Data Structure Choices for On-the-Fly Model Checking of Real-Time Systems

Peter Fontana
Department of Computer Science
University of Maryland, College Park
Email: pfontana@cs.umd.edu

Rance Cleaveland
Department of Computer Science
University of Maryland, College Park
Email: rance@cs.umd.edu

***Abstract*—This paper studies the performance of sparse-matrix-based data structures to represent *clock zones* (convex sets of clock values) in an on-the-fly predicate equation system model checker for timed automata. We analyze the impact of replacing the dense difference bound matrix (DBM) with both the linked-list CRDZone and array-list CRDArray data structure. From analysis on the paired-example-by-example differences in time performance, we infer the DBM is either competitive with or slightly faster than the CRDZone, and both perform faster than the CRDArray. Using similar analysis on space performance, we infer the CRDZone takes the least space, and the DBM takes less space than the CRDArray.**

I. Introduction

Automatic verification of real-time systems is undertaken using notations for verifiable programs and checkable specifications (see [1]–[6]). One common program notation is timed automata [7]. There are specification notations such as timed computation tree logic (TCTL) [1], [8] and timed extensions of a modal mu-calculus, including one in [3] and another given in [5]. Specifications in a timed modal mu-calculus can be written as lists of equations, known as timed modal equation systems [5], [6], [9]. For information on the untimed modal-mu calculi, see [10]–[12], and see [10], [11] for information on modal equation systems.

One approach to model checking timed automata with timed modal mu-calculus specifications is to use predicate equation systems (PES), which were invented independently by Groote and Willemse (as parameterized boolean equation systems) [13] and by Zhang and Cleaveland [6], [9]. Predicate equation systems provide a general framework for program models including parametric timed automata [6] and Presburger systems [14]. They also admit a natural on-the-fly approach to model checking based on proof search: a formula corresponding to the assertion that a timed automaton satisfies a mu-calculus property can be checked in a goal-driven fashion to determine its truth. Zhang and Cleaveland [6] demonstrated the efficiency of this approach *vis à vis* other real-time model-checking approaches.

In this paper we consider the special model checking case of timed automata and timed modal equation systems representing safety properties (also studied in [6]), for which there are still many opportunities for performance improvements. One component of such a model checker that has a noticeable influence on performance is the data structure for the sets of clock values. When analyzing safety properties, each desired set of clock values forms a convex set of clock values, or *clock zone* (see Definition 3). The conventional way to store a clock zone is as a *difference bound matrix* (DBM) (see Definition 4) [15], which stores the constraints as a matrix. This approach is used by UPPAAL [16] and described in [17]. To potentially save space and time, instead of representing the set of constraints as a matrix, one can represent the set as an ordered linked path of constraints where any clock difference not on the path has the implicit constraint $< \infty$. If we generalize this to allow for a union of zones to be represented by a directed graph of constraints (representing a tree of paths as opposed to a single path), we get a *clock restriction diagram* (CRD) [18]. If we compress the nodes to have them represent upper and lower bound constraints as well as explicitly encoding both valid and invalid paths, we get a *clock difference diagram* (CDD) [2]. These two structures are extensions of binary decision diagrams (BDDs) (see [19] for information).

To improve performance, we take the above idea of a linked implementation and incorporate the sparseness of the implementation of CRDs while simplifying (or shrinking) the structure to only support a single clock

Research supported by NSF Grant CCF-0820072.

zone (CRDs and CDDs in general can encode unions of clock zones). This simplified structure is a sparse sorted linked-list implementation of a DBM, the *CRDZone* (see Definition 5). We also implement an array-list version of the CRDZone, the *CRDArray* (see Definition 6). A CRDZone may be seen as a sparse sorted linked-list implementation of a DBM, and the CRDArray a sparse array-list implementation of the CRDZone. We examine the time and space performance of all three clock zone implementations: the matrix DBM, linked-list CRDZone and array-list CRDArray.

The contributions of this paper are:

- We run experiments comparing time and space performance of a model checker (on safety (reachability) properties) with the DBM, CRDZone and CRDArray data structure implementations.
- We formalize and extend the analysis style performed in the model checking experiments of [2], [6], [9], [18], [20], [21] by utilizing *paired data* (each implementation checked the same examples) and applying descriptive statistics on the paired example-by-example differences on time and space consumption. See Section VI for details on the statistics and Section VI-B for the analysis.

After analyzing the experimental results, for time performance we infer the DBM is either competitive with or slightly faster than the CRDZone and both perform faster than the CRDArray for the examples in this experiment. In terms of space, we infer the CRDZone takes up the least space, and the DBM and takes less space than the CRDArray for the examples in this experiment.

II. Program Model and Specifications

A. Timed Automata

A timed automaton encodes the behavior of a real-time system [7], [22].

Definition 1 (Clock constraint $\phi \in \Phi(CX)$)**.** Given a set of clocks CX, a *clock constraint* ϕ is constructed with the following grammar, where x_i is a clock and $c \in \mathbb{Z}$:

$$\phi ::= x_i < c \mid x_i \leq c \mid x_i > c \mid x_i \geq c \mid \phi \wedge \phi$$

$\Phi(CX)$ is the set of all possible clock constraints.

Definition 2 (Timed automaton)**.** A *timed automaton* $TA = (L, L_0, \Sigma, CX, I, E)$ is a tuple where:

- $\mathbf{L}$ is a finite set of locations with the initial set of locations $\mathbf{L_0} \subseteq L$.
- $\mathbf{\Sigma}$ is the set of actions and $\mathbf{CX}$ is the set of clocks.
- $\mathbf{I} : L \longrightarrow \Phi(CX)$ gives a clock constraint for each location l. $I(l)$ is called the *invariant* of l.
- $\mathbf{E} \subseteq L \times \Sigma \times \Phi(CX) \times 2^{CX} \times L$ is the set of *edges*. In an edge $e = (l, a, \phi, Y, l')$ from l to l' with action a, $\phi \in \Phi(CX)$ is the *guard* of e, and Y represents the set of clocks to *reset* to 0.

