal{S}(z)$. Note that we do not require the choice of z to be unique.

3.1 Computing the Reachable States using CZMs

In the following, we describe how to adapt the basic fixed point construction for computing the set of reachable states given in Algorithm 1 to work with CZMs. The first step is to partition the overall transition relation of the system: for each combination of clock guards and resets that occurs along some transition, we build a separate transition relation containing all transitions corresponding to the guard/reset pair. This step separates timing concerns from the discrete transitions of the system and makes it easy to compute successor clock zones from a given source clock zone and some guard/reset pair. Note that, when using BDDs for representing location sets, it is *not* necessary to enumerate the product locations in the global timed automaton explicitly if the system is given as a network of timed automata, as the synchronization between the components can be encoded *symbolically*.

After building the transition relations, the usual fixed point computation is performed, with the small modification of iterating over all such guard/reset pairs in every step. After each discrete transition, we also compute the set of possible timed transitions that can follow in order to obtain the successor clock zone. Algorithm 2 shows the details of these steps.

In each round, the algorithm iterates over the set of reachable states contained in the respective previous pre-fixed point (stored in R). For every clock zone in the domain of R, it computes successor locations L and clock zones z' for each guard/reset pair (φ, λ) in the transition relation. Then, we check if the new CZ/LS pair (z', L) is already contained in the pre-fixed point. If this is not the case, it is added to the next pre-fixed point R'. For a more efficient computation, we furthermore track changes in the CZM in a special waiting set W in order to avoid re-considering CZ/LS pairs which have not changed since the previous round of the algorithm. The computational burden of the added inner loop in which all guard/reset pairs are iterated over is also weakened by the fact that unlike for models checkers keeping the discrete part of the system explicit, this setting allows the computation of timed successor zones for many locations at the same time.

Since the algorithm presented is essentially equivalent to the classical reachability fixed point algorithm, its correctness is guaranteed. Indeed, for every $n \in \mathbb{N}$ and timed state $(l, \boldsymbol{t})$ that is reachable from the initial state in n discrete steps, the set R contains this state after at most n iterations of computing the

Algorithm 2 Computing the set of reachable states R represented as a CZM. The post operator is parametrized by the transition relation used.

```
 1: for all guard/reset pairs (φ, λ) in the system do
 2:    compute the transition relation T[φ, λ]
 3: end for
 4: R := {z_0^⇑ ↦ {l_0}}
 5: W := {z_0^⇑}
 6: R' := R
 7: repeat
 8:    R := R'
 9:    W' := ∅
10:    for all z ∈ W do
11:       for all guard/reset pairs (φ, λ) do
12:          L := post_{T[φ,λ]}(R[z])
13:          z' := (z ∧ φ)[λ := 0]^⇑
14:          if R'[z'] ⋢ L then
15:             R'[z'] := R'[z'] ∪ L
16:             W' := W' ∪ {z'}
17:          end if
18:       end for
19:    end for
20:    W := W'
21: until R = R'
```

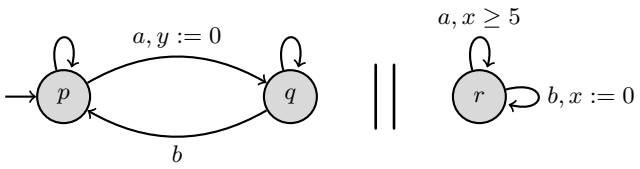

Fig. 1. An example network of timed automata

pre-fixed points. Note that the termination of this algorithm is also guaranteed as clock regions are never split and the number of sets of these is finite.

3.2 An Example

Consider the parallel composition of the timed automata depicted in Figure 1. The product automaton has two locations pr and qr, and two clocks x and y. There are three guard/reset pairs $(\mathbf{true}, \emptyset)$, $(x \geq 5, \{y\})$, and $(\mathbf{true}, \{x\})$, leading to three transition relations

- $T[\mathbf{true}, \emptyset] = \{(qr, qr), (pr, pr)\}$,
- $T[x \geq 5, \{y\}] = \{(pr, qr)\}$, and
- $T[\mathbf{true}, \{x\}] = \{(qr, pr)\}$.

Algorithm 3 Replacement for lines 13–17 to the inner loop of Algorithm 2 to allow handling invariants.

