Fully Symbolic Timed Model Checking using Constraint Matrix Diagrams

Rüdiger Ehlers Daniel Fass Michael Gerke Hans-Jörg Peter
Reactive Systems Group
Saarland University
66123 Saarbrücken, Germany
E-mail: {ehlers | fass | gerke | peter}@cs.uni-saarland.de

Abstract

We present constraint matrix diagrams (CMDs), a novel data structure for the fully symbolic reachability analysis of timed automata. CMDs combine matrix-based and diagram-based state space representations generalizing the concepts of difference bound matrices (DBMs), clock difference diagrams (CDDs), and clock restriction diagrams (CRDs). The key idea is to represent convex parts of the state space as (partial) DBMs which are, in turn, organized in a CDD/CRD-like ordered and reduced diagram. The location information is incorporated as a special Boolean constraint in the matrices. We describe all CMD operations needed for the construction of the transition relation and the reachability fixed point computation. Based on a prototype implementation, we compare our technique with the timed model checkers RED *and* Uppaal*, and furthermore investigate the impact of two different reduced forms on the time and space consumption.*

1 Introduction

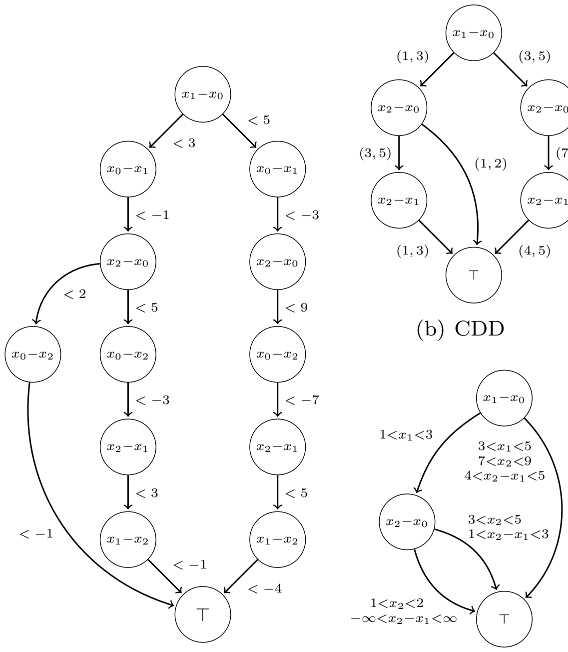
A promising approach for automatically establishing the correctness of real-time systems is *model checking* of timed automata [2, 3, 1], in which an exhaustive state space exploration is performed to check if the system can ever transition into a state in which the given specification is violated.

The set of reachable states is usually constructed using a fixed point computation, which is performed either in a *forward* or *backward* manner. In the former case, states that are reachable from the respective prefixed point (starting with the initial state) are added to the next prefixed point until no more new states can be added (or a reachable error state is found). In the backward construction, the same process is started from the set of error states until a fixed point is reached or the initial state is found to be backward reachable. In both cases, the prefixed point needs to be represented using a suitable data structure.

State-of-the-art model checking techniques for timed systems can broadly be classified into two categories: *semi-symbolic* approaches and *fully symbolic* approaches [17, 22]. 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, as done in the model checker Uppaal [5]. These techniques are well-suited for systems with a small discrete state space. Fully symbolic approaches, on the other hand, represent *both* timing and location information in a symbolic way to lower the effect of state space explosion, as done in the model checker RED [24].

Consequently, the development of suitable data structures for the efficient representation of prefixed points of reachable states has been an active field of research in the last years. While the potential of approaches such as *clock restriction diagrams* (CRDs) [24] or *clock difference diagrams* (CDDs) [18, 6] has been shown in previous papers, the success of *binary decision diagrams* (BDDs) [10, 11], which greatly increased the size of the systems that can be handled by discrete model checkers [19], could not be reproduced to its full extent in the case of timed model checking so far.

In this paper, we present progress towards closing this gap by introducing *constraint matrix diagrams* (CMDs), a CDD-like graph structure whose edges are labeled with so-called *constraint matrices*, a structure similar to *difference bound matrices* (DBMs), which is capable of storing both the timing and location information. Intuitively, like CDDs and CRDs, CMDs share nodes for common subexpressions, but unlike CDDs



(a) CRD (b) CDD (c) CMD

Figure 1. CRD, CDD, and CMD representing the same set of clock valuations.

and CRDs, CMDs collapse sequences of edges representing convex constraints into single edges. Figure 1 shows a comparison of the three data structures.

As a key feature, CMDs are not only able to store the set of reachable states, but can also represent the control structure of the system under consideration, which permits the computation of the successor and predecessor states during the run of the reachability algorithm in a symbolic way by simple operations on the CMD structure.

To show the effectiveness of this new approach, we compare a prototype implementation of a CMD-based model checker with RED and UPPAAL on standard benchmarks from the literature.

Related work. The efficiency of a timed model checker strongly depends on the way in which the state space is represented. In the last two decades, several techniques to represent state spaces that consist of a discrete and a continuous part were proposed.

Decidability of reachability (i.e., emptiness) checking for timed automata was shown in [3] by constructing the so-called *region graph*. For practical applications, however, this construction is too fine-grained. A coarser representation of the continuous part of the state space are *clock zones* [1], which can be efficiently described using *difference bound matrices* (DBMs) [13]. However, operations on DBMs such as union or negation might lead to nonconvex sets which cannot be represented by a single DBM.

Asarin et al. [4] used BDDs to encode sets of clock valuations as *numerical decision diagrams* (NDDs) using a discretization scheme based on region equivalence. Similarly, Bozga et al. [9] approximated the precise clock values to discrete time steps, resulting in a pure discrete semantics allowing a state space representation using a single BDD. Based on *closed timed automata* [7], a restricted form of classical timed automata where only nonstrict clock constraints are allowed, Beyer introduced an integer semantics where clock values and location information can be represented jointly in a single BDD. Besides the fact that such pure BDD-based approaches are sensitive to the magnitude of the clocks, it has been observed that the BDDs can blow-up significantly due to interdependencies in the timing behavior of the system.

Seshia and Bryant [22] 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. 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.

Møller et al. introduced *difference decision diagrams* (DDDs) [20], 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. Behrmann et al. proposed *clock difference diagrams* (CDDs) [18, 6], a more space-efficient data structure, which benefits from sharing clock constraints for several clock zones. CDDs store intervals of clock valuations in a BDD-like structure as a rooted, directed, and acyclic graph. As a further extension, Wang proposed *clock restriction diagrams* (CRDs) [24], in which the disjointness requirement is dropped. In contrast to CDDs, CRDs only store upper bounds of clock differences. Location information is added to CRDs by adding binary variable nodes. Diagram-based representations such as CDDs or CRDs are efficient for fragmented state spaces which are mainly nonconvex. However, they cannot exploit convexity of parts of the state space, thus losing efficiency on mostly convex state spaces.

Yamane and Nakamura [25] combined DBMs with BDDs for implementing an approximation technique proposed by Dill and Wong-Toi [14]. Recently, we introduced a model checking approach based on *clock zone maps* (CZMs) [15], where clock zones, represented as DBMs, are mapped onto sets of locations, represented as BDDs. As CZMs are a less flexible special case of CMDs, the present work can be seen as a continuation of this line of research.

2 Preliminaries

For a set X, we use $\mathcal{P}(X)$ to refer to its power set.