Some sources [6], [23] and our PES checker allow clock assignments ($x_1 := x_2$) in addition to clock resets on edges, other sources [17] allow constraints on clock differences and other sources [1] allow states to be labelled with atomic propositions that each state satisfies.

Timed automata use *clock valuations* $\nu \in \mathcal{V}$ ($\mathcal{V} = CX \longrightarrow \mathbb{R}^{\geq 0}$ is the set of all clock valuations), which at any moment stores a non-negative real value for each clock $x \in CX$. The semantics of a timed automaton are described as an infinite-state machine, where each state is a location-valuation pair (l, ν). Transitions represent either time advances or edge executions (performing an action). For a formal definition of the semantics of a timed automaton, see [7].

Example 1 (Example of a timed automaton)**.** Consider the timed automaton in Figure 1, which models a train in the generalized railroad crossing (GRC) protocol.

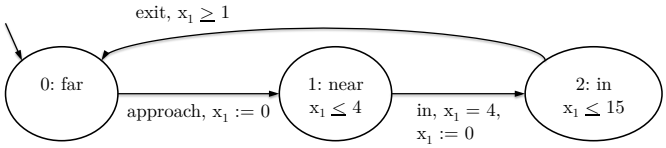

Fig. 1. Timed automaton TA_1, a model of a train in the generalized railroad crossing (GRC) protocol.

There are three locations—0: far (initial location), 1: near and 2: in, with one clock x_1. There are the actions $approach$, in and $exit$ in Σ. Here, location 1 has the invariant $x_1 \leq 4$ while 0 has no invariant. The edge $(1\text{: near}, in, x_1 = 4, \{x_1\}, 2\text{: in})$ has the guard $x_1 = 4$ and resets x_1 to 0.

B. Modal Equation Systems (MES)

We use a *modal equation system* (MES) to represent real-time temporal properties that timed automata can possess. A MES is an ordered list of equations with variables on the left hand side and basic timed temporal logical formulae on the right. Each equation involves a variable X, a basic formula ϕ and a greatest fixpoint (ν) or a least fixpoint (μ), and the equation is labeled with the fixpoint ($X \stackrel{\nu}{=} \phi$ or $X \stackrel{\mu}{=} \phi$). For a formal definition of MES syntax and semantics, see [6], [9].

Example 2 (Continuation of Example 1)**.** Again consider the timed automaton in Figure 1. The MES

$$X_1 \overset{\nu}{=} \quad \text{far} \ \wedge \forall([-](X_1)) \tag{1}$$

represents the safety property "the train is always in state 0: far", read as "the variable X_1 is the greatest fixpoint of being in state 0: far and for all time advances ($\forall$), for all next actions ($[-]$), X_1 is true." This is an invalid specification for the timed automaton because the execution

$$\begin{aligned}(0\text{: far}, [x_i = 0]) &\xrightarrow{2.5} (0\text{: far}, [x_i = 2.5]) \\ &\xrightarrow{approach} (1\text{: near}, [x_i = 0]) \xrightarrow{2} \dots \end{aligned} \tag{2}$$

reaches location 1: near and thus violates the property.

III. Data Structures for Clock Value Sets

A. Clock Zones

This definition of a clock zone is taken from [7], [19].

Definition 3 (Clock zone)**.** A *clock zone* is a convex combination of single-clock inequalities. Each clock zone can be constructed using the following grammar, where x_i and x_j are arbitrary clocks and $c \in \mathbb{Z}$:

$$\begin{aligned}Z ::= x_i < c \mid x_i \leq c \mid x_i > c \mid x_i \geq c \\ \mid x_i - x_j < c \mid x_i - x_j \leq c \mid Z \wedge Z\end{aligned} \tag{3}$$

Clock zones extend clock constraints with inequalities of clock differences. These inequalities are used for model checking even though clock difference inequalities are not used in timed automata. Moreover, in general, the representation of a clock zone is not unique.

Example 3. Let $z = 1 \leq x_1 < 3 \wedge 0 \leq x_2 \leq 5$. There is the *implicitly* encoded constraint $x_2 - x_1 \leq 4$. To see this, consider the longer path of constraints (x_0 is a dummy clock that always has value 0):

$$\begin{array}{rll} & x_2 - x_0 \leq 5 & (x_2 \leq 5) \\ + & x_0 - x_1 \leq -1 & (1 \leq x_1) \\ \hline & x_2 - x_1 \leq 4 & \end{array}$$

To provide a standardized, or canonical, form for clock zone representations, we use shortest path closure [17]. This form makes every implicit constraint explicit. This can be implemented in $O(n^3)$ time using Floyd-Warshall all-pairs shortest path algorithm, described in [24], [25]. Other standard forms exist [18], [20].

While converting to a canonical form takes a considerable amount of time, it is needed to simplify and standardize the algorithms for the zone operations including time successor ($succ(z)$) computations and subset checks. For time successor, having the zone in canonical form allows the time elapse operation to simply set all single-clock upper bound constraints to $< \infty$.

B. Clock Zone Data Structures: Difference Bound Matrix (DBM), CRDZone and CRDArray

One way to represent a clock zone is a *difference bound matrix (DBM)*, described in Definition 4. See [15], [17] for a more thorough description.