1: **for all** $i \in \mathcal{I}$ **do**
2: $L' := L \cap C(i)$
3: $z' := \big((z \wedge \varphi)[\lambda := 0] \wedge i\big)^{\Uparrow} \wedge i$
4: **if** $R'[z'] \not\supseteq L'$ **then**
5: $R'[z'] := R'[z'] \cup L'$
6: $W' := W' \cup \{z'\}$
7: **end if**
8: **end for**

When running the algorithm on this example, we initialize $R = \{(x = y = 0)^{\Uparrow} \mapsto \{pr\}\} = \{(x = y) \mapsto \{pr\}\}$ and $W = \{(x = y)\}$. Then, in the fixed point computation, we iterate over the transition relations and obtain $R = \{\{(x = y) \mapsto \{pr\}, (x \geq 5 \wedge y \leq x - 5) \mapsto \{qr\}\}$ and $W = \{(x \geq 5 \wedge y \leq x - 5)\}$. In the second round, we add $(x \leq y) \mapsto \{pr\}$ to R and obtain $W = \{(x \leq y)\}$. The algorithm then terminates in the following round, leaving us with $R = \big\{\{(x = y) \mapsto \{pr\}, (x \geq 5 \wedge y \leq x - 5) \mapsto \{qr\}, (x \leq y) \mapsto \{pr\}\big\}$ as the CZM representation of the set of reachable states in the system.

3.3 Handling Invariants in CZMs

To incorporate invariants in our fixed point construction, it is necessary to infer them from the location information. Here, we exploit the fact that in our models, the number of clocks and the number of distinct invariants is small. Hence, it is feasible to enumerate all possible invariants of the product timed automaton as a precomputational step.

Let $\mathcal{I}$ be the set of all invariants appearing in the product automaton of a timed system. We assume that each invariant is given in minimal form. For example, if precisely the invariants $x \leq 3$, $x \leq 4$, and $y \leq 2$ are associated to some (but not all) locations of three different components of the input network, we obtain $\mathcal{I} = \{\mathbf{true}, x \leq 3, x \leq 4, y \leq 2, x \leq 3 \wedge y \leq 2, x \leq 4 \wedge y \leq 2\}$. Furthermore, we also introduce a function $C : \mathcal{I} \to 2^L$ that maps each invariant onto the set of locations in which precisely the given invariant must hold. Note that one can easily compute $\mathcal{I}$ and C in a preprocessing step *without* constructing the product automaton.

Now, in our fixed point construction, we need to split the computed successor locations according to the mapped locations in C and compute the successor clock zones taking into account the respective invariants. Algorithm 3 contains the necessary changes to Algorithm 2. If the initial location has an active invariant, it also needs to be taken into account when computing the initial zone. Hence, we also change line 4 of Algorithm 2 to $R := \big\{z_0^{\Uparrow} \wedge I(l_0) \mapsto \{l_0\}\big\}$ and line 5 to $W := \{z_0^{\Uparrow} \wedge I(l_0)\}$.

3.4 Improving the Performance of the Approach

As an optimization of the presented technique, we propose to deviate from the strict rule of computing one pre-fixed point after the other. By storing all newly encountered CZ/LS pairs (z, l) directly into the pre-fixed point R in the algorithm instead of R' (which is copied to R after all elements from the waiting set have been processed), computation time is saved in the case that z is in the waiting set W but not yet processed in the respective round of the fixed point computation. This can easily be seen from that fact that in such a case, the consideration of the newly reachable timed states in not delayed to the next round, resulting in a lower number of steps in total until the fixed point is reached.

Note that this also allows us to use a waiting *queue* instead of a list. Then, in line 10 of Algorithm 2, we *pop* zones (i.e., we remove every zone from W that was picked). Additionally, we modify the waiting queue such that clock zones are drawn from it prioritized by their first appearance in R. This way, the exploration of new clock zones is delayed such that the progress in computing the reachable discrete states for zones encountered earlier can be forwarded to successor zones more efficiently.

4 Guided Counter-Example Generation

The algorithm depicted so far is only capable of computing the set of reachable states. As checking if it contains some given goal state is trivial after it has been computed, this suffices to make the approach presented suitable for a typical verification task for timed systems: checking that no error state is reachable.

For cases in which some error state is reachable, however, obtaining a sequence of transitions from the initial state to some error state is desired as it helps the designer of a timed system to improve the model. Therefore, most modern model checking tools can generate such *counter-examples*. This is also possible with the improved version (see Sect. 3.4) of our fixed point construction as we explain in the following.