2.1 Timed Systems

Clock Constraints. In the rest of the paper, we assume that the set of clocks is given as $\mathcal{C} = \{x_0, \dots, x_n\}$, where x_0 is the special zero-clock whose value is always 0. The set of *clock difference constraints* over $\mathcal{C}$ is defined as

$$\begin{aligned}\mathsf{Cons}(\mathcal{C}) = \{&a \prec^1 x_i - x_j \prec^2 b \mid \\ &a \in \mathbb{Z} \cup \{-\infty\}, b \in \mathbb{Z} \cup \{\infty\}, \prec^{1,2} \in \{<, \leq\}, \\ &0 < i \leq n, 0 \leq j < i\}.\end{aligned}$$

The set of *diagonal-free* clock constraints $\mathsf{Cons}_0(\mathcal{C}) \subseteq \mathsf{Cons}(\mathcal{C})$ is defined as

$$\begin{aligned}\mathsf{Cons}_0(\mathcal{C}) = \{&a \prec^1 x_i - x_0 \prec^2 b \mid \\ &a \in \mathbb{N}_0, b \in \mathbb{N}_0 \cup \{\infty\}, \prec^{1,2} \in \{<, \leq\}, 0 < i \leq n\}.\end{aligned}$$

In the following, we use sets of clock constraints to represent their conjunction and we write **true** for the empty constraint set $\emptyset$.

Timed Automata. The components of a timed system are represented by *timed automata*. A timed automaton [3, 1] is a tuple $\mathcal{A} = (L, L_0, I, \Sigma, \Delta)$, where L is a finite set of (control) locations, $L_0 \subseteq L$ are the initial locations, $I : L \to \mathcal{P}(\mathsf{Cons}_0(\mathcal{C}))$ maps each location to an invariant, Σ is a finite set of actions, and $\Delta \subseteq L \times \Sigma \times \mathcal{P}(\mathsf{Cons}_0(\mathcal{C})) \times \mathcal{P}(\mathcal{C}) \times L$ is a relation defining discrete location switches. Here, we require that invariants do not have lower bounds.

A *clock valuation* $\vec{t} : \mathcal{C} \to \mathbb{R}_{\geq 0}$ assigns a nonnegative value to each clock and can also be represented by a $|\mathcal{C}|$-dimensional vector $\vec{t} \in \mathcal{R}$, where $\mathcal{R} = \mathbb{R}^{\mathcal{C}}_{\geq 0}$ denotes the set of all clock valuations. For the special zero-clock x_0, we always have $\vec{t}(x_0) = 0$. The states of a timed automaton are pairs $(l, \vec{t})$ of locations and clock valuations. Timed automata have two types of transitions: *timed transitions*, where only time passes by and the location remains unchanged, and *discrete transitions*. In a timed transition, the same nonnegative value $d \in \mathbb{R}_{\geq 0}$ is added to all clocks such that, for each $0 \leq d' \leq d$, $\vec{t} + d' \cdot \vec{1}$ satisfies the location invariant $I(l)$. In a discrete transition, for some element $\delta = (l, a, \varphi, r, l')$ of Δ, the state instantaneously changes from $(l, \vec{t})$ to $(l', \vec{t}')$ provided that (1) $\vec{t}$ satisfies the guard φ, (2) $\vec{t}' = \vec{t}[r := 0]$ is obtained from $\vec{t}$ by setting the clocks in r to 0, and (3) $\vec{t}'$ satisfies the next location invariant $I(l')$. We say that a timed state s' is *reachable* from a timed state s iff there is a sequence of timed and/or discrete transitions starting in s and ending in s'.

2.2 Binary Decision Diagrams

For representing sets of locations symbolically, we use *reduced ordered binary decision diagrams* (BDDs) [10, 11], which describe characteristic functions $f : \mathcal{P}(\mathcal{B}) \to \mathbb{B}$ for some finite set of variables $\mathcal{B}$. 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 Boolean functions, abbreviated as BFs) f and f', we define their conjunction and disjunction as $(f \wedge f')(x) = f(x) \wedge f'(x)$ and $(f \vee f')(x) = f(x) \vee f'(x)$ for all $x \subseteq \mathcal{B}$. The negation of a BF is defined similarly. Given some set of variables $V \subseteq \mathcal{B}$ and a BF f, we define $\exists V : f$ as the function that maps all $x \subseteq \mathcal{B}$ to **true** for which there exists some $y \subseteq V$ such that $f(y \cup (x \setminus V)) = \textbf{true}$. Given two ordered lists of variables $W = w_1, \dots, w_n$ and $W' = w'_1, \dots w'_n$ of the same length, we furthermore denote by $f[W/W']$ the BF for which some $x \subseteq \mathcal{B}$ is mapped to true if and only if $f((x \setminus \{w'_1, \dots, w'_n\}) \cup \{w_i \mid \exists 1 \leq i \leq n : w'_i \in x\}) = \textbf{true}$. For the scope of this paper, we use sets of variables and their characteristic functions interchangeably.

Binary functions can be used to encode sets of locations and purely discrete transition relations between them. Given a set of locations L and a set of actions Σ, we can represent a transition relation $\gamma \subseteq L \times \Sigma \times L$ as a BF over three lists of Boolean variables *pre*, *post*, and *acts*, which we use to encode the predecessor locations, successor locations, and actions in γ, respectively. Then, the BF over $\mathcal{B} = \mathit{pre} \cup \mathit{post} \cup \mathit{acts}$ for γ can be written as $\bigvee_{(l,a,l') \in \gamma} (\!|l|\!) \wedge (\!|l'|\!)' \wedge (\!|a|\!)''$ for three functions $(\!|\cdot|\!)$, $(\!|\cdot|\!)'$, and $(\!|\cdot|\!)''$ mapping locations onto BF valuations over *pre* and *post*, and actions onto *acts*, respectively. For the sake of usefulness in model checking, we additionally require the following properties:

- For all $l \in L$: $(\!|l|\!)'[\mathit{pre}/\mathit{post}] = (\!|l|\!)$;
- For all $l, l' \in L$: $l \neq l' \to (\!|l|\!) \wedge (\!|l'|\!) = \textbf{false}$ and for

	x_1	x_2	x_3
x_0	0	1	2
x_1		3	4
x_2			5

Figure 2. Graphical representation of the idx function for the clocks $\mathcal{C} = \{x_0, x_1, x_2, x_3\}$. For a column x_i and a row x_j, the number shown in the table is the index of a constraint of the form $a \prec^1 x_i - x_j \prec^2 b$. For a BF constraint c, we have $\mathsf{idx}(c) = 6$.

all $a, a' \in \Sigma$: $a \neq a' \rightarrow \llparenthesis a \rrparenthesis'' \wedge \llparenthesis a' \rrparenthesis'' = \mathbf{false}$;

- $\bigvee_{l \in L} \llparenthesis l \rrparenthesis = \mathbf{true}$ and $\bigvee_{a \in \Sigma} \llparenthesis a \rrparenthesis'' = \mathbf{true}$.

3 Constraint Matrices

In this section, we describe the matrix-based data structure that is later used for labeling the edges of a CMD. Our matrices are similar to DBMs, but can also accommodate a Boolean constraint over the discrete part of the state space.

