A New Data Structure Based on Intervals for Parametric Counter Automata

Petr Matoušek[*]

Abstract

Traditional approaches to the verification of real-time systems deal with timed models where time is expressed by variables that are compared with explicite values (i.e., integers). Parametric timed and counter models use parameters to define constraints over clocks or counters. Verification of automata with parameters is generally undecidable. However, there are restricted classes of parametric systems that can be successfully verified. Analysis mostly depends on the efficient data structure that is used to express behavior of the system. In this paper we discuss data structures used for representation of timed and counter automata. We introduce a new data structure based on parametrized intervals for counter automata and operation that are needed for verification. This structure makes operations over parametric counter automata simple in comparison to other approaches.

Introduction

Parametric analysis works with systems that contains special variables that are not changed during the execution - parameters. In parametric models clocks and counters can be compared with parameters. Parameters are used in transitions where they define lower and upper bounds on clocks or counters. Parameters may range over infinite domains and are related by a set of constraints. Using parametric reasoning we can either verify that the system satisfies some property for all possible values of the parameters, or we can find constraints on the parameters that define the set of all possible values for which the system satisfies a property.

In this paper, at first we give an overview of parametric counter automata defined by [AHV93], [AAB00], and the basics of verification of parametric systems. The second section shows structures used for data representation of parametric timed and counter automata - parametric DBM's [AAB00] that are implemented in TReX and parametric hypercubes (pHCubes) - a new data structure used for counters. This structure is based on intervals [ST02]. We introduce its structure and operations over it. The advantage of this structure is that it reduces space needed to represent data and simplifies some operations

[*]Faculty of Information Technology, Brno, Czech republic, matousp@fit.vutbr.cz

(emptiness test, intersection, etc.). Second contribution is that this structure allow to represent conditions of the form $x_1 + \ldots + x_n \prec t$ while DBMs allow comparison of terms with only two variables $x_i - x_j \prec t$.

1 Parametric Real-time Reasoning

As mentioned in [AHV93], the main question for parametric automata (we consider both timed and counter automata) is the emptiness: given a parametric timed system, are there concrete values for the parameters so that the automaton has an accepting run? This question in generally undecidable but there exist algorithms for checking the emptiness of restricted classes of parametric timed automata.

Another important question connected to the emptiness testing is termination of the analysis. Values of clocks and counters can grow beyond every limitation. The important issue is to limit their domains to terminate the analysis. There are various techniques that enforce the convergation of the analysis like widening technique or convex hull [ACH$^+$95] implemented in HyTech, or extrapolation based on control loops introduced in [AAB00] and implemented in TReX.

1.1 Parametric Counter Automata

Parametric counter automata are similar to timed automata of [Alu99] augmented with parameters. Timed automata are finite-state machines with clocks which are used to constrain the accepting runs by imposing timing requirements on the transitions. While ordinary automata generate sequence of events (states), time automata are constrained by timing requirements and generate timed sequences. All clocks proceed at the same rate and measure the amount of time that has elapsed since they were started or reset. Each transition of the automaton may reset some of the clocks, and it puts certain constraints on the values of the clocks: a transition can be taken only if the current clock values satisfy the corresponding constraints. Parametric counter automata are similar to parametric timed automata. Only difference is that function *post*() is defined without considering time-transitions.

Parametric Counter Automaton is a tuple $\mathcal{C} = \langle L, C, P, I, \delta \rangle$, where

- L is a finite set of locations,
- C is a finite set of integer valued variables (counters),
- P is a finite set of parameters,
- $I : L \to SC(C, P)$ is a mapping that associates invariants with locations, $SC(C, P)$ is a simple parametric constraint expressed as a conjunction of formulas of the form $x \prec t$ or $x - y \prec t$ where $x, y \in X, \prec \in \{<, \leq\}$, t is a arithmetical linear term over the set of parameters $AT(\mathcal{P})$ defined by the grammar $t ::= c \mid p \mid t - t \mid t + t \mid c * t$ where $p \in P$ is a parameter and $c \in \mathbb{Z}$ is a constant.

- δ is a set of transitions of the form (l_1, g, sop, l_2) where $l_1, l_2 \in L, g \in SC(C, P)$ is a guard, and *sop* is a simple operation over C - a special kind of assignment of the form $x := y + t$ or $x := t$ where $x, y \in C$ and $t \in AT(\mathcal{P})$.

