Verification of Concurrent Systems with Parametric Delays Using Octahedra

Robert Clarisó and Jordi Cortadella*
Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract

A technique for the verification of concurrent parametric timed systems is presented. In the systems under study, each action has a bounded delay where the bounds are either constants or parameters. *Given a safety property, the analysis computes automatically a set of constraints on the parameters sufficient to guarantee the property. The main contribution is an innovative representation of the parametric timed state space based on bit-vectors. Experimental results from the domain of timed circuits show that this representation improves both CPU time and memory usage with respect to another parametric approach, convex polyhedra.*

1. Introduction

The behavior of many concurrent systems depends on their temporal characteristics. Many real-time formalisms describe these characteristics using bounded delays (e.g. timed transition systems [16]) or clocks whose value can be read or reset (e.g. timed automata [2]). In these formalisms, the delays and clock thresholds are usually defined as known constants. A more general class of models is that of *parametric* real-time systems [3], where these values become parameters of the problem. In addition to simply checking whether a temporal property is satisfied by a parametric system, it is also possible to compute *which* values of the parameters satisfy the property.

For example, let us consider the timed Petri Net in Figure 1 depicting the railroad crossing problem. The subnet on the top describes the behavior of a train as it approaches a crossing. The subnet on the bottom depicts the behavior of the gate at the crossing. Each event has a delay bounded by an interval $[d, D]$, which captures the amount of time elapsed since the event becomes enabled until it occurs. Some bounds of the intervals are parameters of the problem: the time required to lower and raise the gate ($[d_L, D_L]$ and $[d_R, D_R]$ respectively), the time required by the controller of the gate to issue a command ($[d_C, D_C]$) and the time between the sensor detects the proximity of the train until the train enters the crossing ($[d_E, D_E]$). The following safety property should be satisfied: "whenever the train is *inside* the crossing, the gate should be *closed*". The analysis described in this paper is able to discover the safety requirement ($d_E > D_L + D_R + D_C$) automatically.

*Research funded by CICYT TIN 2004-07925 and the FPU grant AP2002-3862 from the Spanish Ministry of Education, Culture and Sports.

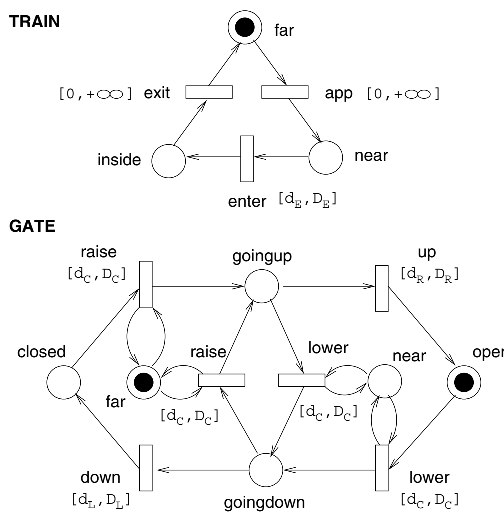


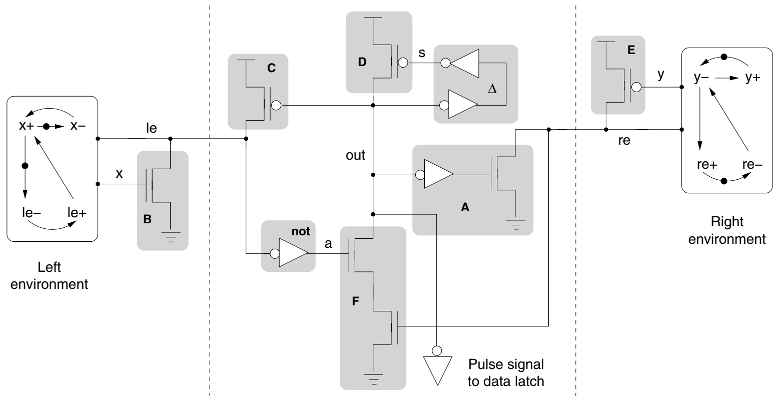
Figure 1. The railroad crossing problem

Techniques used in real-time systems such as Difference Bound Matrices (DBMs) [14] cannot be used to study parametric systems. These methods can only handle constraints that involve at most two variables, while in parametric systems several parameters may appear in the same constraint. Furthermore, many interesting problems for parametric timed systems are undecidable. As such, it is only possible to address them using approximate techniques (e.g. [1]) or semi-decision procedures (e.g. [5]).