For a set of clocks $\mathcal{C}$ and a set of BF variables $\mathcal{B}$, an *atomic constraint* (or just constraint) is either a clock difference constraint from $\mathsf{Cons}(\mathcal{C})$ or a BF over $\mathcal{B}$. We say that two atomic constraints c_1, c_2 have the same *type* if either both are BFs or c_1 and c_2 are clock difference constraints over the same pair of clocks. For each atomic constraint c, we define its *constraint index* (or just index) and write $\mathcal{I}$ to refer to the set of all indices $\{0, \ldots, \mathcal{I}_{max}\}$, where $\mathcal{I}_{max} = \frac{n}{2} \cdot (n+1)$. We define $\mathsf{idx}(c) = \left(\sum_{0 \leq k \leq j} (n-k)\right) - n + i - 1$ for all atomic constraints $c = a \prec^1 x_i - x_j \prec^2 b$ and $\mathsf{idx}(c') = \mathcal{I}_{max}$ for the atomic constraints c' that are BFs. Intuitively, this function induces a total order on the atomic constraint types, which will be needed later to impose an order of the constraints occurring along paths in a CMD. The concrete idx function used in this paper assigns the lowest indices to those clock constraints in which the right-hand variable in the inequality has the lowest number and the highest index to those where this variable has the highest number. The BF is last in the order. Figure 2 illustrates our definition of the idx function for a setting with three clocks. However, for the general applicability of our approach, the precise definition of the idx function can be arbitrary as long as it is a bijection between $\mathcal{I}$ and the constraint types.

A *constraint matrix* (or just matrix) m is a set of atomic constraints in which no two different atomic constraints have the same index. We define $\mathcal{M}$ as the set of all constraint matrices. We write $\mathsf{minIdx}(m)$ ($\mathsf{maxIdx}(m)$) to refer to the minimal (maximal) index of a constraint appearing in m. The conjunction $c_1 \wedge c_2$ of two atomic constraints c_1 and c_2 with $\mathsf{idx}(c_1) = \mathsf{idx}(c_2)$ is defined to be the least restrictive constraint that implies both conjuncts. For two constraint matrices m_1 and m_2, we furthermore define:

$$\begin{aligned} m_1 \wedge m_2 = \\ &\{c_1 \wedge c_2 \mid c_1 \in m_1 \wedge c_2 \in m_2 \wedge \mathsf{idx}(c_1) = \mathsf{idx}(c_2)\} \cup \\ &\{c_1 \in m_1 \mid \forall c_2 \in m_2 : \mathsf{idx}(c_1) \neq \mathsf{idx}(c_2)\} \cup \\ &\{c_2 \in m_2 \mid \forall c_1 \in m_1 : \mathsf{idx}(c_1) \neq \mathsf{idx}(c_2)\} \end{aligned}$$

For two constraint matrices m_1 and m_2, we say that m_1 implies m_2, written as $m_1 \Rightarrow m_2$, if $m_1 \wedge m_2 = m_1$. For two indices i and j with $i \leq j$, we furthermore define $m\downarrow_j^i = \{c \in m \mid i \leq \mathsf{idx}(c) \leq j\}$ as the *projection* of m onto the constraints with indices between i and j. We say that a constraint matrix m is *complete* if it ranges over all constraint types, i.e., $|m| = \mathcal{I}_{max} + 1$.

Given some $x \subseteq \mathcal{B}$, a clock valuation $\vec{t}$, and a constraint matrix m, we say that $(x, \vec{t})$ satisfies m, written as $(x, \vec{t}) \models m$, if for all $c \in m$, either (1) c is a BF and $c(x) = \mathbf{true}$; or (2) the constraint c is of the form $a \prec^1 x_i - x_j \prec^2 b$ and $a \prec^1 \vec{t}(x_i) - \vec{t}(x_j) \prec^2 b$. The semantics of a constraint matrix m is given as the set of pairs of BF variable and clock variable valuations which are represented by m: $\llbracket m \rrbracket = \{s \in \mathcal{P}(\mathcal{B}) \times \mathcal{R} \mid s \models m\}$.

4 Constraint Matrix Diagrams

In this section, we formally introduce our diagram-based data structure, characterize reduced forms, and define necessary Boolean operations.

A *constraint matrix diagram* (CMD) over the set of constraint matrices $\mathcal{M}$ is a tuple $M = (Q, q_0, q_\top, \mathsf{type}, E)$, where

- Q is a finite set of nodes,
- $q_0 \in Q$ is the root node,
- $q_\top \in Q$ is the sink,
- $\mathsf{type} : Q \rightarrow \mathcal{I} \cup \{\mathcal{I}_{max} + 1\}$ is a total function that associates a constraint index to each node, and
- $E \subseteq Q \times \mathcal{M} \times Q$ is an edge relation.

Additionally, we require that (1) (Q, E) is a directed acyclic graph with precisely one source node q_0 and one sink node $q_\top$; (2) $\mathsf{type}(q_0) = 0$ and $\mathsf{type}(q_\top) = \mathcal{I}_{max} + 1$; (3) for each edge $(q, m, q') \in E$, $\mathsf{minIdx}(m) \geq \mathsf{type}(q)$ and $\mathsf{maxIdx}(m) < \mathsf{type}(q')$. We define $\mathcal{D}$ as the set of all CMDs.

If E is clear from the context, for convenience, we write $q \xrightarrow{m} q'$ for $(q, m, q') \in E$. We write $\mathsf{root}(M)$ to refer to M's root node q_0. A (complete) *path* p of M is a sequence of nodes and matrices of the form

$$q_0 \xrightarrow{m_0} q_1 \xrightarrow{m_1} \ldots \xrightarrow{m_{k-1}} q_k$$

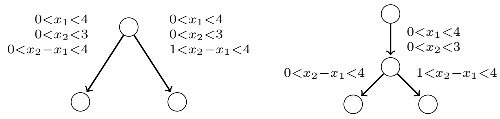


Figure 3. Splitting a CMD edge with common lowest-index atomic constraints.

such that $(q_i, m_i, q_{i+1}) \in E$, for each $0 \le i < k$, and $q_k = q_\top$. We write $\mathsf{nodes}(p) = \{q_0, \ldots, q_k\}$ to refer to the nodes of p, and $\bigwedge p$ to refer to the complete matrix represented by p, $\bigwedge_{0 \le i < k} m_i$. We refer to $\mathsf{paths}(M)$ to denote the set of all paths of M. The semantics of M is defined as $[\![M]\!] = \bigcup_{p \in \mathsf{paths}(M)} [\![\bigwedge p]\!]$. The empty CMD is given (for some suitable function type satisfying the properties stated above) by

$$\mathsf{cmd}(\mathbf{false}) = (\{q_0, q_\top\}, q_0, q_\top, \mathsf{type}, \emptyset).$$

We convert a matrix m into a CMD by

$$\mathsf{cmd}(m) = (\{q_0, q_\top\}, q_0, q_\top, \mathsf{type}, \{(q_0, m', q_\top)\}),$$

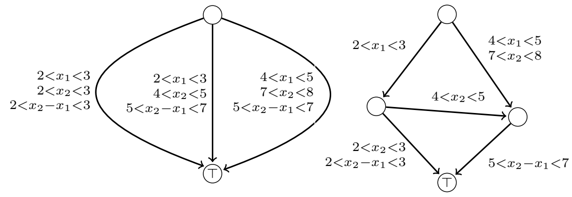
where $m' \supseteq m$ is a complete matrix that contains all constraints from m plus, for each constraint type that is not contained in m, the weakest possible constraint with that type.

4.1 Reduced Forms

Unlike DBMs or BDDs, CMDs do not have a canonical form which, for a set of states, defines a unique CMD. For simplicity, we thus define the following two types of *reduced forms*: (1) the *diagram form* where a single edge may belong to many complete paths and thus maximizes the sharing of common constraints along the paths, and (2) the *compact form* where the CMD comprises only the root node and the true node. Note that, in general, one could also define other reduced forms. However, in this paper, we stick to the two reduced forms mentioned above and leave the investigation of other forms as future work.