A configuration of $\mathcal{C}$ is a triplet $\langle l, \nu, \gamma \rangle$ where $l \in L$ is a location, $\nu \in C \to \mathcal{D}$ is a valuation of the counters, and $\gamma : P \to \mathcal{D}$ is a valuation of parameters. Set $\mathcal{D}$ denotes a domain of counter values. $\mathcal{D}$ can be $\mathbb{R}^{\geq 0}$ or $\mathbb{N}$. Given a transition $\tau \in \delta$ we define an action relation $\to_\tau$: For a state $\langle l_1, \nu_1, \gamma_1 \rangle$ and a transition $\tau = (l_1, g, sop, l_2) \in \delta$ we define a transition relation $\to_\tau$ between configurations as $\langle l_1, \nu_1, \gamma_1 \rangle \to_\tau \langle l_2, \nu_2, \gamma_2 \rangle$ such that (ν_1, γ_1) satisfies g and $\nu_2 = sop(\nu_1) \wedge \gamma_1 = \gamma_2$. The function $post_\tau$ here is defined without considering time-transitions.

1.2 Verification

In automata-theoretic verification, a finite-state system is modeled by an automaton. The set of words accepted by the automaton corresponds to the possible behaviours (runs) of the system. While automata on infinite words can be used to deal with nonterminating processes, for verifying safety properties it suffices to consider automata over finite words.

- For verification of parametric systems we want to prove that a system satisfies its specification for all parameters values that meet a given set of constraints. Given a set $\Delta \subseteq [\mathcal{P} \mapsto \mathcal{D}]$ of possible parameter valuations, we wish to verify that no $\gamma \in \Delta$ is consistent with A (automaton of desirable behavior), that is $\Delta \cup \Gamma(A) = \emptyset$.

- In parameter synthesis, we want to find all parameter valuations $\Gamma(A)$ that are consistent with A, or we want to find a parameter valuation that is consistent with A and is optimal with respect to some criterion.

The question of deciding whether a specific parameter valuation γ is consistent with A can be solved using techniques developed in [AD90]. Given a parameter valuation γ, one can construct a finite-state automaton A_γ that accepts $L_\gamma(A)$. Then $\gamma \in \Gamma(A)$ iff A_γ accepts some string. The solution is known to be PSPACE-complete.

2 Data structures for parametric counter systems

2.1 Parametric DBM's

We define parametric DBM's according to [AAB00]. Let $\mathcal{M}$ be a parametric difference bound matrix that encodes the contraints in form $x_i - x_j \prec t$ (similar to DBMs) where $x_i, x_j \in C$ are counters, $t \in AT(\mathcal{P})$, and $\prec \in \{<, \leq\}$. A constrained PDBM is a pair $(\mathcal{M}, \varphi)$ where $\mathcal{M}$ is a PDBM and φ is a parameter constraint - quantifier-free formula over parameters given by grammar $\varphi ::= t \leq t \mid \neg\varphi \mid \varphi \vee \varphi$. Basic operations on constrained PDBM are:

- *Transformation into a canonical form.* Canonical forms of DBM's (nonparametric case) are constructed using Floyd Warshall algorithm which computes the minimum path between all pairs of entries. In parametric case we follow the same principle by running a symbolic Floyd Warshall algorithm. During computation the algorithm needs to determine minimums between terms. For that, algorithm assumes each of the two possible cases and check their consistency with respect to the parameter constraints: given two terms t_1 and t_2 it considers the case where $min(t_1, t_2) = t_1$, resp. t_2, and adds $t_1 < t_2$, resp. $t_1 \geq t_2$ in the parameter constraints. In order to check the consistency of each of the possible cases when computing the minimum between two terms, we have to test the satisfiability of formulas φ of the form $\Phi(P) \wedge t_1 \prec t_2$ where $\prec \in \{<, \leq\}$ and Φ is a parameter constraint. The transformation into a canonical form is used for emptiness check.

- *Intersection.* Let $S_1 = (M_1, \Phi_1)$ and $S_2 = (M_2, \Phi_2)$. Intersection consists of computing minimum for every i, j between two terms $M_1(i, j)$ and $M_2(i, j)$ under the parameter constraints $\Phi_1 \wedge \Phi_2$.

- *Inclusion test.* The inclusion of S_1 in S_2 can be expressed by formula $\forall P.\Phi_1(P) \wedge \Phi_2(P) \Rightarrow M_1 \leq M_2$.

PDBMs use two-dimensional arrays for representing clocks and counter. However, for counters we don't need two dimensions because it is not needed to store differences between every two clocks/counters. So we propose a new structure that reduces both the space and time requirements for storing and manipulating data.

2.2 Parametric hypercubes

Parametric hypercubes (pHCube) symbolically represent data domain of variables $x_1, \ldots, x_n$. We use this structure to represent counters in extended time automata. Manipulation with this structure (intersection, union, widening) is easier in comparison with other parametric data structures like PDBMs or pohyhedra.