Suppose that our *improved* fixed point construction terminates early with a set of *forward* reachable states R comprising some error states E. Then, we execute a *nonimproved* fixed point construction to compute a sequence of pre-fixed points $F_0, \ldots, F_n$, where each F_i is restricted to R and $E_f = F_n \cap E \neq \emptyset$. Note that for all $0 \leq i \leq n$, the set F_i contains only states that are reachable after exactly i steps.

Now, we can compute a counter-example using a *backward nonimproved* fixed point construction producing a sequence of backward reachable sets of states $B_n, \ldots, B_0$ with $B_n = E_f$. For each $0 < j \leq n$, we compute B_{j-1} by picking one particular semi-symbolic state (i.e., a zone and one concrete location) from B_j, compute its predecessors, and restrict them to F_{j-1}. After each iteration j, we pick some transition connecting a state in B_{j-1} and B_j, and add it to the counter-example. Note that the computation of the predecessors can be done in a symbolic way using our technique (the adaption to the backward case is straight-forward).

5 Experimental Results

5.1 Prototype Implementation

We implemented our approach in a prototype model checker using the UPPAAL-DBM library [5] for representing DBMs and the CUDD library [20] for representing BDDs. To allow a fair comparison with UPPAAL [3] and RED [21], our tool reads automata-based specifications as input. The first step in its execution is to call the tool NOVA from the SIS toolset [18] as a back-end for finding efficient assignments of control locations to BDD variable valuations. Then, the guard/reset pairs of the given timed system are collected and, for each pair, the BDD representing the symbolic transition relation over the discrete control structure is computed (using the assignments obtained in the first step). In the last step of the preparation phase, the possible invariant combinations are collected and, for each combination, the BDD representing the associated locations is computed.

The actual fixed point computation of the reachable states is implemented as described in Sect. 3. For the state space representation, we use a hash map that maps DBMs to BDDs. We do not provide a fixed BDD variable ordering a priori or use any other insight into the model to optimize the BDD representation. Instead, our implementation only relies on the automatic on-the-fly reordering heuristics implemented in the CUDD library.

5.2 The FlexRay Communication Protocol

The emergence of drive-by-wire and the need for high bandwidths in the design of automotive electronics calls for a communication protocol that is both fast and highly reliable. The recently developed FlexRay protocol [15] represents a state-of-the-art industrial X-by-wire communication protocol that is used in many modern cars. Its purpose is to enable reliable communication between the various *electronic control units* (ECUs) that are connected by a bus. In our case study, we investigate the critical physical layer protocol of the FlexRay protocol, where a message is transmitted during a so-called *static segment* from a sending ECU to a receiving ECU. As a crucial correctness property, it is required that there is no deviation of the message received from the message sent. In the following, we explain the important details of [15] which are reflected in our model.

Clocks. Since the receiving and sending ECU are running asynchronously, we introduce two clocks to model the timing behavior. The length of a clock cycle may deviate by at most 0.15 % from the standard rate.

Bit Stream Format. The actual payload of a transmission between two ECUs has a maximal length of 262 bytes. It is embedded into a structured bit stream that consists of (1) the initial *transmission start sequence* (TSS), (2) the *frame start sequence* (FSS), (3) the individual bytes of the payload, each prepended with a *byte start sequence* (BSS), and finally (4) the *frame end sequence* (FES). Thus, the maximal bit stream length is 2638 bits.

		CZM model checker				Uppaal	
Payload	Correct	Steps	Zones	Time	Memory	Time	Memory
1	Yes	6566	1858	86 s	252 MB	23 s	88 MB
2	Yes	8606	2498	2 min	251 MB	69 s	205 MB
3	Yes	10423	3142	7 min	527 MB	2 min	325 MB
4	Yes	12143	3782	2 min	251 MB	3 min	436 MB
5	Yes	13863	4422	4 min	312 MB	4 min	563 MB
20	Yes	45029	14038	5 min	261 MB	18 min	2 GB
21	Yes	46750	14678	6 min	259 MB	18 min	2 GB
22	Yes	48470	15318	5 min	262 MB	19 min	2 GB
23	Yes	50190	15958	4 min	259 MB	20 min	3 GB
24	Yes	51991	16024	7 min	264 MB	22 min	3 GB
25	Yes	52629	16024	5 min	314 MB	22 min	3 GB
31	Yes	54309	16024	16 min	415 MB	29 min	4 GB
32	Yes	54589	16024	6 min	313 MB	31 min	4 GB
33	Yes	54869	16024	28 min	955 MB	31 min	4 GB
34	Yes	55149	16024	13 min	313 MB	MEMOUT	
60	Yes	66230	16024	18 min	520 MB	MEMOUT	
100	Yes	90230	16024	57 min	941 MB	MEMOUT	
150	Yes	120230	16024	30 min	406 MB	MEMOUT	
200	Yes	150230	16024	72 min	938 MB	MEMOUT	
262	Yes	187430	16024	28 min	413 MB	MEMOUT	