This paper focuses on the verification of a specific class of timed systems: timed circuits [18]. They rely on timing constraints to ensure a correct operation. Timing constraints limit the degree of concurrency in the circuit: some behaviors valid in the untimed domain become forbidden in the timed domain. The results presented in this paper extend the approach presented in [10]. This method, based on *abstract interpretation* [11], discovers a very general class of timing constraints that can be used later to:

- Efficiently check if an implementation of a circuit with specific delays satisfies the timing constraints.
- *Choose* which delays should be used in the implementation in order to improve performance, while ensuring a correct functionality.

This method does not impose *a priori* any restriction on the delays of the elements of the circuit. Instead, delays are modeled as *symbols*. The output timing constraints are *linear inequalities* describing a set of sufficient constraints on these symbols that guarantee a correct behavior. Notice that



$$\begin{array}{rcl}
\delta(x-) & > & \delta(B)\\
\delta(B)+\delta(not)+\delta(F) & > & \delta(x-)\\
\delta(y+) & > & \delta(E)\\
\delta(y-)+\delta(A) & > & \delta(\Delta)+\delta(D)+\delta(C)\\
\delta(\Delta) & > & \delta(not)+\delta(C)\\
\delta(y+) & > & \delta(\Delta)\\
\delta(x+) & > & \delta(not)+\delta(C)\\
\delta(x+) & > & \delta(\Delta)+\delta(D)+\delta(C)\\
\delta(x+) & > & \delta(y+)+\delta(A)+\delta(y-)\\
\delta(D)+\delta(\Delta) & > & \delta(A)\\
\delta(y-) & > & \delta(not)+\delta(B)+\delta(C)
\end{array}$$

Figure 2. GasP FIFO controller [22]. Each shaded area has been modeled with a different symbolic delay. On the right, the timing constraints that ensure a correct operation of the circuit.

this kind of timing constraints is less restrictive than metric timing constraints[1] [6, 7] and easier to validate than relative timing constraints[2] [19, 21]. This additional freedom can be used to select more aggressive delays for larger performance gains.

1.1. Motivating example: GasP FIFO controller

The approach presented in this paper is suitable for the verification of small controllers, typically designed by hand or by sophisticated synthesis tools, whose behavior depends on the timing characteristics of the components, such as asynchronous controllers (e.g. [20, 22]).

For example, Figure 2 shows a GasP FIFO controller [22]. The environment of this controller is modeled with Signal Transition Graphs (STG) [8]. Gates, transistors and environment events have a delay specified as a symbol (δ).

The correctness of the circuit has been verified with respect to three criteria: *absence of short-circuits*; *absence of hazards*; and *conformance*, i.e. all output events produced by the circuit are expected by the environment. These criteria can be satisfied with the timing constraints that appear in Fig.2. For example, the constraint ($\delta(x-) > \delta(B)$) models the fact that changes in the input signal x must be slow enough to let the transistor B discharge the signal le. On the other side, the constraint ($\delta(B) + \delta(not) + \delta(F) > \delta(x-)$) establishes that the event $x-$ must be faster than the path defined by the transistor B, the inverter and the pair of transistors in F. Otherwise, there is a short-circuit as transistors B and C may be both on.

1.2. The contribution

The main contribution of this paper is the innovative representation for the linear timing constraints presented in Section 3. Instead of using linear constraints, only *unit* constraints, i.e. linear constraints with coefficients $\{-1, 0, +1\}$, are considered. This restriction is useful because most timing constraints are implicitly comparing the delay of two paths in the circuit:

$$\underbrace{(\delta_1 + \cdots + \delta_i)}_{\text{delay(path}_1)} - \underbrace{(\delta_{i+1} + \cdots + \delta_n)}_{\text{delay(path}_2)} \geq k$$

The encoding of unit constraints is based on bit-vectors and it improves memory and time usage with respect to previous representations based on convex polyhedra [10] and decision diagrams [9], although it may generate more restrictive timing constraints. Using this new method, larger circuits which were not analyzable previously can be successfully studied. Furthermore, the analysis of other types of parametric timed systems, such as parametric timed automata, can also benefit from this enconding.