This structure contains a parametrized interval for every variable constrained by a formula φ. First, we define contrained parametrized interval over variable x as follows:

$$\tilde{I}(x, \mathcal{P}) = (\langle a, b \rangle, \varphi) = (\langle (\prec_i, t_i), (\prec_s, t_s) \rangle, \varphi)$$

where $\prec_i, \prec_s \in \{<, \leq\}.a = (\prec_i, t_i), b = (\prec_s, t_s)$ are constrained parametrized bounds such that $(-x \prec_i t_i) \wedge (x \prec_s t_s) \wedge \varphi$. Terms t_i, t_s are from $AT(\mathcal{P}) \cup \{\infty, -\infty\}$ and constraint φ is a quantifier-free formula over $\mathcal{P}$ given by grammar $\varphi ::= t \leq t \mid \neg\varphi \mid \varphi \vee \varphi$.

Parametric hypercube is an abstract data structure over intervals. It can be described using parameterized interval formula, i.e., conjuctions of constraints over parametrized terms of the form:

$$\bigwedge_j (t_i^j \prec_i x_j \prec_s t_s^j)$$

where $x_j \in X$ is a variable, and t_i^j, resp. t_s^j, is a lower bound (infimum), resp. an upper bound (supremum).

Formally, *parameterized hypercube ph* over set of variables $X = \{x_1, \ldots, x_n\}$ is a vector of parametrized bounds constrained by formula φ:

$$ph(X) = (I, \varphi) = (I_1, \ldots, I_n, \varphi) = (\langle a_1, b_1\rangle, \ldots, \langle a_n, b_n\rangle, \varphi)$$

where a_i, resp. b_i are the lower, resp. the upper parameterized bound of variable x_i, φ is a constraint over parameters $\mathcal{P}$. The form of a parameterized bound is $(\{<, \leq\}, t_i)$, where $t_i \in AT(\mathcal{P}) \cup \{\infty, -\infty\}$.

For example, let $\vec{v} = (x, y, z)$ is a set of variables over X, $\mathcal{P} = \{p, q\}$, and $\varphi \in F(\mathcal{P})$. Let $ph(\vec{v})$ be a pHCube over vector $\vec{v}$ such that

$$ph(\vec{v}) = (\langle(<, 0), (\leq, 2 * p)\rangle, \langle(<, \infty), (<, \infty)\rangle, \langle(\leq, -3), (<, 3 + q)\rangle, -p \leq 1 \ \wedge -q \leq -1)$$

This structure represents data domain of three variable x, y, z. Two variables x and z are bounded by intervals $(0 < x \leq 2{*}p)$, $(3 \leq z < 3{+}q)$ respectively. Variable y is unbounded $(-\infty < y < \infty)$ and covers entire domain. Parameters $\{p, q\}$ are constrained by formula $\varphi = p \geq -1 \ \wedge \ q \geq 1$.

2.3 Operations on pHCubes

In this section we will define operations on pHCubes and their relation to the verification of counter automata. Because of lack of space we show only few operations over parameterized hypercubes.

- *Emptiness test.* We test emptiness before applying any other operation during system analysis, for example, inclusion, equality etc. Let *ph* be a pHCube. We say that *ph is not empty*, if $\exists p \in \mathcal{P} \ . \ \varphi \bigwedge_i (a_i \leq b_i)$. *ph is empty* if negation of this formula is satisfied.

- *Universality test.* pHCube is universal if every its bound is infinite and does not depend on a constraint expressed by formula φ. Implementation of this simple. pHCube *ph* **is universal** if and only if $\forall i \in \{1, \ldots, n\} \ . \ a_i = \{<, \infty\} \ \wedge \ b_i = \{<, \infty\}$. For instance, unbounded pHCubes is universal.

- *Inclusion.* Test of inclusion of pHCubes is very important operation over pHCubes. After computation of a new configuration we test if a new pHCube is included into existing ones.

  Inclusion of pHCubes is based on the relation of the total order $\subseteq$. Here, we extend this relation on pHCubes. Let $ph = (\langle a_1, b_1\rangle, \ldots, \langle a_n, b_n\rangle, \varphi)$ and $ph' = (\langle a_1', b_1'\rangle, \ldots, \langle a_n', b_n'\rangle, \varphi')$ be two pHCubes. We define **inclusion** on pHCubes as follows:

$$\begin{aligned} ph \subseteq ph' \quad &\iff \quad \forall p \in \mathcal{P} \ . \ \varphi \Rightarrow (\varphi' \ \wedge \ \bigwedge_i (a_i \leq a_i' \ \wedge \ b_i \leq b_i')) \\ &\iff \quad \forall p \in \mathcal{P} \ . \ \neg\varphi \ \vee \ (\varphi' \ \wedge \ \bigwedge_i (a_i \leq a_i' \ \wedge \ b_i \leq b_i') \end{aligned}$$