Definition 4 (Difference bound matrix (DBM))**.** Let $n - 1$ be the number of clocks. A *DBM* is an $n \times n$ matrix where entry (i, j) is the upper bound of the clock constraint $x_i - x_j$, represented as $x_i - x_j \leq u_{ij}$ or $x_i - x_j < u_{ij}$. The 0^{th} index is reserved for a dummy clock x_0, which is always 0, allowing bounds on single clocks to be represented by the clock differences $x_i - x_0$ and $x_0 - x_j$. See Figure 2 for a picture of the DBM structure and Example 4 for a concrete example.

Definition 5 (CRDZone)**.** A *CRDZone* is a sparse sorted linked-list representation of a clock zone. Each constraint is encoded like a constraint in a DBM, as an upper bound constraint on $x_i - x_j$, labeled as (i, j), with x_0 always being 0. The CRDZone has these properties:

1) Nodes are in lexicographical order of clock constraint: $(i_1, j_1) \prec (i_2, j_2)$ iff $i_1 < i_2$ or ($i_1 = i_2$ and $j_1 < j_2$).
2) The $(0, 0)$ node is always given to ensure a universal head node with an initial value of $x_0 - x_0 \leq 0$.
3) If a CRDZone node (i, j) is missing then the zone has an implicit constraint: $(i, i) : x_i - x_i \leq 0$ and $(i, j), i \neq j : x_i - x_j < \infty$.

See Figure 3 for a picture of the CRDZone structure and Example 4 for a concrete example.

This lexicographical ordering is the same ordering used in CDDs and CRDs [2], [18]. While the ordering

$$i \overbrace{\begin{bmatrix} \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & x_i - x_j \leq u_{ij} \end{bmatrix}}^{j}$$

Fig. 2. DBM: a matrix with constraint $x_i - x_j \leq u_{ij}$ in entry (i, j).

is conjectured to influence performance, this is the only ordering we implemented. Likewise, having different implicit constraints, such as making $x_i - x_0 \leq 0$ (for all i) implicit, is conjectured to influence performance.

Definition 6 (CRDArray)**.** The *CRDArray* is an array list implementation of the CRDZone. Thus, instead of using linked nodes, we use an array to store the nodes with the 0^{th} element being the head. We statically allocate the array to hold the maximum number of elements and store a back-pointer to the back of the array list. See Figure 4 for a picture of the CRDArray structure and Example 4 for a concrete example.

Using a dynamic allocation instead of our static allocation for the CRDArray array list is conjectured to save space at the expense of time.

Example 4 (Clock zone in various representations)**.** Consider the clock zone from Example 3, which is $z = 1 \leq x_1 < 3 \wedge 0 \leq x_2 \leq 5 \wedge x_2 - x_1 \leq 4$.

DBM representation of z:

$$\begin{bmatrix} x_0 - x_0 \leq 0 & x_0 - x_1 \leq -1 & x_0 - x_2 \leq 0 \\ x_1 - x_0 < 3 & x_1 - x_1 \leq 0 & x_1 - x_2 < \infty \\ x_2 - x_0 \leq 5 & x_2 - x_1 \leq 4 & x_2 - x_2 \leq 0 \end{bmatrix}$$

CRDZone representation of z:

$$\boxed{x_0 - x_0 \leq 0} \longrightarrow \boxed{x_0 - x_1 \leq -1} \longrightarrow \boxed{x_0 - x_2 \leq 0}$$

$$\longrightarrow \boxed{x_1 - x_0 < 3} \longrightarrow \boxed{x_2 - x_0 \leq 5} \longrightarrow \boxed{x_2 - x_1 \leq 4}$$

CRDArray representation of z:

$$[x_0 - x_0 \leq 0 | x_0 - x_1 \leq -1 | x_0 - x_2 \leq 0$$
$$| x_1 - x_0 < 3 | x_2 - x_0 \leq 5 | x_2 - x_1 \leq 4]$$

Remark 1 (On DBM vs. CRDZone and CRDArray methods). Due to the sparse implementation and removal of implicit nodes, the CRDZone and CRDArray can improve time by reducing the number of nodes, and thus the number of nodes looked at during a full traversal. This can speed up traversal-based algorithms such as intersect and subset check. However, algorithms like clock reset, emptiness check and canonical form use $O(1)$ access of middle nodes in DBMs (the CRDZone and CRDArray do not have $O(1)$ access for all nodes), resulting in a performance slowdown for those CRDZone and CRDArray methods. For space, the CRDZone and CRDArray can store fewer nodes but must store the explicit indices, resulting in more space per node.

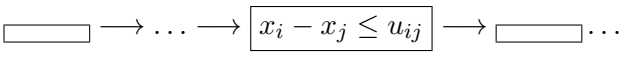

Fig. 3. CRDZone: A linked list with nodes in lexicographical order of constraint $x_i - x_j \leq u_{ij}$.

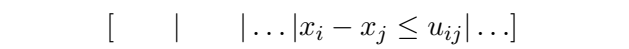

Fig. 4. CRDArray: An array list with nodes in lexicographical order of constraint $x_i - x_j \leq u_{ij}$.

IV. On-The-Fly Model Checking: Converting to a PES and Proof Search

Our model checker takes in a *predicate equation system* (PES) (taken from [9], [13]), which is a general framework representing logical expressions that involve fixpoints and first order quantifiers. We take a timed automaton and a MES and convert it to a PES. Currently the PES model checker can only check safety properties, which involve only greatest fixpoints in both the PES and the input MES. For more information on a PES, including its syntax, semantics and how to convert a timed automaton and a MES to a PES, see [6], [9].

The model checker takes the conclusion sequent (the sequent we wish to prove) and applies proof rules in a recursive goal-driven tree-like fashion on the premise sequents, trying to prove each premise sequent until it reaches a proof rule with no premise (called a leaf) or a circularity (a previously seen premise sequent). When checking a proof, we will often encounter circularity. In general, when the circularity reached is a greatest fixpoint, we can stop and declare the proof branch valid. For the formal conditions for circularity and the proof rules, see [6], [9].