Formally, a CMD $M = (Q, q_0, q_\top, \mathsf{type}, E)$ is in *diagram form* iff

$$\begin{aligned}
&\forall (q_1, m_1, q_1') \in E\ \forall (q_2, m_2, q_2') \in E :\\
&\quad \Big(\big(q_1 = q_2 \wedge m_1{\downarrow}^{\mathsf{type}(q_1)}_{\mathsf{type}(q_1)} = m_2{\downarrow}^{\mathsf{type}(q_2)}_{\mathsf{type}(q_2)}\big)\\
&\qquad\qquad \Rightarrow \big(m_1 = m_2 \wedge q_1' = q_2'\big)\Big) \wedge\\
&\quad \Big(\big(q_1' = q_2' \wedge m_1{\downarrow}^{\mathsf{type}(q_1')-1}_{\mathsf{type}(q_1')-1} = m_2{\downarrow}^{\mathsf{type}(q_2')-1}_{\mathsf{type}(q_2')-1}\big)\\
&\qquad\qquad \Rightarrow \big(m_1 = m_2 \wedge q_1 = q_2\big)\Big).
\end{aligned}$$



(a) Compact form (b) Diagram form

Figure 4. Semantically equivalent CMDs.

Intuitively, this definition requires that in a CMD in diagram form, the overall number of atomic constraints is minimized by introducing intermediate nodes in the CMD whenever (1) two outgoing edges from the same node in a CMD share their lowest-index atomic constraint, and (2) two ingoing edges to the same node in a CMD share their highest-index atomic constraint. Figure 3 shows an example of introducing such an intermediate node.

A CMD $M = (Q, q_0, q_\top, \mathsf{type}, E)$ is in *compact form* iff $Q = \{q_0, q_\top\}$, i.e., all paths go directly from the root to the true node. For illustration, Fig. 4 shows two semantically equivalent CMDs that are in compact and diagram form, respectively. Note that any CMD can be transformed into both forms.

4.2 Boolean Operations

In this subsection, we describe a conjunction operator for two CMDs and a disjunction operator for a CMD and a matrix. Both are used in the reachability fixed point algorithm described in Sect. 5.

Disjunction. We start by explaining the disjunction operator. The function $\mathsf{Or} : \mathcal{M} \times \mathcal{D} \to \mathcal{D}$ takes a matrix m and a CMD M to compute a CMD M' such that $[\![M']\!] = [\![M]\!] \cup [\![m]\!]$. We give two versions of the Or operator that maintain diagram and compact form, respectively, and start with the former. Our concrete definition of Or described below assures that if there is a path $p \in \mathsf{paths}(M)$ with $m \Rightarrow \bigwedge p$ then $M' = M$.

Before we come to the actual definition of Or for the diagram form, we introduce some auxiliary definitions. The set of *backward-deterministic paths*, i.e., paths with a backward unique sequence of nodes, of highest index $i \in \mathcal{I}$ is defined as

$$\begin{aligned}
\mathsf{dpaths}(M, i) = \big\{&p \in \mathsf{paths}(M) \mid\\
&\forall q \in \mathsf{nodes}(p) : \mathsf{type}(q) \le i \Rightarrow \mathsf{indeg}(q) \le 1\big\},
\end{aligned}$$

where the *indegree* of $q \in Q$ is defined as

$$\mathsf{indeg}(q) = |\{(q_1, m, q_2) \in E \mid q_2 = q\}|.$$

With this definition, we define the set of *backward-deterministic prefixes* of M for a matrix m:

$$\begin{aligned}\mathsf{dpref}(M, m) = \Big\{&(x, i) \in (E \cup Q) \times \mathcal{I} \mid \\ &\exists p \in \mathsf{dpaths}(M, i) : \big(\textstyle\bigwedge p\big)\downarrow_i^0 = m\downarrow_i^0 \wedge \\ &\big(x \in \mathsf{nodes}(p) \wedge \mathsf{type}(x) = i \vee \\ &x = (q_1, m', q_2) \in E \wedge \{q_1, q_2\} \subseteq \mathsf{nodes}(p) \wedge \\ &\mathsf{type}(q_1) < i < \mathsf{type}(q_2)\big)\Big\} \cup \{(q_0, \mathsf{type}(q_0))\}\end{aligned}$$

Due to the lack of space, we omit the (analogous) definition of the set of *forward-deterministic suffixes* of M for a matrix m, $\mathsf{dsuf}(M, m)$.

With these auxiliary definitions, for a matrix m and a CMD M, we can define $\mathsf{Or}(m, M)$ for the diagram form in Algorithm 1. The basic idea is to (1) find a maximal backward-deterministic prefix which starts in q_0 and ends in a node q_f, (2) find a maximal forward-deterministic suffix which starts in a node q_b and ends in $q_\top$, and (3) connect q_f and q_b by a matrix $m' \subseteq m$ that contains the atomic constraints whose types do not occur on the paths from q_0 to q_f and q_b to $q_\top$. First, in a forward traversal over the graph structure (whose running time is linear in $|Q| + |E|$) starting in q_0, we check if m is already subsumed by some path in M. During the same traversal, we compute a maximal backward-deterministic prefix, which is used in the function $\mathsf{SplitTop}$ (Algorithm 2). Here, we determine q_f as a node that is already contained in M, or if the prefix ends between two nodes q and q', we split the edge that connects q and q', and introduce a new intermediate node q_f. Then, in $\mathsf{SplitBottom}$ (Algorithm 3) we analogously determine (or introduce a new) node q_b in a backward traversal over the graph structure (whose running time is linear in $|Q|+|E|$) starting in $q_\top$. Once q_f and q_b are determined, we connect q_f and q_b with a new edge labeled with m projected to the appropriate constraint indices, m_{fb}. Additionally, we also locally remove all edges from q_f to q_b which are subsumed by m_{fb}. Note that if the input CMD M is in diagram form, the CMD resulting from taking the disjunction is also in diagram form.

Theorem 4.1 *For two CMDs M and M', and a matrix m, if $M' = \mathsf{Or}(m, M)$ then*
1. $[\![M']\!] = [\![M]\!] \cup [\![m]\!]$ *and*
2. *if* $\exists p \in \mathsf{paths}(M) : m \Rightarrow \bigwedge p$ *then* $M' = M$.

Sketch of proof: The first claim follows from the fact that (1) $\mathsf{SplitTop}$ and $\mathsf{SplitBottom}$ do not change the semantics of M, (2) since q_f represents a backward-deterministic prefix and q_b represents a forward-deterministic suffix, all paths that go through

Algorithm 1 $\mathsf{Or}(m, M)$, for a matrix m and a CMD $M = (Q, q_0, q_\top, \mathsf{type}, E)$.

```
if ∃p ∈ paths(M) : m ⇒ ⋀p then
  return M /* do nothing */
else
  (M'', q_f) := SplitTop(m, M)
  (M', q_b) := SplitBottom(m, M'', q_f)
      /* M' = (Q', q_0, q_⊤, type', E') */
  m_fb := m↓^{type'(q_f)}_{type'(q_b)−1}
  E' := E' \ {(q_f, m'', q_b) ∈ E' | m_fb ⇒ m''}
      ∪ {(q_f, m_fb, q_b)}
  return M'
```