For implementation is useful to test non-inclusion. The new formula is

$$\begin{aligned} ph \not\subseteq ph' &\iff \exists p \in \mathcal{P} \,.\, \varphi \wedge (\neg\varphi' \vee \bigvee_i (\neg(a_i \le a_i' \wedge b_i \le b_i')) \\ &\iff \exists p \in \mathcal{P} \,.\, \neg(\varphi \Rightarrow \varphi') \vee (\varphi \wedge \bigwedge_i (\neg(a_i \le a_i' \wedge b_i \le b_i')) \end{aligned}$$

Using this formula we test inclusion by testing satisfiability of the formula. If it is not satisfied, the phcubes are included. An empty pHCube is included in every phCube. If ph is empty, $ph \subseteq ph'$. If ph' is empty, $ph \not\subseteq ph'$.

- *Equality.* The operation equality can be substituted by the operation of inclusion because $ph = ph' \iff (ph \subseteq ph') \wedge (ph' \subseteq ph)$.

- *Intersection.* Computation of intersection is an expensive operation. For every dimension we need to test all possible cases of relations between two intervals. Here, we demonstrate intersection for pHCubes with only one variable. Let $ph = (I, \varphi), ph' = (I', \varphi')$, where $I = \langle a, b\rangle, I' = \langle a', b'\rangle$. The result of the intersection will be a list of pHCubes $(I'', \varphi \wedge \varphi\prime \wedge \psi)$. We distinguish following cases:

  1. If $I' < I \vee I < I'$, then intersection will be empty. The following constraint must be satisfied: $\psi = b' < -a \vee b < -a'$.
  2. If $I' \subseteq I \vee I \subseteq I'$, then one interval is included into another. The result will be one of the intervals. For $I' \subset I$ formula $\psi = a' < a \wedge b' < b$ must be satisfied, for $I \subset I'$ formula $\psi = a < a' \wedge b < b'$ must be satisfied.
  3. If $I \cap I' \neq \emptyset$ the following four cases are possible:
     (a) $\psi = a < a' \wedge b < b'$, $I'' = \langle a, b\rangle$
     (b) $\psi = a < a' \wedge b' \le b$, $I'' = \langle a, b'\rangle$
     (c) $\psi = a' \le a \wedge b < b'$, $I'' = \langle a', b\rangle$
     (d) $\psi = a' \le a \wedge b' \le b$, $I'' = \langle a', b'\rangle$

     For symbolic computation only the third case is enought to implement. The second case can be simply implemented by inclusion test as introduced above. In addition, we don't need to detect the first case because we are interested in intersection only. If the first case happens none of the formulae from the third case will be satisfied and the result will be an empty pHCube.

  While looking at the relations above remember that interpretation of interval $I(x) = \langle a, b\rangle$ is $-a < x < b$.

- *Extrapolation.* Extrapolation computes the effect of applying a loop on φ if the first step of the loop gives φ_D. Types of extrapolation are widening, that removes the constraint with changes, and acceleration, that adds the difference between code φ and φ_D as a period. In the result, the acceleration method may transform the φ into an open domain by introducing acceleration parameters, see [AAB00], for instance.

3 Conclusion

This paper deals with the problem of verification of parametric systems. We discuss parametric counter automata and introduce a new structure to represent data - parametric hypercubes. In comparison with parametric DBMs parametric hypercubes are smaller in space and operation over them are simpler. They allow to represent constraints with more than two variables in comparison with PDMBs. This new data structure is implemented now in verification tool TREX[BCAS01]. Now we test the implementation and compare with PDBMs. The result of the comparison with PDBMs and the full description of the pHCube library will be included in the author's PhD. thesis.

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In E.A. Emerson and A.P. Sistla, editors, *Proceedings of the 12th CAV*, volume 1855 of *LNCS*, pages 419–434. Springer Verlag, July 2000.

[ACH$^+$95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[AD90] R. Alur and D.L. Dill. Automata for modeling real-time systems. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 322–335. Springer, 1990.

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, pages 592–601, 1993.

[Alu99] R. Alur. Timed automata. In *Proceedings of 11th CAV*, volume 1633 of *LNCS*, pages 8–22, 1999.

[BCAS01] A. Bouajjani, A. Collomb-Annichini, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *Proceedings of CAV*, volume 2102 of *LNCS*, pages 368–372. Springer Verlag, June 2001.

[ST02] Karsten Strehl and Lothar Thiele. Interval diagrams for efficient symbolic verification of process networks. *IEEE Transactions on Computer-Aided Design of Integrated Ciruits and Systems*, 2002.