2. Timing analysis algorithm

2.1. Overview

The timing analysis algorithm is presented using the example in Figure 3. The input of the algorithm consists of three elements:

- An implementation of a circuit, described as a netlist of gates. In the case of Fig. 3(a), it is a circuit with two inputs a and b, and one output x.
- A description of the expected interaction with the environment. Fig. 3(b) shows a Signal Transition Graph describing how the environment changes the inputs ab and how it expects the circuit will modify the output x.
- A correctness criterion. Typically, it is defined as *conformance* to the specification and *absence of hazards*. However, any safety property can be used as the correctness criterion.

[1]Setting the lower and upper bound delays of each element, or introducing constant delay paddings to ensure correctness.

[2]Restrictions on the relative order of concurrent events.

From the first two elements, it is possible to compute the untimed state space of the circuit, as shown in Fig. 3(c). In this untimed state space, *failure* transitions that do not satisfy the correctness criterion can be identified. For example, the transition `x+` from the state $\overline{\mathtt{a}}\mathtt{bt}\overline{\mathtt{x}}$ does not satisfy the criterion, as the rising of x is not expected after the rising of b. Therefore, this transition is a failure that should be avoided by the timing constraints computed by the algorithm.

Timing analysis uses the following delay model. Wires are considered to have zero delay. Gates and events from the environment are given a bounded delay $[d, D]$, where d and D are symbols s.t $(0 \leq d \leq D)$. If an event e is given a fixed delay, i.e. $(d_e = D_e)$, the notation $\delta(e)$ will be used instead (as in Fig. 2). Other gates/events might fire in between, as long as the upper delay bound is not exceeded. If the absence of hazards is a part of the correctness criterion, any gate/event that becomes enabled must be fired before becoming disabled (otherwise it is considered a hazard).

In order to characterize the timed behavior of the circuit, a *clock* is defined for each gate and each environment event. In our example from Fig 3, there would be a clock for the OR gate $(clock_{OR})$, another for the AND gate $(clock_{AND})$ and one clock for each event from the environment (`a` and `b`). These clocks keep track of the amount of time that a gate/event has been enabled. Its value is reset to zero when it becomes enabled. When time elapses while a gate/event is enabled, its clock must be incremented. The values of the clocks can be represented using different formalisms, such as *convex polyhedra* [12,15], a system of linear inequalities.

Fig. 3(d) shows a part of the timing analysis algorithm. In state $\overline{\mathtt{abt}}\overline{\mathtt{x}}$, there are only two enabled events: the environment event `b+` and the OR gate (`t-`). The clocks for these events are set to zero, as the events have become enabled in this state. After a period of time, one of the two events should occur. If `b+` occurs before the OR gate fires, the state becomes $\overline{\mathtt{a}}\mathtt{bt}\overline{\mathtt{x}}$. In this state, the following holds:

$$d_{b+} \leq clock_{OR} \leq D_{b+}$$

as the amount of time spent firing `b+` is $[d_{b+}, D_{b+}]$. Also, the upper bound of the OR gate has not been reached, as the OR gate is still enabled. Therefore, $(clock_{OR} \leq D_{OR})$ also holds. In $\overline{\mathtt{a}}\mathtt{bt}\overline{\mathtt{x}}$ the AND gate becomes enabled. Without timing constraints, this gate can fire before the OR gate, leading to the previously mentioned failure. This failure can only happen if the following holds:

$$d_{b+} + d_{AND} \leq clock_{OR} \leq D_{b+} + D_{AND}$$

as the OR gate has remained enabled during the firing of both `b+` and the AND gate. Again, $(clock_{OR} \leq D_{OR})$ should also be satisfied.

The goal of the algorithm is the discovery of timing constraints among the symbolic delays that can avoid the failure transitions. These constraints are the complement of the inequalities required to reach the errors. In Fig. 3(d) there are only two constraints on the symbolic delays required to reach the error (abstracting the clock variable). These constraints are:

$$\begin{array}{lll} d_{b+} + d_{AND} & \leq & D_{b+} + D_{AND} \qquad \wedge \\ d_{b+} + d_{AND} & \leq & D_{OR} \end{array}$$

The first constraint is always true as $(0 \leq d_{b+} \leq D_{b+})$ and $(0 \leq d_{AND} \leq D_{AND})$ hold by definition. Therefore, the complement (false) is not a valid timing constraint. However, the complement of the second constraint, $(d_{b+} + d_{AND} > D_{OR})$, is feasible. Intuitively, it means that the circuit is correct if the OR gate is not slower than the rising time of b followed by a change in the AND gate.