Algorithm 2 $\mathsf{SplitTop}(m, M)$, for a matrix m and a CMD $M = (Q, q_0, q_\top, \mathsf{type}, E)$.

```
pick (x, i) ∈ dpref(M, m) s.t. i is maximal
if x = (q, m', q') ∈ E then
  Q' := Q ⊎ {q_f}, type' := type[q_f ↦ i]
  E' := E \ {x}
        ∪ {(q, m'↓^{type'(q)}_{i−1}, q_f)}
        ∪ {(q_f, m'↓^{i}_{type'(q')−1}, q')}
  return ((Q', q_0, q_⊤, type', E'), q_f)
else /* x ∈ Q */
  return (M, x)
```

Algorithm 3 $\mathsf{SplitBottom}(m, M, q_f)$, for a matrix m, a CMD $M = (Q, q_0, q_\top, \mathsf{type}, E)$, and a node $q_f \in Q$.

```
pick (x, j) ∈ dsuf(M, m) s.t.
    max(type(q_f) + 1, j) is minimal
if x = (q, m', q') ∈ E then
  Q' := Q ⊎ {q_b}, type' := type[q_b ↦ j]
  E' := E \ {x}
      ∪ {(q, m↓^{type'(q)}_{j−1}, q_b)}
      ∪ {(q_b, m↓^{j}_{type'(q')−1}, q')}
  return ((Q', q_0, q_⊤, type', E'), q_b)
else /* x ∈ Q */
  return (M, x)
```

q_f and q_b only differ between q_f and q_b, and (3) all paths having an edge between q_f and q_b whose matrix is subsumed by $m_{fb} \subseteq m$ are replaced by a new path that exactly represents m. The second claim directly follows from the first line of Algorithm 1. $\square$

The Or operator for the compact form is defined by a slightly changed Algorithm 1, where the calls to $\mathsf{SplitTop}$ and $\mathsf{SplitBottom}$ are replaced by $q_f := q_0$, $q_b := q_\top$, and $M' := M$.

Conjunction. We define the conjunction operator $\mathsf{And} : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ that takes two CMDs A and B, and computes a third CMD C such that $[\![C]\!] = [\![A]\!] \cap [\![B]\!]$. It recursively combines the paths of A and B, and computes the conjunction of the matrices represented by the paths. Instead of providing such a binary And operator directly, we first define a more generic operator $\mathsf{AndApply}$ that additionally takes a polymorphic function as parameter which, in turn, is applied to each computed matrix combination. This will become useful when defining the reachability algorithm in Sect. 5.2. For a polymorphic type α, we define the function

$$\mathsf{AndApply} : (\mathcal{D} \times \mathcal{D} \times (\mathcal{M} \times \alpha \to \alpha) \times \alpha) \to \alpha,$$

which is defined as

$$\mathsf{AndApply}(A, B, f, X) := \mathsf{ApplyRec}\left(\mathsf{root}(A), \mathsf{root}(B), f, \mathbf{true}, X\right),$$

where $\mathsf{ApplyRec}$ is defined in Algorithm 4. For two CMDs A and B, a polymorphic function $f : \mathcal{M} \times \alpha \to \alpha$, and a context $X \in \alpha$, the basic idea is to (1) recursively combine each path of A with each path of B thereby computing the conjunction of the constraints observed along both paths, and (2) when the recursion jointly reaches the sink nodes, we apply the function f to the propagated matrix m and the context X. Note that in practice, applying $\mathsf{ApplyRec}$ to CMDs in diagram form turns out to be time-efficient as (1) the recursion can stop as soon as m becomes unsatisfiable, and (2) only intersections on partial matrices have to be computed.

We can now easily define And as follows: $\mathsf{And}(A, B) := \mathsf{AndApply}(A, B, \mathsf{Or}, \mathsf{cmd}(\mathbf{false}))$.

Theorem 4.2 *For CMDs A, B, and C, if $C = \mathsf{And}(A, B)$ then $[\![C]\!] = [\![A]\!] \cap [\![B]\!]$.*

Sketch of proof: The correctness follows directly from (1) the correctness of Or (Theorem 4.1), (2) $[\![A]\!] \cap [\![B]\!] = \bigcup_{p \in \mathsf{paths}(A)} [\![\bigwedge p]\!] \cap \bigcup_{p' \in \mathsf{paths}(B)} [\![\bigwedge p']\!] = \bigcup_{p \in \mathsf{paths}(A), p' \in \mathsf{paths}(B)} [\![\bigwedge p \wedge \bigwedge p']\!]$, and (3) by structural induction on $\mathsf{ApplyRec}$. $\square$

Algorithm 4 $\mathsf{ApplyRec}(a, b, f, m, X)$, for two CMD nodes a and b, a function $f : \mathcal{M} \times \alpha \to \alpha$, a matrix m, and a context $X \in \alpha$.

```
if m ≢ false then
  if a = b = q⊤ then
    return f(m, X)
  else if type(a) = type(b) then
    for all a --m'_1--> a' do
      for all b --m'_2--> b' do
        X := ApplyRec(a', b', f, m ∧ m'_1 ∧ m'_2, X)
  else if type(a) > type(b) then
    for all b --m'--> b' do
      X := ApplyRec(a, b', f, m ∧ m', X)
  else if type(a) < type(b) then
    for all a --m'--> a' do
      X := ApplyRec(a', b, f, m ∧ m', X)
return X
```

5 Model Checking using CMDs

In this section, we describe the actual CMD-based timed reachability checking algorithm. Both *forward* and *backward* reachability checking are possible with our data structure. For the sake of brevity, we focus only on the forward case since the backward case can be done analogously.

In the following, we assume that some fixed timed automaton $\mathcal{A} = (L, L_0, I, \Sigma, \Delta)$, describing the system under consideration, is given. Using the notation from Sect. 2.2, for a CMD M, we define $[\![M]\!]_L = \{(l, \vec{t}) \in L \times \mathcal{R} \mid ((\!|l|\!), \vec{t}) \in [\![M]\!]\}$ as the set of timed states represented by M.

5.1 Invariants and Transition Relations

As a prerequisite for the reachability algorithm, we need to construct some CMDs that represent $\mathcal{A}$'s control structure. More precisely, on the one hand, we construct a CMD $\mathfrak{I}$ such that $(l, \vec{t}) \in [\![\mathfrak{I}]\!]_L$ if and only if $\vec{t} \models I(l)$, i.e., $\mathfrak{I}$ represents those timed states which do not violate any location invariant. On the other hand, for each set of resets $r \subseteq \mathcal{C}$ appearing in Δ, we compute a CMD $\mathfrak{T}_r$ over $\mathcal{B}$ and $\mathcal{C}$ such that for two locations l, l' from L and a clock valuation $\vec{t}$, we have $((\!|l|\!) \wedge (\!|l'|\!)', \vec{t}) \in [\![\mathfrak{T}_r]\!]$ if and only if there exists some action $a \in \Sigma$ and a set of clock constraints φ with $\vec{t} \models \varphi$ such that $(l, a, \varphi, r, l') \in \Delta$. That is, each $\mathfrak{T}_r$ relates those locations which are connected by a discrete location switch with guard φ and resets r.

The function $\mathsf{CreateInv}()$, defined in Algorithm 5, constructs $\mathfrak{I}$. For a set of clock resets $r \subseteq \mathcal{C}$, the function $\mathsf{CreateTrans}(r)$, defined in Algorithm 6, constructs $\mathfrak{T}_r$. The function iterates over the actions and locations and successively adds the transitions found to the transition CMD B. Before returning it, the actions are removed from the BFs in the CMD by existentially quantifying them out. The function is straight-forward to extend to, e.g., the verification of networks of timed automata [1] by computing the transition CMDs A for the individual automata in the network and taking their conjunction before removing the actions from their BFs.