V. Experiments: Various Data Structure Implementations

We compare the DBM implementation to the CRDZone and CRDArray implementations. Each implementation uses shortest path closure to compute canonical form. The only difference in the DBM, CRDZone and CRDArray versions is the data structure implementation.

Remark 2 (On our analysis approach). We ask: *what does it mean for an implementation to perform better than another?* **We consider consider *better* to be measured in the number (or percentage) of examples that one system outperforms another in.** The larger aim is for any implementation, if we were to know all the examples that it would run (including and beyond the experiment examples), we would *like* one implementation to perform

(strictly) better for at least 51% of this hypothetical set. This influences our analysis.

Given our meaning of *better* in Remark 2, we consider the median, 25% and 75% percentile values as *insights* into typical examples and use the histograms to get a bigger picture of the sample distribution of the performance differences for the experiment, and weigh these more heavily than the mean and standard deviation values. The mean and the standard deviation provide us with an alternative picture of the overall performance and give hints of either a unusual sample distribution (since in a symmetric distribution the mean equals the median) or the presence of potential outliers.

The benchmark choice was modeled off of [6], with the addition of a model of the generalized railroad crossing (GRC) protocol [26]. We also used all the protocols in [6], which are the Carrier Sense, Multiple Access with Collision Detection (CSMA/CD), the Fiber Distributed Data Interface (FDDI), Fischer's Mutual Exclusion (FISCHER), the Leader Election protocol (LEADER and LBOUND) and the PATHO Operating System (PATHOS) protocol, where each of these protocols is described some in [6]. There are 53 benchmarks that ran on each implementation.

Experiments were run on a Linux machine with a 3.4 GHz Intel Pentium 4 Dual Processor (each a single core) with 4 GB RAM. Time and space measurements (maximum space used) were made using the `memtime` (http://www.update.uu.se/~johanb/memtime/) tool (using time `elapsed` and `Max VSize`). The data tables are in the Appendix.

TABLE I
DESCRIPTIVE STATISTICS FOR PAIRED DBM − (MINUS) CRDZONE EXAMPLES, FOR TIME (S) AND SPACE (MB).

Statistic	DBM − CRDZone (Time)	DBM − CRDZone (Space)
Mean	-67.55	34.96
Standard Deviation	428.35	212.65
25% Percentile	-1.24	0.00
Median	0.00	1.85
75% Percentile	0.06	25.70

TABLE II
DESCRIPTIVE STATISTICS FOR PAIRED DBM − (MINUS) CRDARRAY EXAMPLES, FOR TIME (S) AND SPACE (MB).

Statistic	DBM − CRDArray (Time)	DBM − CRDArray (Space)
Mean	-112.95	-47.75
Standard Deviation	655.65	235.63
25% Percentile	-3.16	-20.54
Median	-0.29	-2.81
75% Percentile	0.00	-0.01

TABLE III
DESCRIPTIVE STATISTICS FOR PAIRED CRDZONE − (MINUS) CRDARRAY EXAMPLES, FOR TIME (S) AND SPACE (MB).

Statistic	CRDZone − CRDArray (Time)	CRDZone − CRDArray (Space)
Mean	-45.40	-82.71
Standard Deviation	229.06	160.91
25% Percentile	-2.02	-52.67
Median	-0.21	-19.35
75% Percentile	-0.03	-1.63

VI. STATISTICS, ANALYSIS AND DISCUSSION

A. Histograms and Descriptive Statistics

Running the different data structure implementations with the same examples yields *paired data*. Hence, we can take the two implementations and pair them example-by-example on their time and space differences to analyze their performance. When we pair the DBM − CRDZone samples, we take the DBM measurement and subtract the CRDZone measurement for the same example to get a DBM − CRDZone paired data point. For instance, the MUX-5-a paired point is -0.92s, 1.94MB, since the DBM point is 1.22s, 14.67MB, and the CRDZone point is 2.14s, 12.73MB. Pairings are likewise done to obtain the paired samples for DBM − CRDArray and CRDZone − CRDArray. For more information, see a Statistics text such as [27].

Tables I, II and III contain descriptive statistics on the paired difference in example-by-example performance of the DBM, CRDZone and CRDArray. Figures 5, 6 and 7 have histograms that plot the overall time and space differences between the DBM, CRDZone and CRDArray implementations. Bin colors and are changed to help more easily find the -0.001 to 0.001 (equal performance, since our measurement precision is 0.01 units), and -0.25 to -0.001 and 0.001 to 0.25 bins (slight differences).

We do not use 95% confidence intervals, paired two-sample hypothesis (z) tests or ANOVA (Analysis of Variance) because the independence assumption of the samples (the example benchmarks) does not hold. Furthermore, we do not use a Wilcoxon signed-rank test for the median because the symmetry assumption of the distribution is not believed to hold, and thus we cannot analyze the hypothetical benchmark distribution referred to in Remark 2. We do use paired sampling since we have its only requirement—perfect correlation of the samples. More information is in [27].