The following subsections describe the fundamental parts of the algorithm: how the values of clocks are updated when an event occurs (Section 2.2); how the values of clocks from different paths are combined (Section 2.3); and how the timing constraints are chosen (Section 2.4). A detailed description of the algorithm can be found in [10].

2.2. Updating clocks values

Firing an event modifies the state of the system at two levels: untimed and timed.

At the untimed level, the new values of signals change the enabled/disabled condition of environment events and gates. Events may become enabled, become disabled, remain enabled or remain disabled. Each of these scenarios implies a different change to the clocks.

	Enabled before t	Disabled before t
Enabled after t	Increase	Reset to zero
Disabled after t	Abstract	No change

At the timed level, some time elapses between reaching a state and firing a transition towards a new state. This amount of time, called *step*, is restricted by the lower and upper delay bounds of the enabled transitions and the values of its clocks. More precisely, this step should satisfy the following properties:

- If the event being fired is x, then its lower and upper delay bounds should be fulfilled: $(d_x \leq clock_x + step \leq D_x)$.
- The upper delay bound of the other enabled events should not be exceeded:$(\forall y : y \text{ is enabled} : clock_y + step \leq D_y)$.

From these principles, the algorithm to update the clocks when an event is fired can be formulated as:

1. Define a temporary variable *step*.
2. Add the restrictions on step required by the event being fired and the events enabled/disabled in the previous/next state.
3. Reset the clocks of events that become enabled in the next state: $clock := 0$.

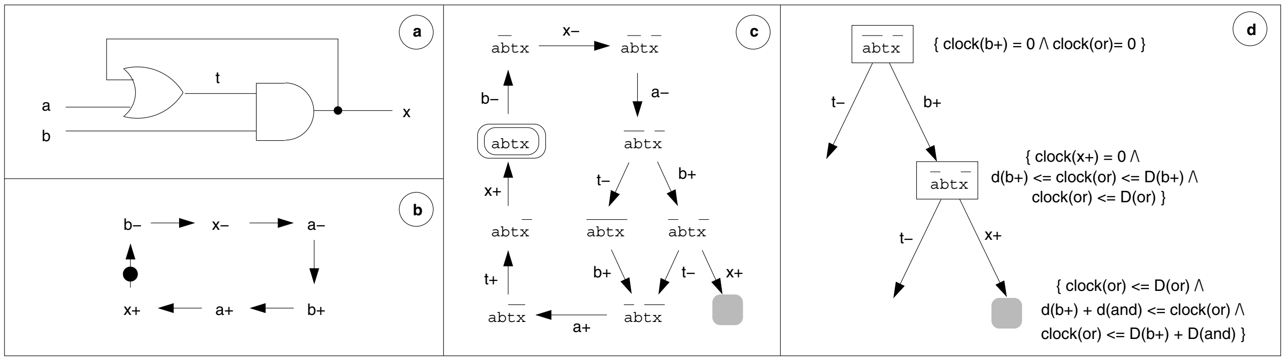

a
b
t
x
b– x– a–
x+ a+ b+
c
x– a– b– x+ t– b+ t+ b+ t– x+ a+
d
{ clock(b+) = 0 ∧ clock(or)= 0 }
t– b+
{ clock(x+) = 0 ∧
d(b+) <= clock(or) <= D(b+) ∧
clock(or) <= D(or) }
t– x+
{ clock(or) <= D(or) ∧
d(b+) + d(and) <= clock(or) ∧
clock(or) <= D(b+) + D(and) }

Figure 3. Example of the timing analysis algorithm: (a) Implementation of a circuit; (b) Signal Transition Graph describing the interaction with the environment; (c) Untimed state space, highlighting a transition that does not fulfill the specification (d) Timing constraints computed by the algorithm.

4. Increase all clocks of events that remain enabled in the next state: $clock := clock + step$.
5. Existentially abstract all clocks of events that become disabled (these clocks are no longer relevant).
6. Existentially abstract the variable *step*.

2.3. Combining clocks

The timing analysis computes the set of values that clocks can take in each untimed state of the system. Computing the exact set is not practical because the set may be very complex or even infinite. Instead, an upper approximation of this set will be computed. Upper approximations are conservative, so safety properties like the correctness criterion can still be checked.

