Timed Automata Verification using Interpolants

Tamás Tóth* and István Majzik†
Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
Fault Tolerant Systems Research Group
Email: *totht@mit.bme.hu, †majzik@mit.bme.hu

***Abstract*—The behavior of safety critical systems is often constrained by real time requirements. To model time dependent behavior and enable formal verification, timed automata are widely used as a formal model of such systems. However, due to real-valued clock variables, the state space of these systems is not finite, thus model checkers for timed automata are usually based on zone abstraction. The coarseness of the abstraction profoundly impacts performance of the analysis. In this paper, we propose a lazy abstraction algorithm based on interpolation for zones to enable efficient reachability checking for timed automata. In order to efficiently handle zone abstraction, we define interpolation in terms of difference bound matrices. We extend the notion of zone interpolants to sequences of transitions of a timed automaton, thus enabling the use of zone interpolants for abstraction refinement.**

I. Introduction

The behavior of safety critical systems is often time dependent. Timed automata [1] are widely used in the formal modeling and verification of such systems. Model checkers for timed automata, e.g. UPPAAL [2], are most commonly based on zone abstraction [3], used in conjunction with an extrapolation operator to ensure termination and accelerate convergence. The quality of the abstraction obtained this way is imperative for performance, and many different techniques have been proposed, including abstractions based on *LU*-bounds [4], [5] and region closure [6]. In [7], a lazy abstraction scheme is proposed for *LU*-abstraction where the abstraction is refined only if it enables a spurious step in the abstract system. This results in an abstraction that is based not only on structural properties of the automaton (like constants appearing in guards) but the state itself.

On the other hand, in SAT/SMT based model checking, interpolation [8] is a commonly used technique [9], [10] for building safe abstractions of systems. As interpolants can be used to extract coarse explanations for the infeasibility of error paths, they can be used for abstraction refinement [11] in a way relevant for the property to be proved. Algorithms for interpolant generation have been proposed for several first order theories, including linear arithmetic ($\mathcal{LA}(\mathbb{Q})$) and difference logic ($\mathcal{DL}(\mathbb{Q})$) over the rationals [12]. To better support certain verification tasks, classical interpolation has been generalized in various ways, e.g. to sequence interpolants [10].

In this paper, we propose a *lazy abstraction* algorithm similar to [7] for reachability checking of timed automata based on interpolation. In order to efficiently handle zone abstraction, we define *interpolation for zones* represented in terms of difference bound matrices. Moreover, we extend the notion of zone interpolants to *sequences of transitions of a timed automaton*, thus enabling the use of zone interpolants for abstraction refinement.

The rest of the paper is organized as follows. In Section II, we define the notations used throughout the paper, and present the theoretical background of our work. In Section III we present our work: we give an interpolation algorithm for zones represented as DBMs (Section III-A), describe how inductive sequences of zones can be computed for timed automata using zone interpolation (Section III-B), and outline how such sequences can be used to speed up convergence for reachability checking of timed automata (Section III-C). Section IV describes the implementation and a preliminary evaluation of the proposed algorithm. Finally, conclusions are given in Section V.

II. Background and Notations

In this section, we define the notations used throughout the paper, and outline the theoretical background of our work.

A. Timed Automata

Timed automata [1] is a widely used formalism for modeling real-time systems. A *timed automaton* (TA) is a tuple $(Loc, Clock, \hookrightarrow, Inv, \ell_0)$ where

- Loc is a finite set of locations,
- $Clock$ is a finite set of clock variables,
- $\hookrightarrow \subseteq Loc \times ClockConstr \times \mathcal{P}(Clock) \times Loc$ is a set of transitions where for $(\ell, g, R, \ell') \in \hookrightarrow$, g is a guard and R is a set containing clocks to be reset,
- $Inv : Loc \to ClockConstr$ is a function that maps to each location an invariant condition over clocks, and
- $\ell_0 \in Loc$ is the initial location.