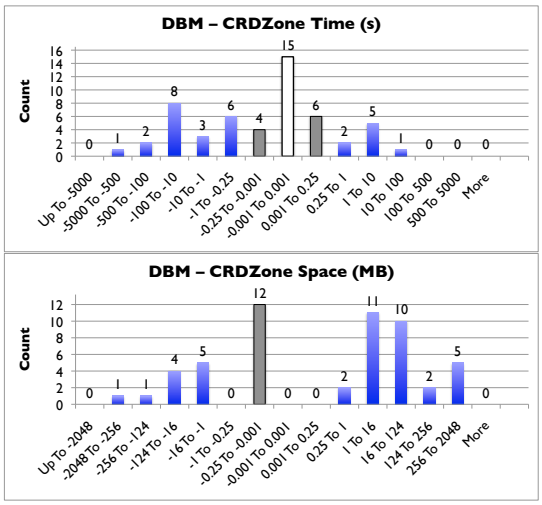

Fig. 5. Histograms comparing the DBM − (minus) CRDZone time (s) (top) and space (MB) (bottom) differences.

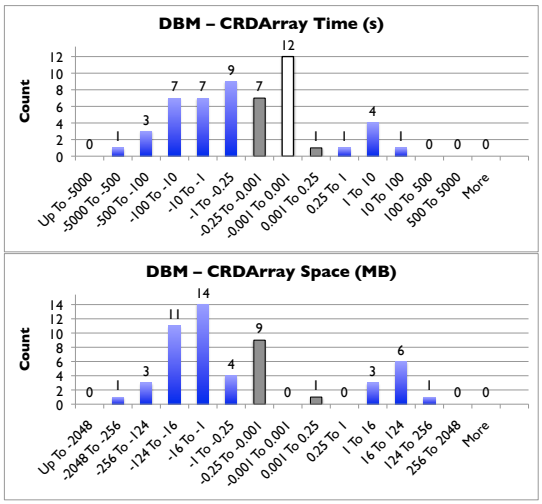

Fig. 6. Histograms comparing the DBM − (minus) CRDArray time (s) (top) and space (MB) (bottom) differences.

B. Analysis of Results

1) DBM vs. CRDZone: The CRDZone performs slower for 45% of the tested examples (at least as slow for 74%) with a median difference of 0.00s slower, while the CRDZone has a mean difference of 67.55s slower. Thus, we infer the CRDZone is either slightly slower or competitive to the DBM for this benchmark, but due to insufficient evidence (45% of the examples is not enough) do not infer that the DBM performs strictly faster than the CRDZone.

The CRDZone takes less space for 57% of the tested examples (at most as much space for 57%) with a median amount of 1.85MB less space and a mean amount of 34.96MB less space. The CRDZone takes at least

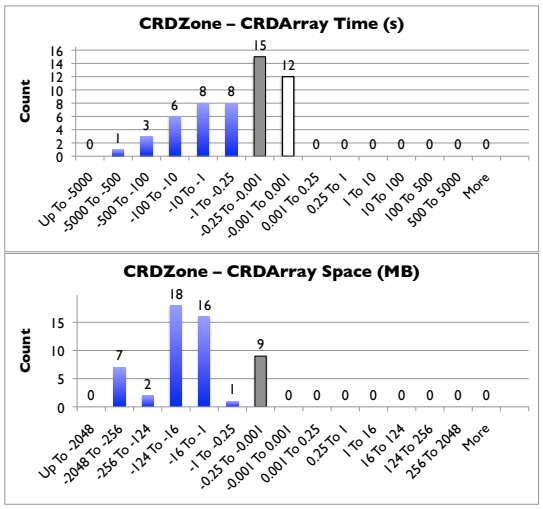

Fig. 7. Histograms comparing the CRDZone − (minus) CRDArray time (s) (top) and space (MB) (bottom) differences.

0.25MB less space for 28 such examples and more than 0.25MB space for only 11 examples. Thus (even though 57% is not a large majority), we infer the CRDZone takes less space overall for this benchmark.

2) DBM vs. CRDArray: The CRDArray performs slower for 64% of the tested examples (at least as slow for 89%) with a median difference of 0.29s slower and a mean difference of 112.95s slower. Thus we infer the CRDArray performs slower overall for this benchmark.

The CRDArray takes more space for 77% (at least as much space for 77%) of the examples with a median amount of 2.81MB more space and mean amount of 47.75MB more. Thus we infer the CRDArray takes more space overall for this benchmark.

3) CRDZone vs. CRDArray: The CRDArray performs slower for 77% of the tested examples (at least as slow for 100%) with a median difference of 0.21s slower and a mean difference of 45.40s slower. Thus we infer the CRDArray is slower overall for this benchmark.

The CRDArray takes more space for 100% of the examples with a median amount of 19.35MB more space and a mean amount of 82.71MB more. Thus we infer the CRDArray takes more space overall for this benchmark.

C. Discussion of Results

The CRDZone and CRDArray method that converts zones to canonical form was implemented using array-based algorithms, where it temporarily converts the clock zone to and from a matrix (the algorithm is similar to a DBM algorithm for those methods). It is possible that performance can be improved by trying an algorithm that

does not require copying to and from a matrix. All time vs. space tradeoffs were taken to save time.

Furthermore, we ran a CRDZone execution twice (first execution reported) and compared the distribution of their differences to get an idea of noise and/or measurement error. The differences in the histograms are larger than the differences of the noisy implementation, so thus we suspect the differences in performances are due to more than just uncertain measurement/noisy data. However, slight differences ($\leq$ 0.25s or $\leq$ 0.25MB) may be due to execution noise or examples that require very little time and/or space. The PATHOS-7-b is one such resource-light example, taking at most 0.10s and 2.89MB for each implementation. In contrast, the MUX-7-a example is resource-heavy, being the only example to takes more than 2000s for each implementation.

When profiling the implementations using `gprof` (http://www.gnu.org/s/binutils/), CRDZones take more time than DBMs for clock resets and emptiness checks, but the invariant checking method (mostly clock zone intersections) takes less time for CRDZones than DBMs.