The computation of this upper approximation follows the *abstract interpretation* paradigm [11]. In abstract interpretation, the behavior of the system is encoded as a set of fixpoint equations that can be solved iteratively. The method guarantees that (i) the solution can be found in a finite number of steps and (ii) the discovered solution is an upper approximation of the exact solution.

The system of fixpoint equations for an asynchronous circuit the following: For each state, there is one equation that defines the clock values in terms of the clock values from the incoming transitions. For example, let T_S denote the set of possible clock valuations in a state S and let $(T_S \xrightarrow{x})$ denote the possible clock valuations after firing an event x from an state S. Then, the system of equations for the circuit in Figure 3 can be defined as follows:

$$T_{\mathtt{abtx}} = InitValues \cup (T_{\mathtt{abt\overline{x}}} \xrightarrow{\mathtt{x+}})$$
$$T_{\mathtt{\overline{a}b\overline{tx}}} = (T_{\mathtt{\overline{abtx}}} \xrightarrow{\mathtt{b+}}) \cup (T_{\mathtt{\overline{a}bt\overline{x}}} \xrightarrow{\mathtt{t-}})$$
$$T_{\mathtt{a\overline{b}tx}} = (T_{\mathtt{abtx}} \xrightarrow{\mathtt{b-}}) \qquad T_{\mathtt{a\overline{b}t\overline{x}}} = (T_{\mathtt{a\overline{b}tx}} \xrightarrow{\mathtt{x-}})$$
$$T_{\mathtt{\overline{ab}t\overline{x}}} = (T_{\mathtt{a\overline{b}t\overline{x}}} \xrightarrow{\mathtt{a-}}) \qquad T_{\mathtt{\overline{abtx}}} = (T_{\mathtt{\overline{ab}t\overline{x}}} \xrightarrow{\mathtt{t-}})$$
$$T_{\mathtt{\overline{a}bt\overline{x}}} = (T_{\mathtt{\overline{ab}t\overline{x}}} \xrightarrow{\mathtt{b+}}) \qquad T_{\mathtt{ab\overline{tx}}} = (T_{\mathtt{\overline{a}b\overline{tx}}} \xrightarrow{\mathtt{a+}})$$
$$T_{\mathtt{abt\overline{x}}} = (T_{\mathtt{ab\overline{tx}}} \xrightarrow{\mathtt{t+}})$$

Intuitively, each equation defines the clock values in a state as the union of clock values after its incoming transitions. In the initial state, the enabled clocks are set to zero, while the delays can have any value that satisfies the following *invariant*: for each event x with delay $[d_x, D_x]$, $(0 \le d_x \le D_x)$. In the example from Figure 3 the values for clocks and delays in the initial state are:

$$InitValues = \{\ InitClocks\ \wedge\ Invariant\ \}$$
$$InitClocks = \{\ clock_{b-} = 0\ \}$$
$$Invariant = \{\ (0 \le d_{a+} \le D_{a+}) \wedge (0 \le d_{a-} \le D_{a-}) \wedge (0 \le d_{b+} \le D_{b+}) \wedge (0 \le d_{b-} \le D_{b-}) \wedge (0 \le d_{AND} \le D_{AND}) \wedge (0 \le d_{OR} \le D_{OR})\ \}$$

A solution to these equations can be computed using *forward increasing* [11] propagation. Initially, all states except the initial state do not have any reachable clock values. Then, all equations are applied, propagating some clock values from the initial state to its successors, using the algorithm in Figure 4. Each iteration *increases* the solution in the sense that new reachable clock values are discovered. This computation continues until a fixpoint is reached: further iterations do not discover new values. For instance the computation for state $\mathtt{a\overline{b}tx}$ would proceed as follows:

$$T^0_{\mathtt{a\overline{b}tx}} = \emptyset$$
$$T^1_{\mathtt{a\overline{b}tx}} = (T^0_{\mathtt{abtx}} \xrightarrow{\mathtt{b-}}) = \{\ (clock_{AND} = 0) \wedge Invariant\ \}$$
$$T^2_{\mathtt{a\overline{b}tx}} = (T^1_{\mathtt{abtx}} \xrightarrow{\mathtt{b-}}) = T^1_{\mathtt{a\overline{b}tx}} \qquad \rightarrow \text{Fixpoint!}$$

Figure 3(d) shows some of the timing constraints that appear during the computation of these fixpoint equations. For the sake of brevity, the constraints from the invariant do not appear.