Algorithm 5 $\mathsf{CreateInv}()$.

$A := \mathsf{cmd}(\mathbf{true})$
for all $l \in L$ **do**
  $A := \mathsf{And}\Big(A, \mathsf{Or}\left(\neg [\![l]\!], \mathsf{cmd}(I(l))\right)\Big)$
return A

Algorithm 6 $\mathsf{CreateTrans}(r)$, for clock resets $r \subseteq \mathcal{C}$.

$A := \mathsf{cmd}(\mathbf{true})$
for all $a \in \Sigma$ **do**
  $B := \mathsf{cmd}(\mathbf{true})$
  for all $l \in L$ **do**
    $C := \mathsf{cmd}(\neg [\![l]\!])$
    for all $l \xrightarrow{a,c,r} l'$ **do**
      $C := \mathsf{Or}(c \wedge [\![l']\!]', C)$
    $B := \mathsf{And}(B, C)$
  $B := \mathsf{Or}(\neg [\![a]\!]'', B)$
  $A := \mathsf{And}(A, B)$
return A, where each BF b is replaced by $\exists acts : b$

5.2 Reachability Algorithm

In this subsection, we present the actual reachability algorithm that checks if some target states (e.g., the states that violate a safety property) are reachable from some source states (e.g., the initial states $L_0 \times \mathcal{R}$). The computation is carried out in a least fixed point construction that starts with the source states and computes a successively increasing series of so-called *prefixed points* converging to those states which are exactly reachable from the source states. Each prefixed point is represented as a CMD over the set of clocks $\mathcal{C}$ and the BF variables *pre*.

For a complete constraint matrix, we define the clock reset operator $m[r := 0]$, for $r \subseteq \mathcal{C}$, and the time elapse operator $m^{\Uparrow}$, which are defined analogously to those defined for DBMs [13] (by ignoring the BF constraint). Note that for these operations, analogously to DBMs, we assume the matrix to be in canonical form, i.e., each constraint is as strong as possible. We assume that the time elapse operator also performs *maximal constant widening* to ensure that only finitely many matrices will arise in the forward analysis [8].

Before we come to the actual definition of the algorithm, we first introduce the operator $\mathsf{Succ} : \mathcal{P}(\mathcal{C}) \times \mathcal{D} \to \mathcal{M} \times \mathcal{D} \to \mathcal{D}$ which constructs, for a set of clock resets $r \subseteq \mathcal{C}$ and an invariant CMD $\mathfrak{I}$, a corresponding *successor* function:

$$\mathsf{Succ}(r, \mathfrak{I}) := \lambda(m, R). \\ \mathsf{AndApply}\left(\mathsf{Post}(m[r := 0]), \mathfrak{I}, \mathsf{Or}, R\right)$$

Here, $\mathsf{Post} : \mathcal{M} \to \mathcal{D}$ is the combined symbolic *post* operator $\mathsf{Post}(m) := \mathsf{cmd}(\exists post : m^{\Uparrow}[pre/post])$. More clearly, if $f = \mathsf{Succ}(r, \mathfrak{I})$ then, for a complete matrix m and a CMD R, $f(m, R)$ returns a CMD that represents all states from $[\![R]\!]_L$ extended by precisely those states which are reachable from $[\![m]\!]_L$ by first executing a discrete and then a timed transition.

With these and the other definitions from the previous sections, we can state Algorithm 7, which represents a sound and complete CMD-based decision procedure for checking timed reachability.

Algorithm 7 $\mathsf{Reachable}(sources, targets)$, for two CMDs *sources* and *targets*.

for all relevant $r \subseteq \mathcal{C}$ **do**
  $\mathfrak{T}_r := \mathsf{CreateTrans}(r)$
$\mathfrak{I} := \mathsf{CreateInv}()$
$R := \mathsf{And}(sources, \mathfrak{I})$
$D := R$
while $[\![D]\!] \neq \emptyset$ and $[\![\mathsf{And}(D, targets)]\!] = \emptyset$ **do**
  $D' := \mathsf{cmd}(\mathbf{false})$
  for all relevant $r \subseteq \mathcal{C}$ **do**
    $R := \mathsf{AndApply}\left(D, \mathfrak{T}_r, \mathsf{Succ}(r, \mathfrak{I}), R\right)$
    for all newly added matrices m in R **do**
      $D' := \mathsf{Or}(m, D')$
  $D := D'$
return $[\![\mathsf{And}(R, targets)]\!] \neq \emptyset$

In the computation of the successor states, by keeping track of all matrices m that modify R in an $\mathsf{Or}(m, R)$ operation, we avoid an expensive comparison of two CMDs representing subsequent prefixed points.

Theorem 5.1 *The following statements hold true:*

1. *Algorithm 7 terminates in a finite number of steps;*
2. *For two CMDs sources and targets, some states in $[\![targets]\!]_L$ are reachable from some states in $[\![sources]\!]_L$ iff $\mathsf{Reachable}(sources, targets)$ is true.*

Sketch of proof: The first claim follows from the fact that (1) there are only finitely many BFs, (2) there are only finitely many difference constraints since we apply a finiteness-ensuring widening after time elapsing $m^{\Uparrow}$, and (3) $\mathsf{Or}(m, M)$ leaves M unchanged if m is already subsumed in M. The correctness of the widening operation is guaranteed since we only allow diagonal-free constraints in the definition of timed automata [8].

The second claim follows (1) by structural induction on $\mathsf{AndApply}$, (2) from the fact that the clock reset and time elapse operations are only performed on complete matrices, (3) from the fact that in each iteration of the while loop of Algorithm 7, for each relevant $r \subseteq \mathcal{C}$, $p_d \in \mathsf{paths}(D)$, $p_t \in \mathsf{paths}(\mathfrak{T}_r)$, $p_i \in \mathsf{paths}(\mathfrak{I})$, $[\![R]\!]_L$ is extended by $[\![\exists post : ((\bigwedge p_d \wedge \bigwedge p_t)[r := 0]^{\Uparrow}[pre/post]) \wedge \bigwedge p_i]\!]_L$, which corresponds exactly to the classical post operator for symbolic timed and discrete model checking, and (4) by induction over the construction of R. □

6 Experimental Results

6.1 Prototype Implementation

We implemented a CMD prototype model checker in C^{++} using the CuDD BDD library [23] to represent the BFs in the constraint matrices. The first step is to call the Nova tool from the SIS toolset [21] to find efficient assignments of control locations to BDD variable valuations for all timed automata in the given network. This defines the functions $(\!|\cdot|\!)$ and $(\!|\cdot|\!)'$ for the automata in the network. We then take the Cartesian product of these functions for the individual automata to obtain the functions $(\!|\cdot|\!)$ and $(\!|\cdot|\!)'$ for the product automaton.

A run of our tool is parametrized in (1) the direction of exploration: either forward or backward, and (2) the reduced form of the CMDs: either diagram or compact. Depending on the selected reduced form, the appropriate disjunction operator Or is chosen. Depending on the direction of exploration, as described in Sect. 5.1, we initialize the CMDs representing the transition relations for the various clock resets of the input timed system. Then, as written in Sect. 5.2, we compute the fixed point of (forward or backward) reachable states. If the direction of exploration is backward, before we construct the transition relation, we also compute an over-approximation of the *discrete* forward reachable states in a (cheap) purely BDD-based fixed point construction. Then, the backward fixed point construction starts with the error states restricted to the forward reachable discrete states.