From the data (see the Appendix), we notice that the data structure implementation choice has a noticeable influence on model checking performance for specific examples. To see this, consider the examples MUX-7-a, where the DBM finished 3118.94 seconds earlier than the CRDZone, and LEADER-100-c, where the CRDZone checked the example in 55% of the time required for the DBM. There are also noticeable differences relative to the CRDArray.

D. Related Work

A different way of modeling programs using discrete-time is discussed in [4]. PES are in [9], and they have been used to model check various systems in [6], [9], [14]. Difference bound matrices originated from [15] and are used in various studies such as in those in [17], [23]. Other tools that can model check timed automata (safety and liveness properties) include *KRONOS* [23], *UPPAAL* [16], *RED* [18] and *Rabbit* [28]. We built upon Zhang's implementation using predicate equation systems in [6], [9], which supports safety properties. A similar experiment involving using a reduced canonical form is in [20], which focuses on the influence of different standard clock zone forms instead of comparing list vs. array implementations.

A remark on a sparse DBM representation saving space, with neither a mention on time nor experimental data, is given in [17]. There is an experiment comparing CDDs to another BDD-like structure used by *Rabbit* in [21], but this experiment compares data structures for non-convex sets of clock valuations (unions of clock zones), and the comparison is across different tools with different model checking approaches, and not the same tool with different data structures.

VII. Conclusions and Future Work

Here are our conclusions from the experiment:

1) **Time:** (DBM $\leq_t$ CRDZone) $<_t$ CRDArray). For this benchmark, we infer that the DBM is either competitive with or slightly faster than the CRDZone and both perform faster than the CRDArray. There is insufficient evidence to conclude that the DBM is strictly faster.
2) **Space:** (CRDZone $<_s$ DBM) $<_s$ CRDArray. For this benchmark, we infer that the CRDZone takes the least amount of space and the DBM takes less space than the CRDArray for this experiment.

For potential reasons for performance differences and analysis of some theoretical differences, see Remark 1.

Future work is to model check least fixpoint (μ) equations, to implement all of the proof rules given in [6], [9] and to model check any PES. Comparing lists of DBMs to a CRD or CDD to improve the performance of the expanded implementation is future work, as well as comparing different kinds of standard forms and different CRDZone node orderings.

Acknowledgements

We thank Dezhuang Zhang for providing the code base [6] and for his insights.

References

[1] R. Alur, C. Courcoubetis, and D. Dill, "Model-Checking in Dense Real-Time," *Inf. Comput.*, vol. 104, pp. 2–34, 1993.

[2] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi, "Efficient Timed Reachability Analysis Using Clock Difference Diagrams," in *CAV '99*, vol. 1633. Springer Berlin / Heidelberg, 1999.

[3] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-time Systems," *Inf. Comput.*, vol. 111, pp. 193–244, 1994.

[4] L. Lamport, "Real-Time Model Checking Is Really Simple," *Correct Hardware Design and Verification Methods*, pp. 162 –175, 2005.

[5] O. V. Sokolsky and S. A. Smolka, "Local model checking for real-time systems," in *CAV '95*. Springer-Verlag, 1995, pp. 211–224.

[6] D. Zhang and R. Cleaveland, "Fast Generic Model-Checking for Data-Based Systems," in *FORTE*, F. Wang, Ed., vol. 3731. Springer, 2005, pp. 83–97.

[7] R. Alur, "Timed Automata," in *CAV '99*. London, UK: Springer-Verlag, 1999, pp. 8–22.

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

[9] D. Zhang and R. Cleaveland, "Fast On-the-Fly Parametric Real-Time Model Checking," in *RTSS '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 157–166.

[10] J. Bradfield and C. Stirling, "Local Model Checking for Infinite State Spaces," *Theor. Comput. Sci.*, vol. 96, no. 1, pp. 157–174, 1992.

[11] R. Cleaveland and B. Steffen, "A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus," *Form. Methods Syst. Des.*, vol. 2, no. 2, pp. 121–147, 1993.

[12] E. A. Emerson and C. L. Lei, "Efficient Model Checking in Fragments of the Propositional Mu-Calculus," in *LICS '86*. IEEE Computer Society Press, 1986, pp. 267–278.

[13] J. F. Groote and T. A. Willemse, "Parameterised Boolean Equation Systems," *Theoretical Computer Science*, vol. 343, no. 3, pp. 332 – 369, 2005.

[14] D. Zhang and R. Cleaveland, "Efficient Temporal-Logic Query Checking for Presburger Systems," in *ASE '05*. New York, NY, USA: ACM, 2005, pp. 24–33.

[15] D. L. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," in *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*. London, UK: Springer-Verlag, 1990, pp. 197–212.

[16] G. Behrmann, A. David, and K. G. Larsen, "A Tutorial on Uppaal," in *Formal Methods for the Design of Real-Time Systems*, vol. 3185. Springer Berlin / Heidelberg, 2004, pp. 200–236.

[17] J. Bengtsson and W. Yi, "Timed Automata: Semantics, Algorithms," in *Lecture Notes in Computer Science*, vol. 3098. Springer, 2004, pp. 87–124.

[18] F. Wang, "Efficient Verification of Timed Automata with BDD-Like Data-Structures," in *VMCAI 2003*. London, UK: Springer-Verlag, 2003, pp. 189–205.

[19] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, 1999.

[20] K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction," in *RTSS 1997*. IEEE Computer Society, December 1997, pp. 14–24.

[21] D. Beyer and A. Noack, "Can decision diagrams overcome state space explosion in real-time verification?" in *FORTE 2003*. Springer Berlin / Heidelberg, 2003, vol. 2767, pp. 193–208.

[22] E.-R. Olderog and H. Dierks, *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, 2008.

[23] S. Yovine, "Model Checking Timed Automata," in *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*. London, UK: Springer-Verlag, 1998, pp. 114–152.