Here, $ClockConstr$ denotes the set of *clock constraints*, that is, difference logic formulas of the form

$$\varphi \quad ::= \quad \mathbf{true} \mid x_i \prec c \mid x_i - x_j \prec c \mid \varphi \wedge \varphi$$

where $x_i, x_j \in Clock$, $\prec \in \{<, \leq, \doteq\}$ and $c \in \mathbb{Z}$.

The operational semantics of a TA can be defined as a labeled transition system $(S, Act, \to, I)$ where

- $S = Loc \times Eval$ is the set of states,
- $I = \{(\ell_0, \eta_0)\}$ where $\eta_0(x) = 0$ for all $x \in Clock$ is the set of initial states,

*This work was partially supported by Gedeon Richter's Talentum Foundation (Gyömrői út 19-21, 1103 Budapest, Hungary).

- $Act = \mathbb{R}_{\geq 0} \cup \{\alpha\}$, where α denotes discrete transitions,
- and a transition $t \in \rightarrow$ of the transition relation $\rightarrow \subseteq S \times Act \times S$ is either a delay transition that increases all clocks by a value $\delta \geq 0$:

$$\frac{\ell \in Loc \quad \delta \geq 0 \quad \eta' = Delay_\delta(\eta) \quad \eta' \models Inv(\ell)}{(\ell, \eta) \xrightarrow{\delta} (\ell, \eta')}$$

or a discrete transition:

$$\frac{\ell \xhookrightarrow{g,R} \ell' \quad \eta \models g \quad \eta' = Reset_R(\eta) \quad \eta' \models Inv(\ell')}{(\ell, \eta) \xrightarrow{\alpha} (\ell', \eta')}$$

Here, $Eval$ denotes the set of clock valuations, that is, mappings of clocks to real values. For a real number $\delta \geq 0$ the function $Delay_\delta : Eval \to Eval$ assigns to a valuation $\eta \in Eval$ a valuation $Delay_\delta(\eta)$ such that for all $x \in Clock$

$$Delay_\delta(\eta)(x) = \eta(x) + \delta$$

Moreover, $Reset_R(\eta)$ models the effect of resetting all clocks in R to 0 for a valuation $\eta \in Eval$:

$$Reset_\eta(R)(x) = \begin{cases} 0 & \text{if } x \in R \\ \eta(x) & \text{otherwise} \end{cases}$$

For these functions, we define images and preimages in the usual way.

As the concrete semantics of a timed automaton is infinite due to real valued clock variables, model checkers are often based on a symbolic semantics defined in terms of zones. A *zone* Z is the solution set of a clock constraint φ, that is $Z = [\![\varphi]\!] = \{\eta \mid \eta \models \varphi\}$. The following functions are operations over zones:

- *Conjunction*: $Z \cap g = Z \cap [\![g]\!]$,
- *Future*: $Z^\uparrow = \bigcup_{\delta \geq 0} Delay_\delta(Z)$,
- *Past*: $Z_\downarrow = \bigcup_{\delta \geq 0} Delay_\delta^{-1}(Z)$,
- *Reset*: $R(Z) = Reset_R(Z)$,
- *Inverse reset*: $R^{-1}(Z) = Reset_R^{-1}(Z)$.

For timed automata, an exact pre- and postimage operator can be defined over zones. Let $e = (\ell, g, R, \ell')$ be an edge of a TA with invariants Inv. Then

- $\mathbf{post}(Z, e) = R(Z^\uparrow \cap Inv(\ell) \cap g) \cap Inv(\ell')$,
- $\mathbf{pre}(Z, e) = (R^{-1}(Z \cap Inv(\ell')) \cap g \cap Inv(\ell))_\downarrow$.

B. Difference Bound Matrices

Clock constraints and thus zones can be efficiently represented by difference bound matrices.