Note that our prototype *does not* make use of any other optimization techniques such as, e.g., symmetry reduction, or redundant clock removal.

6.2 Benchmarks

We evaluated our approach on several benchmarks[1] from the real-time model checking domain. When checking safety properties, we check the (un)reachability of error states. When checking bounded liveness properties, we (1) add an additional observer automaton that enters a timeout location after a certain amount of time without having seen the global goal event, and (2) check the (un)reachability of the timeout location.

The *Gear Production Stack* (GPS) benchmark represents a manufacturing plant that consists of communicating processing stations. Whenever a gear is loaded into the plant, it gets processed by each station in a sequential manner. We check the bounded liveness property whether a gear is always processed within a certain time. The *FlexRay* benchmark (introduced in [15]) represents the physical layer protocol of FlexRay's CODEC process as defined in [16], using a simplified model of an unreliable physical layer. As a safety property, we check that in the received message there is no deviation from the sent message. The *Fischer* benchmark models Fischer's mutual exclusion protocol. We check the safety property that two processes never enter the critical section at the same time. Here, the models that do not satisfy the property (Sat=No instances in Table 2) comprise two processes that have unsafe timing parameters. The *FDDI* benchmark models a fiber-optic token ring local area network [12]. We check the safety property that the token is always at exactly one station. The *Leader Election* benchmark models a timed leader election in a ring protocol. We check the bounded liveness property that a leader is always elected within a certain time.

6.3 Results

We compared the results of our prototype with the real-time model checkers RED version 8.100429 [24] and Uppaal version 4.0.11 [5]. While our prototype can do both forward and backward reachability checking, RED only performs a backward reachability analysis, and Uppaal only does a forward analysis. All experiments were executed on a 2.6 GHz AMD Opteron processor running Linux. The time limit was set to four hours (TIMEOUT) while the memory peak consumption limit was set to 4 GB (MEMOUT).

Table 2 shows the results of the comparison, where all running times are given in seconds and the memory peak consumptions are given in MB. In the first two

[1]The models are available at
http://www.avacs.org/Benchmarks/Open/rtss10.tgz

columns, the benchmark instance is specified. Then, in the next four columns, the results for our prototype CMD model checker are given comprising the mode (B/F = backward/forward reachability analysis, D/C = diagram/compact reduced form), the number of exploration steps (i.e., fixed point iterations), the running time, and the memory peak consumption. The next three columns show the results for RED comprising the number of exploration steps, the running time, and the memory peak consumption. In the last four columns, the results for Uppaal are shown comprising the command line parameters (-C = use DBMs, -S2 = aggressive space optimization), the number of explored states, the running time, and the memory peak consumption. For our prototype and Uppaal, we always selected the mode/parameters with the best running times without suffering running out of memory.

For the GPS benchmark, our prototype model checker always clearly outperforms both RED and Uppaal. Here, the fully symbolic state space representation as well as the small number of distinct clock difference constraints which arise in the reachable states turn out to be beneficial for CMDs. As already observed in [15], the (discrete) data-intensive FlexRay benchmark greatly benefits from a BDD-based representation of the untimed part of the state space. Here, our approach is capable of handling messages up to the full length of 262 bytes, which is not possible for RED or Uppaal. Interestingly, our approach also outperforms RED and Uppaal on the safe Fischer instances. However, for the unsafe Fischer instances, the semi-symbolic reachability analysis in Uppaal appears to be very effective here. The correctness of the FDDI instances can be established already in the pure discrete over-approximation that is computed prior to the actual precise reachability fixed point construction. That is why our prototype as well as RED outperform Uppaal on this benchmark. On the Leader Election benchmark, our prototype performs better than RED but cannot compete with Uppaal. Similar to the unsafe Fischer instances, it appears that the semi-symbolic approach is more appropriate here.

Table 3 shows the performance of our prototype for different parameter combinations. Here, one can observe that, e.g., Fischer benefits from the diagram reduced form, while, e.g., FlexRay performs better with the compact form.

To have a fair comparison with RED on the Fischer benchmark, we ran RED (1) on nonparametrized models where each process is explicitly modeled as a separate timed automaton, and (2) on parametrized models comprising one timed automaton template which is instantiated for each process[2]. The results for the parametrized instances are shown in Table 1. Since our prototype does not make use of the additional insight that is given through the parametric modeling, we used the nonparametrized models in the comparison shown in Table 2. However, as one can see in both tables, our prototype also outperforms RED on the unsafe parametrized Fischer instances.

		RED		
Benchmark	Sat	Steps	Time	Mem
Fischer (param.) 13	No	14	447	1314
Fischer (param.) 14	No	14	1270	2209
Fischer (param.) 15	No	MEMOUT		
Fischer (param.) 13	Yes	5	346	1444
Fischer (param.) 14	Yes	5	1019	2685
Fischer (param.) 15	Yes	MEMOUT		

Table 1. RED **on the parametrized Fischer benchmark.**

7 Conclusion and Outlook

We presented clock matrix diagrams, a novel data structure for representing state sets in the fully symbolic reachability analysis of real-time systems. In contrast to pure matrix-based or pure diagram-based approaches, CMDs are more versatile as they represent convex subparts as matrices and arrange them in a diagram structure. Inspired by the very promising results, we plan to investigate constraint ordering heuristics, and beyond reachability checking, other application areas such as abstraction refinement or timed game solving based on CMDs.

Acknowledgment. This work was partially 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).

The authors want to thank the anonymous reviewers for their helpful suggestions and Arnd Hartmanns for comments on an early draft of this paper.

References

[1] R. Alur. Timed automata. *CAV*, vol. 1633 of *LNCS*, pp. 8–22, 1999.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.

[2] For unsafe instances, we have a safe and an unsafe template.

		CMD model checker				RED			UPPAAL			
Benchmark	Sat	Mode	Steps	Time	Mem	Steps	Time	Mem	Params	States	Time	Mem
GPS 16	No	B/D	49	4	204	49	795	2923	-C	1365519	55	266
GPS 17	No	B/D	52	4	163	MEMOUT			-C -S2	3174448	139	470
GPS 19	No	B/D	58	4	221	MEMOUT			-C -S2	17155714	974	2425
GPS 20	No	B/D	61	5	229	MEMOUT			-S2	MEMOUT		
GPS 22	No	B/D	67	7	284	MEMOUT			-S2	MEMOUT		
GPS 15	Yes	B/D	46	3	146	46	169	1437	-S2	43046719	1612	3640
GPS 16	Yes	B/D	49	4	204	49	820	2923	-S2	MEMOUT		
GPS 17	Yes	B/D	52	4	218	MEMOUT			-S2	MEMOUT		
GPS 22	Yes	B/D	67	6	278	MEMOUT			-S2	MEMOUT		
FlexRay 1	Yes	F/C	987	16	172	MEMOUT			-C -S2	2368799	17	88
FlexRay 33	Yes	F/C	11851	524	577	MEMOUT			-C -S2	182095135	1515	3907
FlexRay 34	Yes	F/C	12191	527	584	MEMOUT			-S2	MEMOUT		
FlexRay 100	Yes	F/C	34599	695	761	MEMOUT			-S2	MEMOUT		
FlexRay 200	Yes	F/C	68551	2599	1299	MEMOUT			-S2	MEMOUT		
FlexRay 262	Yes	F/C	89603	2869	1482	MEMOUT			-S2	MEMOUT		
Fischer 11	No	B/D	6	13	228	6	8540	3472		2525	0	37
Fischer 12	No	B/D	6	28	297	MEMOUT			-C -S2	4521	0	37
Fischer 19	No	B/D	6	2864	3788	MEMOUT			-C	42941	7	130
Fischer 20	No	B/D	MEMOUT			MEMOUT			-C -S2	54341	9	147
Fischer 11	Yes	B/D	13	119	395	5	6693	3470	-C	2730268	112	233
Fischer 12	Yes	B/D	14	308	698	MEMOUT			-C	8936216	450	693
Fischer 13	Yes	B/D	15	1546	1434	MEMOUT			-C	29016288	1789	2262
Fischer 14	Yes	B/D	16	5727	2800	MEMOUT			-S2	MEMOUT		
Fischer 15	Yes	B/D	MEMOUT			MEMOUT			-S2	MEMOUT		
FDDI 40	Yes	B/D	0	63	495	0	72	729	-C	185535	2713	411
FDDI 50	Yes	B/D	0	109	495	0	624	2959	-C	TIMEOUT		
FDDI 75	Yes	B/D	0	360	934	MEMOUT			-C	TIMEOUT		
FDDI 100	Yes	B/D	0	1315	1779	MEMOUT			-S2	TIMEOUT		
Leader 5	No	F/D	30	30	182	30	190	1034		3257	0	37
Leader 6	No	F/D	38	4394	475	MEMOUT			-C -S2	21375	0	37
Leader 7	No	F/D	TIMEOUT			MEMOUT			-C	86645	1	40
Leader 5	Yes	F/D	97	105	209	83	417	1413		7398	0	37
Leader 6	Yes	F/D	TIMEOUT			MEMOUT				42482	1	38
Leader 7	Yes	F/D	TIMEOUT			MEMOUT				227253	4	41