[24] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[26] C. L. Heitmeyer, B. G. Labaw, and R. D. Jeffords, "A Benchmark for Comparing Different Approaches for Specifying and Verifying Real-Time Systems," Naval Research Laboratory, Tech. Rep. ADA462244, 1993.

[27] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*, 6th ed. Duxbury Press, 2003.

[28] D. Beyer, C. Lewerentz, and A. Noack, "Rabbit: A tool for bdd-based verification of real-time systems," in *CAV '03*. Springer Berlin / Heidelberg, 2003, vol. 2725, pp. 122–125.

TABLE IV
EXPERIMENT RESULTS—A EXAMPLES—TIME (S): CORRECT SYSTEM, CORRECT SPECIFICATION.

Example	DBM	CRDZone	CRDArray
CSMACD-3-a	**0.10**	0.20 (200%)	0.20 (200%)
CSMACD-4-a	**3.16**	4.48 (142%)	6.50 (206%)
FDDI-20-a	**2.04**	3.03 (149%)	4.66 (228%)
FDDI-40-a	**58.49**	79.2 (135%)	126.82 (217%)
FDDI-50-a	**169.66**	230.7 (136%)	370.71 (219%)
MUX-5-a	**1.22**	2.14 (175%)	2.75 (225%)
MUX-6-a	**35.49**	74.44 (210%)	98.08 (276%)
MUX-7-a	**2623.61**	5742.55 (219%)	7383.73 (281%)
LEADER-6-a	**0.41**	0.71 (173%)	0.92 (224%)
LEADER-7-a	**12.99**	25.89 (199%)	34.22 (263%)
LBOUND-6-a	**0.51**	1.02 (200%)	1.32 (259%)
LBOUND-7-a	**17.36**	37.07 (214%)	49.64 (286%)
PATHOS-4-a	**13.7**	35.23 (257%)	50.58 (369%)
GRC-3-a	**0.92**	1.63 (177%)	2.12 (230%)
GRC-4-a	**252.05**	431.63 (171%)	748.01 (297%)

TABLE V
EXPERIMENT RESULTS—A EXAMPLES—SPACE (MB): CORRECT SYSTEM, CORRECT SPECIFICATION.

Example	DBM	CRDZone	CRDArray
CSMACD-3-a	**2.88**	7.55 (262%)	11.02 (382%)
CSMACD-4-a	209.97	**104.47** (50%)	179.53 (86%)
FDDI-20-a	**5.96**	9.00 (151%)	13.57 (227%)
FDDI-40-a	**27.55**	57.24 (208%)	100.30 (364%)
FDDI-50-a	**53.91**	116.79 (217%)	209.29 (388%)
MUX-5-a	14.57	**12.73** (87%)	18.55 (127%)
MUX-6-a	**84.05**	116.35 (138%)	168.38 (200%)
MUX-7-a	**625.42**	1667.94 (267%)	2302.39 (368%)
LEADER-6-a	**3.57**	6.59 (185%)	7.82 (219%)
LEADER-7-a	**20.98**	104.02 (496%)	133.39 (636%)
LBOUND-6-a	**3.93**	8.66 (220%)	10.39 (264%)
LBOUND-7-a	**27.89**	157.54 (565%)	199.99 (717%)
PATHOS-4-a	40.73	**38.11** (94%)	57.45 (141%)
GRC-3-a	10.48	**7.87** (75%)	11.23 (107%)
GRC-4-a	318.22	**220.64** (69%)	355.02 (112%)

APPENDIX
EXPERIMENTAL DATA

For the experiments, we use three kinds of examples:

- **Valid A Examples (in Tables IV and V):** Correct system implementations with valid safety specifications.
- **Invalid B Examples (in Tables VI and VII):** A examples with invalid specifications.
- **Invalid C Examples (in Tables VIII and IX):** A examples with buggy implementations of the systems that do not satisfy the A specifications.

TABLE VI
EXPERIMENT RESULTS—B EXAMPLES—TIME (S): CORRECT SYSTEM, INVALID SPECIFICATION.

Example	DBM	CRDZone	CRDArray
CSMACD-4-b	**0.10**	**0.10** (100%)	0.20 (200%)
CSMACD-5-b	**0.51**	**0.51** (100%)	0.71 (139%)
CSMACD-6-b	3.35	**2.73** (81%)	3.97 (119%)
FDDI-30-b	**1.53**	**1.53** (100%)	2.23 (146%)
FDDI-40-b	4.66	**4.58** (98%)	6.60 (142%)
FDDI-60-b	8.64	**5.07** (59%)	5.48 (63%)
MUX-20-b	**0.41**	**0.41** (100%)	0.51 (124%)
MUX-30-b	0.92	**0.91** (99%)	1.21 (132%)
MUX-40-b	1.93	**1.73** (90%)	2.23 (116%)
LEADER-10-b	**0.10**	**0.10** (100%)	**0.10** (100%)
LEADER-20-b	**0.10**	0.20 (200%)	0.20 (200%)
LBOUND-10-b	**0.10**	**0.10** (100%)	0.20 (200%)
LBOUND-40-b	6.82	17.46 (256%)	29.54 (433%)
PATHOS-7-b	**0.10**	**0.10** (100%)	**0.10** (100%)
PATHOS-8-b	**0.10**	**0.10** (100%)	**0.10** (100%)
PATHOS-9-b	**0.10**	**0.10** (100%)	**0.10** (100%)
GRC-3-b	**0.10**	**0.10** (100%)	**0.10** (100%)
GRC-4-b	**0.51**	0.61 (120%)	0.82 (161%)
GRC-5-b	**9.75**	13.4 (137%)	19 (195%)

TABLE VII
EXPERIMENT RESULTS—B EXAMPLES—SPACE (MB): CORRECT SYSTEM, INVALID SPECIFICATION.