A *bound* is either ∞ or of the form $(m, \prec)$ where $m \in \mathbb{Z}$ and $\prec \in \{<, \leq\}$. Difference bounds can be totally ordered by "strength", that is, $(m, \prec) < \infty$, $(m_1, \prec_1) < (m_2, \prec_2)$ iff $m_1 < m_2$ and $(m, <) < (m, \leq)$. Moreover the sum of two bounds is defined as $b + \infty = \infty$, $(m_1, \leq) + (m_2, \leq) = (m_1 + m_2, \leq)$ and $(m_1, <) + (m_2, \prec) = (m_1 + m_2, <)$.

Let $Clock = \{x_1, x_2, \ldots, x_n\}$ and x_0 a reference clock with constant value 0. A *difference bound matrix* (DBM) over $Clock$ is a square matrix D of bounds of order $n + 1$ where an element $D_{ij} = (m, \prec)$ represents the clock constraint $x_i - x_j \prec m$. We say that a clock x_i is constrained in D iff $D_{ii} < (0, \leq)$, or there exists an index $j \neq i$ such that $D_{ij} < \infty$ or $D_{ji} < \infty$.

We denote by $[\![D]\!]$ the zone represented by the constraints in D. When it is unambiguous from the context, we omit the brackets. D is *consistent* iff $D \neq \emptyset$. D is *closed* iff constraints in it can not be strengthened without losing solutions, formally, iff $D_{ij} \leq D_{ik} + D_{kj}$ for all $0 \leq i, j, k \leq n$. We will call D *canonical* iff it is either closed, or $D_{0,0} = (0, <)$. The canonical form of a DBM is unique up to the ordering of clocks. The zone operations above can be efficiently implemented over canonical DBMs [13].

The intersection of DBMs A and B, defined over the same set of clocks with the same ordering, is $A \sqcap B = [\min(A_{ij}, B_{ij})]_{ij}$. Here, $[\![A \sqcap B]\!] = [\![A]\!] \cap [\![B]\!]$, regardless of whether A and B are closed, but $A \sqcap B$ might not be closed even if A and B are. We denote by $A \sqsubseteq B$ iff $[\![A]\!] \subseteq [\![B]\!]$. Moreover, $\top = ((0, \leq))$ and $\bot = ((0, <))$, that is, $\top$ and $\bot$ are the smallest canonical DBMs representing $Eval$ and $\emptyset$, respectively.

III. Interpolation for Timed Automata

A. Binary Interpolants from DBMs

Let A and B two DBMs such that $A \sqcap B$ is inconsistent. An interpolant for the pair (A, B) is a DBM I such that

- $A \sqsubseteq I$,
- $I \sqcap B$ is inconsistent and
- clocks constrained in I are constrained in both A and B.

This definition of a DBM interpolant is analogous to the definition of an interpolant in the usual sense [9]. As DBMs encode formulas in $\mathcal{DL}(\mathbb{Q})$, a theory that admits interpolation [12], an interpolant always exists for a pair of inconsistent DBMs. Our approach for interpolant generation is the direct adaptation of the graph-based algorithm of [12] for DBMs. For the ease of exposition, we assume that A and B are defined over the same set of clocks with the same ordering. This is without the loss of generality, as a DBM can be easily extended with extra columns and rows representing unconstrained clocks without changing its canonicity or the zone it encodes.

The algorithm searches for a negative cycle in $A \sqcap B$ to establish its inconsistency by running a variant of the Floyd-Warshall algorithm [14] on it. In this context, a cycle is negative iff the sum of its constituting bounds is less than $(\leq, 0)$. If no such cycle is found, then A is consistent with B, thus no interpolant exists, and as a side effect the canonical representation of their intersection is obtained. Otherwise, the cycle is reconstructed as a list of indexes and from it an interpolant is extracted in the following way.