Table 2. Comparison of our CMD-based prototype model checker with RED and UPPAAL.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. *HART*, vol. 1201 of *LNCS*, pp. 346–360, 1997.

[5] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. *SFM*, vol. 3185 of *LNCS*, pp. 200–236, 2004.

[6] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using clock difference diagrams. *CAV*, vol. 1633 of *LNCS*, pp. 341–353, 1999.

[7] D. Beyer. Improvements in BDD-based reachability analysis of timed automata. *FME*, vol. 2021 of *LNCS*, pp. 318–343, 2001.

[8] P. Bouyer. Untameable timed automata! *STACS*, vol. 2607 of *LNCS*, pp. 620–631, 2003.

[9] M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. *CAV*, vol. 1254 of *LNCS*, pp. 179–190, 1997.

[10] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[11] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[12] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. *TACAS*, vol. 1384 of *LNCS*, pp. 313–329. Springer, 1998.

[13] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. *AVMFSS*, vol. 407 of *LNCS*, pp. 197–212, 1989.

[14] D. L. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximation. *CAV*, vol. 939 of *LNCS*, pp. 409–422, 1995.

		Forward / Diagram			Forward / Compact			Backward / Diagram			Backward / Compact		
Benchmark	Sat	Steps	Time	Mem	Steps	Time	Mem	Steps	Time	Mem	Steps	Time	Mem
GPS 16	No	49	3	186	49	3	186	49	4	204	49	4	204
GPS 17	No	52	3	199	52	3	199	52	4	163	52	4	189
GPS 19	No	58	4	243	58	4	241	58	4	221	58	5	244
GPS 20	No	61	5	258	61	4	256	61	5	229	61	5	239
GPS 22	No	67	23	293	67	25	287	67	7	284	67	7	281
GPS 15	Yes	155	3	177	155	3	176	46	3	146	46	2	172
GPS 16	Yes	166	4	205	166	3	187	49	4	204	49	3	144
GPS 17	Yes	177	4	233	177	4	200	52	4	218	52	4	158
GPS 22	Yes	232	33	297	232	33	289	67	6	278	67	7	288
FlexRay 1	Yes	987	27	198	987	16	172	TIMEOUT			TIMEOUT		
FlexRay 33	Yes	11851	1246	570	11851	524	577	TIMEOUT			TIMEOUT		
FlexRay 34	Yes	12191	1373	542	12191	527	584	TIMEOUT			TIMEOUT		
FlexRay 100	Yes	34599	5401	1109	34599	695	761	TIMEOUT			TIMEOUT		
FlexRay 200	Yes	TIMEOUT			68551	2599	1299	TIMEOUT			TIMEOUT		
FlexRay 262	Yes	TIMEOUT			89603	2869	1482	TIMEOUT			TIMEOUT		
Fischer 11	No	MEMOUT			TIMEOUT			6	13	228	6	28	169
Fischer 12	No	MEMOUT			TIMEOUT			6	28	297	6	66	186
Fischer 19	No	MEMOUT			TIMEOUT			6	2864	3788	6	10301	2487
Fischer 20	No	MEMOUT			TIMEOUT			MEMOUT			TIMEOUT		
Fischer 11	Yes	MEMOUT			TIMEOUT			13	119	395	13	239	260
Fischer 12	Yes	MEMOUT			TIMEOUT			14	308	698	14	1501	563
Fischer 13	Yes	MEMOUT			TIMEOUT			15	1546	1434	15	6667	1079
Fischer 14	Yes	MEMOUT			TIMEOUT			16	5727	2800	TIMEOUT		
Fischer 15	Yes	MEMOUT			TIMEOUT			MEMOUT			TIMEOUT		
FDDI 40	Yes	MEMOUT			MEMOUT			0	63	495	0	64	495
FDDI 50	Yes	MEMOUT			MEMOUT			0	109	495	0	109	495
FDDI 75	Yes	MEMOUT			MEMOUT			0	360	934	0	327	934
FDDI 100	Yes	MEMOUT			MEMOUT			0	1315	1779	0	1317	1779
Leader 5	No	30	30	182	30	158	194	30	812	999	30	346	519
Leader 6	No	38	4394	475	TIMEOUT			TIMEOUT			38	7721	1836
Leader 7	No	TIMEOUT			TIMEOUT			TIMEOUT			TIMEOUT		
Leader 5	Yes	97	105	209	97	2694	170	87	974	999	87	563	519
Leader 6	Yes	TIMEOUT			TIMEOUT			TIMEOUT			TIMEOUT		
Leader 7	Yes	TIMEOUT			TIMEOUT			TIMEOUT			TIMEOUT		

Table 3. Comparison of different operation modes for our CMD-based prototype model checker.

[15] R. Ehlers, M. Gerke, and H.-J. Peter. Making the right cut in model checking data-intensive timed systems. *ICFEM*, vol. 6447 of *LNCS*, pp. 565–580, 2010.

[16] FlexRay Consortium. *FlexRay Communications System Protocol Specification Version 2.1 Rev. A*, 2005.

[17] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.

[18] K. G. Larsen, C. Weise, W. Yi, and J. Pearson. Clock Difference Diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[19] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[20] J. B. Møller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. *ENTCS*, 23(2), 1999.

[21] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, University of California, 1992.

[22] S. A. Seshia and R. E. Bryant. Unbounded, fully symbolic model checking of timed automata using boolean methods. *CAV*, vol. 2725 of *LNCS*, pp. 154–166, 2003.

[23] F. Somenzi. CUDD: CU Decision Diagram 2.4.2, 2009.

[24] F. Wang. Efficient verification of timed automata with BDD-like data structures. *STTT*, 6(1):77–97, 2004.

[25] S. Yamane and K. Nakamura. Development and evaluation of symbolic model checker based on approximation for real-time systems. *Systems and Computers in Japan*, 35(10):83–101, 2004.