Example	DBM	CRDZone	CRDArray
CSMACD-4-b	**2.88**	2.89 (100%)	13.67 (474%)
CSMACD-5-b	144.14	**72.38** (50%)	123.52 (86%)
CSMACD-6-b	1134.30	**553.21** (49%)	961.90 (85%)
FDDI-30-b	9.60	**9.06** (94%)	19.08 (199%)
FDDI-40-b	17.19	**16.03** (93%)	39.00 (227%)
FDDI-60-b	27.85	**14.53** (52%)	63.52 (228%)
MUX-20-b	19.37	**11.15** (58%)	16.96 (88%)
MUX-30-b	28.28	**16.87** (60%)	28.58 (101%)
MUX-40-b	43.01	**21.85** (51%)	42.81 (100%)
LEADER-10-b	**2.88**	2.89 (100%)	2.89 (100%)
LEADER-20-b	**2.88**	4.59 (159%)	5.69 (197%)
LBOUND-10-b	**2.88**	2.89 (100%)	3.38 (117%)
LBOUND-40-b	18.29	**15.23** (83%)	30.73 (168%)
PATHOS-7-b	**2.88**	2.89 (100%)	2.89 (100%)
PATHOS-8-b	**2.88**	2.89 (100%)	2.89 (100%)
PATHOS-9-b	**2.88**	2.89 (100%)	2.89 (100%)
GRC-3-b	**2.88**	2.89 (100%)	2.89 (100%)
GRC-4-b	58.74	**32.08** (55%)	53.42 (91%)
GRC-5-b	717.21	**379.44** (53%)	648.00 (90%)

TABLE VIII
EXPERIMENT RESULTS—C EXAMPLES—TIME (S): BUGGY SYSTEM, CORRECT SPECIFICATION.

Example	DBM	CRDZone	CRDArray
CSMACD-6-c	0.51	**0.41** (80%)	0.51 (100%)
CSMACD-7-c	2.03	**1.82** (90%)	2.03 (100%)
CSMACD-8-c	9.55	**8.42** (88%)	9.55 (100%)
FDDI-30-c	0.51	**0.41** (80%)	0.41 (80%)
FDDI-40-c	1.52	**0.92** (61%)	1.01 (66%)
FDDI-60-c	6.71	**3.98** (59%)	4.17 (62%)
MUX-6-c	**139.02**	258.32 (186%)	401.84 (289%)
LEADER-60-c	6.81	**3.96** (58%)	4.06 (60%)
LEADER-70-c	14.42	**8.12** (56%)	8.13 (56%)
LEADER-100-c	82.94	**45.78** (55%)	45.88 (55%)
LBOUND-6-c	**0.10**	**0.10** (100%)	0.20 (200%)
LBOUND-7-c	**0.61**	0.81 (133%)	1.12 (184%)
LBOUND-8-c	**12.48**	32.00 (256%)	52.9 (424%)
PATHOS-5-c	**0.10**	**0.10** (100%)	**0.10** (100%)
PATHOS-6-c	**0.10**	**0.10** (100%)	**0.10** (100%)
PATHOS-7-c	**0.10**	**0.10** (100%)	**0.10** (100%)
GRC-3-c	**0.10**	**0.10** (100%)	**0.10** (100%)
GRC-4-c	**0.51**	0.81 (159%)	1.02 (200%)
GRC-5-c	**9.65**	13.31 (138%)	18.88 (196%)

TABLE IX
EXPERIMENT RESULTS—C EXAMPLES—SPACE (MB): BUGGY SYSTEM, CORRECT SPECIFICATION.

Example	DBM	CRDZone	CRDArray
CSMACD-6-c	85.32	**18.53** (22%)	79.30 (93%)
CSMACD-7-c	337.43	**191.84** (57%)	320.89 (95%)
CSMACD-8-c	1369.07	**787.75** (58%)	1338.62 (98%)
FDDI-30-c	4.88	**4.60** (94%)	9.93 (203%)
FDDI-40-c	9.55	**6.41** (67%)	19.63 (206%)
FDDI-60-c	24.07	**14.24** (59%)	55.57 (231%)
MUX-6-c	1607.73	**1047.64** (65%)	1723.25 (107%)
LEADER-60-c	29.24	**10.79** (37%)	63.66 (218%)
LEADER-70-c	51.18	**15.30** (30%)	108.80 (213%)
LEADER-100-c	203.89	**40.65** (20%)	431.71 (212%)
LBOUND-6-c	**2.88**	2.89 (100%)	4.48 (155%)
LBOUND-7-c	12.52	**10.34** (83%)	14.42 (115%)
LBOUND-8-c	75.66	**59.23** (78%)	90.48 (120%)
PATHOS-5-c	**2.88**	2.89 (100%)	2.89 (100%)
PATHOS-6-c	**2.88**	2.89 (100%)	2.89 (100%)
PATHOS-7-c	**2.88**	2.89 (100%)	2.89 (100%)
GRC-3-c	**2.88**	2.89 (100%)	2.89 (100%)
GRC-4-c	58.74	**35.94** (61%)	59.47 (101%)
GRC-5-c	717.29	**379.35** (53%)	647.94 (90%)

The experimental data for the 53 example benchmarks is provided in Tables IV, V, VI, VII, VIII and IX, with the best entry(ies) in each row **bolded** and percentage change relative to the DBM, to the nearest %, in parenthesis. Time data is given to the nearest 0.01s (second) and space data is given to the nearest 0.01MB (Megabyte). Given the percentage rounding, sometimes an example with slightly different performance may still have a 100% value.