Without loss of generality, assume that the cycle is $C = (0, 1, \ldots, l-1)$, and addition for indexes is interpreted modulo l. If for all indexes $0 \leq i < l$ it holds that $A_{i,i+1} \geq B_{i,i+1}$, then B is inconsistent, and the interpolant is $\top$. Dually, if for all indexes $0 \leq i < l$ we have

$A_{i,i+1} \leq B_{i,i+1}$, then A is inconsistent and the interpolant is $\bot$. Otherwise maximal A-paths can be established, each being a maximal list $(i, i+1, \ldots, j)$ of indexes in C such that $A_{i-1,i} \geq B_{i-1,i}$, $A_{i,i+1} < B_{i,i+1}$, $A_{k,k+1} \leq B_{k,k+1}$ for all $i \leq k < j$ and $A_{j,j+1} > B_{j,j+1}$. Notice that clocks x_i and x_j are constrained in both A and in B. The interpolant I is then defined as

$$I_{ij} = \begin{cases} \sum_{k=i}^{j-1} A_{k,k+1} & \text{if } (i, \ldots, j) \text{ is a maximal } A\text{-path} \\ (0, \leq) & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$$

The algorithm thus yields a canonical DBM in time $\mathcal{O}(n^3)$, even if A and B are not canonical. However, the interpolant obtained for canonical DBMs might be simpler.

B. Sequence interpolants for Timed Automata

In SAT/SMT based model checking, interpolation is used to obtain inductive sequences of states in order to refine abstractions [10]. To do this, the transitions constituting of a given path are transformed to a formula that is satisfiable iff the path is feasible. If the formula is unsatisfiable, a sequence interpolant can be extracted from its refutation produced by the solver.

This same approach can be applied to timed automata, as for a sequence of transitions, such a formula can be constructed in $\mathcal{LA}(\mathbb{Q})$ [15], a theory that admits interpolation. However, due to delay transitions, the formula will not be in $\mathcal{DL}(\mathbb{Q})$, thus the interpolant is not guaranteed to be a sequence of zones.

So in order to make interpolation applicable for the zone-based model checking of timed automata, we generalize binary interpolation for DBMs to sequences of transitions of a timed automaton, analogously to the usual definition of sequence interpolation.

Let $(e_1, e_2, \ldots e_n)$ be a sequence of edges of a timed automaton that induce an infeasible path. Then there exists a sequence of zones, represented by DBMs $(I_0, I_1, \ldots, I_n)$ such that

- $I_0 = \top$ and $I_n = \bot$,
- $\mathbf{post}(I_i, e_{i+1}) \sqsubseteq I_{i+1}$ for all $0 \leq i < n$ and
- for all $0 < i < n$, the clocks constrained in I_i are constrained both in $I_0, \ldots, I_i$ and $I_{i+1}, \ldots, I_n$.

Such an interpolant can be calculated by using the image operators $\mathbf{post}$ and $\mathbf{pre}$. Let $B_n = \top$ and $B_{i-i} = \mathbf{pre}(B_i, e_i)$ for all $0 \leq i < n$. Moreover, let $A_0 = \top$, and $A_{i+1} = \mathbf{post}(I_i, e_{1+1})$ for all $0 \leq i < n$. Then we can define I_i for all $0 \leq i \leq n$ as the interpolant for the pair (A_i, B_i). This calculation is analogous to the usual computation of a sequence interpolant in terms of binary interpolation, and it can be easily shown that $(I_0, I_1, \ldots, I_n)$ is indeed a sequence interpolant.

C. Lazy Abstraction for Timed Automata using Interpolants

For our purposes, an *abstract reachability tree* (ART) is a rooted directed tree of nodes labeled by pairs of the form (ℓ, E), where E is an abstract state representing a zone $[\![E]\!]$. A node where $[\![E]\!] = \emptyset$ is called infeasible. A node might be marked covered by an other node, called the covering node. A node is excluded iff it is either covered, infeasible or has an excluded parent. An ART is ℓ-safe iff all nodes labeled by location ℓ are excluded. An ART is said to be well-labeled iff it satisfies the following properties:

1) the root of the tree is labeled (ℓ_0, E_0) for E_0 such that $[\![E_0]\!] = \{\eta_0\}$,
2) for each non-excluded node labeled (ℓ, E) and transition $e = (\ell, g, R, \ell')$, there is a successor node labeled (ℓ', E') such that $\mathbf{post}([\![E]\!], e) \subseteq [\![E]\!]$, and
3) each covered node labeled (ℓ, E) is covered by a non-excluded node (ℓ, E') such that $[\![E]\!] \subseteq [\![E']\!]$.