Table 1. Comparison of our prototype with Uppaal on the FlexRay physical layer protocol case study. The first column shows the length of the payload in bytes. The second column states the obtained verification result. The next four columns show the number of symbolic steps (i.e., applications of the post operator) until the reachability fixed point is reached, the number of distinct clock zones encountered, the running time, and the memory consumption of our prototype model checker. The last two columns show the running time and space consumption of Uppaal. All benchmarks were executed on an AMD Opteron processor with 2.6 GHz and 4 GB RAM.

Redundancy and Error Model. Each bit of the bit stream is fed to the bus in 8 consecutive clock cycles. As a reasonable error model, we assume that in any sequence of 5 consecutive bits on the bus, 1 bit might be flipped.

Voting. In order to compensate for the flipped bits, the receiver determines the *voted value* over the last 5 received bits (i.e., high for 3 or more high bits, low otherwise).

Strobing and Bit Clock Alignment. In order to compensate for the clock drifts, the receiver uses a counter to *strobe* the 5^{th} out of 8 voted values. The received stream consists of the sequence of strobed values. The strobe counter is realigned at the start of the TSS or during a BSS.

	CZM model checker				Uppaal		RED		
Benchmark	Steps	Zones	Time	Memory	Time	Memory	Steps	Time	Mem
Fischer 5	3156	1496	1 s	108 MB	0 s	37 MB	5	0 s	21 MB
Fischer 6	42528	17426	32 s	156 MB	0 s	37 MB	5	1 s	45 MB
Fischer 7	612531	227522	17 min	302 MB	1 s	37 MB	5	1 s	66 MB
Fischer 8	TIMEOUT				3 s	38 MB	5	3 s	105 MB
Fischer 9	TIMEOUT				15 s	42 MB	5	8 s	174 MB

Table 2. Comparison of our prototype with Uppaal and RED on the (timing-intensive) Fischer protocol benchmark.

5.3 Model Checking FlexRay

We modeled the physical layer protocol of the FlexRay protocol [15] as a network of timed automata[2] for usage with Uppaal[3] [3], RED[4] [21], and our prototype model checker. As a safety property, we check the reachability of a dedicated error location which the receiver enters upon an uncompensatable deviation of the received from the sent bit stream. Table 1 shows the results of our evaluation. Unfortunately, for every payload length, RED runs out of memory (e.g., for the smallest instance it hits the 4 GB limit after 18 minutes).

The most striking observation is that our prototype overall needs much less memory than Uppaal or RED, which allows us to verify the full payload length of 262 bytes. In fact, while our model checker's memory consumption always stays below 1 GB, Uppaal's memory and time consumption increases linearly in the length of the payload, resulting in running out of memory with a payload length of 34 bytes or more. It is also noteworthy that in most of the cases our approach also outperforms Uppaal w.r.t. the running time. An oscillation effect can be observed in the running times and space consumptions of our implementation which is caused by the variable reordering and caching heuristics of the Cudd library. This BDD-related phenomenon is also observable in other contexts (see, e.g., [8]). Nevertheless, the number of symbolic exploration steps increases linearly in the length of the payload and the set of encountered clock zones reaches its fixed point at a payload length of 24.

5.4 Model Checking Fischer

In addition to the FlexRay case study from Sect. 5.3, we also considered the Fischer mutual exclusion protocol, a standard benchmark from the timed model checking domain with a small discrete state space and one clock per component. Table 2 shows that the existing model checking techniques implemented in Uppaal and RED perform better than our prototype on this benchmark.

[2] The models are available at http://www.avacs.org/Benchmarks/Open/flexray.tgz
[3] Version 4.0.11, running with aggressive space optimization – option -S2
[4] Version 8.100511

This is however not surprising as the Fischer protocol does not fall into the class of systems whose verification our approach aims at. We presented a specialized technique for timed systems with a large discrete state space but only a few clocks, an important class of models that comprise, e.g., data-intensive asynchronous communication protocols. The Fischer protocol model, on the other hand, has a large number of clocks (one per component), but only few locations, thus the standard semi-symbolic state space representation used in UPPAAL is already quite effective here. Also, RED's symmetry reduction is beneficial for this particular protocol.