A well-labeled, ℓ-safe ART for a timed automaton is a proof that ℓ is unreachable. The goal of model checking is to find such an ART, or show a counterexample.

For this end, one extreme would be to expand the tree based on a precise post-image operator $\mathbf{post}$ and never apply refinement. An other approach, known as IMPACT [10] in software model checking, would expand the tree with abstraction $\top$ for each new node, and apply interpolation-based refinement to maintain well-labeledness and safety in the following cases:

- A non-excluded node labeled with ℓ_{err} is encountered. To exclude the error node, an interpolant is calculated for the path from root to the node. If such an interpolant exists, it is used to strengthen the current abstraction, otherwise a counterexample is given.
- To enforce coverage. To enforce $[\![E]\!] \subseteq [\![E']\!]$ and enable coverage between non-excluded nodes (ℓ, E) and (ℓ, E'), an interpolant is calculated for the path from the least common ancestor of the two nodes and the node to cover. If such an interpolant exists, it is used to strengthen the abstraction, and thus enable coverage.

Our approach can be seen as the combination of the two algorithms above. It is the adaptation of the strategy proposed in [7] for interpolation-based refinement. In this framework, an abstract state is essentially a pair of zones (Z, W). Here, Z tracks the precise reachability information, and W an abstraction of it, thus we define $[\![(Z, W)]\!]$ as $[\![W]\!]$. The root of the tree is labeled (Z_0, Z_0). When expanding a node labeled (ℓ, Z, W) for a transition $e = (\ell, g, R, \ell')$, the successor obtains label (ℓ', Z', W') such that $Z' = \mathbf{post}(Z, e)$ and $W' = \top$. Once a node labeled (ℓ_{err}, Z, W) is encountered where $Z \neq \bot$, we know that ℓ_{err} is reachable, and the counterexample can be reported.

We apply interpolation in the following two cases to maintain well-labeledness:

- A non-excluded node labeled $(\ell_{err}, \bot, W)$ is encountered. In this case, the only reasonable thing to do is to exclude the node. Suppose the node can not be excluded by covering any of its ancestors. We then strengthen the abstraction by a zone interpolant computed for the path from the root to the node, thus excluding the node.

- To enforce coverage. Given a pair of non-excluded nodes labeled (ℓ, Z, W) and (ℓ, Z', W'), respectively, such that $Z \subseteq W'$ but $W \not\subseteq W'$, we can strengthen the abstraction to obtain $W \subseteq W'$. This however requires the complementation of zone W', which may yield more than one zones. We use each of those zones as a target for computing an interpolant from the root to the node. By strengthening the abstraction with each of those interpolants, we effectively enforce coverage.

Note that by strengthening the abstraction for a given path, some previously covered nodes might become uncovered. For these nodes, we can try to enforce coverage, which might be a costly operation. However, forced coverage can be applied incrementally: when the abstraction W of a covering node is strengthened by an interpolant I, it is sufficient to use I as a target for interpolation, which is possibly simpler than the new abstraction. Moreover, a heuristic can be applied that restricts forced coverage for cases where the target is represented by a small DBM.

IV. Implementation

We implemented the DBM-based interpolation approach described in Section III-A. Moreover, as a proof of concept, we implemented the algorithm proposed in Section III-C as a prototype in the formal analysis framework THETA. Our current implementation does not support forced coverage.