6 Conclusion and Outlook

DBMs and BDDs impressively demonstrate their effectiveness in model checkers such as UPPAAL and NUSMV. However, since NUSMV can only handle pure discrete models and UPPAAL does not have a symbolic representation for the discrete part of the state space, both tools fail in verifying timed systems with large discrete control structures.

This paper presented a fully symbolic approach to timed model checking that combines DBMs with BDDs. In contrast to other approaches, our technique neither suffers from a loss of modeling precision (we remain in the classical timed automata framework) nor leads to blow-ups in the BDDs (we avoid the encoding of timing interdependencies in the BDDs).

Inspired by the encouraging experimental results, in future work, we plan to extend the scope of our approach to arbitrary timed systems. A promising direction is to investigate efficient representations of sets of pairs of DBMs and BDDs. So far, our prototype uses a simple hash map for assigning *complete* DBMs to BDDs. However, for many problem instances, considering *partial* DBMs might be more appropriate as it gives more flexibility in finding efficient representations.

Acknowledgment. This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2) (1994) 183–235
2. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In Bernardo, M., Corradini, F., eds.: SFM. Volume 3185 of Lecture Notes in Computer Science., Springer (2004) 200–236
4. Behrmann, G., Larsen, K.G., Pearson, J., Weise, C., Yi, W.: Efficient timed reachability analysis using clock difference diagrams. In Halbwachs, N., Peled, D., eds.: CAV. Volume 1633 of Lecture Notes in Computer Science., Springer (1999) 341–353
5. Bengtsson, J.: Clocks, DBM, and States in Timed Systems. PhD thesis, Uppsala University (2002)

6. Beyer, D.: Improvements in BDD-based reachability analysis of timed automata. In Oliveira, J.N., Zave, P., eds.: FME. Volume 2021 of Lecture Notes in Computer Science., Springer (2001) 318–343
7. Beyer, S., Böhm, P., Gerke, M., Hillebrand, M.A., der Rieden, T.I., Knapp, S., Leinenbach, D., Paul, W.J.: Towards the formal verification of lower system layers in automotive systems. In: ICCD, IEEE Computer Society (2005) 317–326
8. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: Hardware from psl. Electr. Notes Theor. Comput. Sci. **190**(4) (2007) 3–16
9. Bozga, M., Maler, O., Pnueli, A., Yovine, S.: Some progress in the symbolic verification of timed automata. In Grumberg, O., ed.: CAV. Volume 1254 of Lecture Notes in Computer Science., Springer (1997) 179–190
10. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986) 677–691
11. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2) (1992) 142–170
12. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NuSMV: A new symbolic model checker. STTT **2**(4) (2000) 410–425
13. Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In Sifakis, J., ed.: Automatic Verification Methods for Finite State Systems. Volume 407 of Lecture Notes in Computer Science., Springer (1989) 197–212
14. Dill, D.L., Wong-Toi, H.: Verification of real-time systems by successive over and under approximation. In Wolper, P., ed.: CAV. Volume 939 of Lecture Notes in Computer Science., Springer (1995) 409–422
15. FlexRay Consortium: FlexRay Communications System Protocol Specification Version 2.1 Revision A. (2005)
16. Møller, J.B., Lichtenberg, J., Andersen, H.R., Hulgaard, H.: Fully symbolic model checking of timed systems using difference decision diagrams. Electr. Notes Theor. Comput. Sci. **23**(2) (1999)
17. Pigorsch, F., Scholl, C., Disch, S.: Advanced unbounded model checking based on AIGs, BDD sweeping, and quantifier scheduling. In: FMCAD, IEEE Computer Society (2006) 89–96
18. Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, EECS Department, University of California, Berkeley (1992)
19. Seshia, S.A., Bryant, R.E.: Unbounded, fully symbolic model checking of timed automata using boolean methods. In Jr., W.A.H., Somenzi, F., eds.: CAV. Volume 2725 of Lecture Notes in Computer Science., Springer (2003) 154–166
20. Somenzi, F.: CUDD: CU Decision Diagram package release 2.4.2 (2009)
21. Wang, F.: Efficient verification of timed automata with BDD-like data structures. STTT **6**(1) (2004) 77–97
22. Yamane, S., Nakamura, K.: Development and evaluation of symbolic model checker based on approximation for real-time systems. Systems and Computers in Japan **35**(10) (2004) 83–101
23. Yovine, S.: Kronos: A verification tool for real-time systems. STTT **1**(1-2) (1997) 123–133