Nevertheless, the algorithm shows promising results in terms of both speed and the size of the ART it constructs. We compared our algorithm with basic forward search (without extrapolation), and two versions of IMPACT without forced covering implemented for timed automata, one for predicate abstraction (P), and one for zone abstraction (Z). We used a small model, the model of Fischer's protocol for four processes. Besides execution time of the analysis and the number of nodes in the resulting ART, we calculated the number of nodes that are either non-excluded or leaf, as this is the number of meaningful nodes.

	e. time (s)	#n	#n minimized
our algorithm	7	4946	985
basic search	10	9857	9857
IMPACT(Z)	TO	-	-
IMPACT(P)	TO	-	-

From the results in the table above, it can be seen that IMPACT without forced covering reaches timeout (set to one minute) regardless of the abstract domain used, as the algorithm spends a significant amount of time searching in unreachable segments of the state space. On the other hand, our algorithm generates an ART half as big as the basic algorithm, even without forced covering. As the number of nodes in the reduced ART suggests, with a better search and covering strategy, the number of nodes in the ART can be significantly decreased.

V. Conclusions and Future Work

In this paper, we proposed a lazy abstraction algorithm for reachability checking of timed automata based on interpolation for zones. In order to efficiently handle zone abstraction, we treat interpolant generation as an operation over difference bound matrices. Furthermore, we extended the notion of zone interpolation to sequences of transitions of a timed automaton. By using interpolants for abstraction, the proposed algorithm has the potential to significantly speed up the convergence of reachability checking.

We are currently extending our implementation with forced covering. In the future, we plan to investigate how interpolation can be efficiently used in conjunction with known abstractions for zones to obtain coarse abstractions for timed automata. Moreover, we intend to search for more efficient interpolant generation algorithms for DBMs. Furthermore, we are going to thoroughly evaluate an optimized version of our algorithm on usual benchmarks for timed automata, and compare it to the lazy abstraction algorithm based on LU-bounds [7].

References

[1] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[2] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Petterson, Y. Wang, and M. Hendriks, "UPPAAL 4.0," in *Third International Conference on the Quantitative Evaluation of Systems*. IEEE, 2006, pp. 125–126.

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

[4] G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek, *Lower and Upper Bounds in Zone Based Abstractions of Timed Automata*. Springer Berlin Heidelberg, 2004, vol. 2988 LNCS, pp. 312–326.

[5] F. Herbreteau, B. Srivathsan, and I. Walukiewicz, "Better abstractions for timed automata," in *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, ser. LICS. IEEE, 2012, pp. 375–384.

[6] F. Herbreteau, D. Kini, B. Srivathsan, and I. Walukiewicz, "Using non-convex approximations for efficient analysis of timed automata," in *FSTTCS 2011*, ser. LIPIcs, vol. 13, 2011, pp. 78–89.

[7] F. Herbreteau, B. Srivathsan, and I. Walukiewicz, "Lazy abstractions for timed automata," in *Computer Aided Verification*, vol. 8044 LNCS. Springer International Publishing, 2013, pp. 990–1005.

[8] W. Craig, "Three uses of the herbrand-gentzen theorem in relating model theory and proof theory," *The Journal of Symbolic Logic*, vol. 22, no. 3, pp. 269–285, 1957.

[9] K. L. McMillan, "Interpolation and sat-based model checking," in *Computer Aided Verification*, vol. 2725 LNCS. Springer Berlin Heidelberg, 2003, pp. 1–13.

[10] ——, "Lazy abstraction with interpolants," in *Computer Aided Verification*, vol. 4144 LNCS. Springer Berlin Heidelberg, 2006, pp. 123–136.

[11] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *Journal of the ACM*, vol. 50, no. 5, pp. 752–794, 2003.

[12] A. Cimatti, A. Griggio, and R. Sebastiani, "Efficient interpolant generation in satisfiability modulo theories," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 4963 LNCS. Springer Berlin Heidelberg, 2008, pp. 397–412.

[13] J. Bengtsson and W. Yi, *Timed Automata: Semantics, Algorithms and Tools*. Springer Berlin Heidelberg, 2004, vol. 3098 LNCS, pp. 87–124.

[14] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, no. 6, pp. 345–, 1962.

[15] R. Kindermann, T. Junttila, and I. Niemelä, "Beyond Lassos: Complete SMT-Based Bounded Model Checking for Timed Automata," in *Formal Techniques for Distributed Systems*, vol. 7273 LNCS. Springer Berlin Heidelberg, 2012, pp. 84–100.