When a cycle in the state graph is translated into the equivalent equations, cyclic dependencies among equations may appear, e.g. $T_1 = f(T_2)$ and $T_2 = g(T_1)$. These dependencies are solved with a special operator called *widening* (∇), which extrapolates the result of applying the equations of the cycle an infinite number of times. Using a widening is required to guarantee that a solution can be reached in a finite number of iterations. A typical widening operator removes the constraints that are modified after

```
$i := 0$
do {
    for $all\ states\ S$
        $T_S^{i+1} := T_S^i \cup (equation\ for\ T_S^{i+1}\ using\ T^i)$
        if $(cycle)$ $T_S^{i+1} := T_S^i \nabla T_S^{i+1}$
    $i := i + 1$
} while $(\exists S : T_S^{i+1} \neq T_S^i)$;
```

Figure 4. Abstract interpretation algorithm.

the cycle, assuming that they possibly could be modified again after another iteration of the cycle. For instance, if the valuation of clocks are:

$$T_S^i = \{ (clock_1 \leq 0) \wedge (clock_2 \leq x) \}$$
$$T_S^{i+1} = \{ (clock_1 \leq 1) \wedge (clock_2 \leq x) \}$$

the widening $T_S^i \nabla T_S^{i+1}$ will be $\{clock_2 \leq x\}$. The constraint $(clock_1 \leq 1)$ is removed assuming that the next iterations may alter it indefinitely, as in $(clock_1 \leq 2)$ $\ldots (clock_1 \leq n)$. Notice that the removal of constraints may lead to a loss in precision, but the result is always an upper approximation.

2.4. Choosing timing constraints

In each failure transition, a set of constraints required for the reachability is computed by the timing analysis. Abstracting the values of the clocks, a set of necessary constraints on the symbolic delays is obtained:

$$\text{ineq}_1 \wedge \text{ineq}_2 \wedge \ldots \wedge \text{ineq}_n$$

The timing constraints that avoid the failures are the complement of these inequalities:

$$\neg\text{ineq}_1 \vee \neg\text{ineq}_2 \vee \ldots \vee \neg\text{ineq}_n$$

These disjunctions can be used directly to check whether a set of known bounded delays satisfies the timing constraints. However, other uses of the timing constraints may require a selection of specific constraints from the disjunctions to present the output of the verification as a *conjunction* of linear inequalities. This choice must be performed for each failure transition in the system, and it should attempt to select the least restrictive timing constraints. Also, the selected set of constraints should be non-contradictory.

Several heuristics are used to select the best timing constraints among the candidates. These heuristics favor the following kinds of constraints:

- Constraints where a long sequence of delays must be slower than a shorter path, e.g. $(\delta_x + \delta_y + \delta_z > \delta_t)$.
- Constraints where environment events must be slower than internal events, e.g. $(\delta_{b+} > \delta_{NOT} + \delta_{AND})$.
- Constraints that avoid several failure transitions of the circuit. Due to the concurrency in the circuit, a single error might be the cause of different failure transitions. For instance, if a transition where "`a+` happens before `b+`" is an error, there can be several failure transitions derived from this single conceptual error.

Currently, the timing constraints are selected automatically by a backtracking procedure based on these heuristics. This procedure computes the best k sets of consistent timing constraints according to the heuristics. Computing all the possible combinations would also be possible but very inefficient. This procedure is executed *after* the timing analysis, and it does not require repeating the timing analysis phase. In contrast, some related approaches [24] select one timing constraint at a time and repeat the timing analysis phase to detect new timing constraints.

3. Parameters in timing analysis

3.1. Related work

In order to study properties in a timed system, the computation of the possible valuations of clocks is required. As considering each clock valuation individually is not feasible, sets of clock valuation are analyzed collectively. For example, *zones* [14] are sets of clock valuations that can be characterized by constraints of the form $(c_1 \leq clk_i \leq c_2)$ and $(clk_i - clk_j \leq c_i)$. In parametric timed systems, sets of clock valuations similar to zones are difficult to represent, as more than two variables may appear in the same constraint. Some formalisms like *convex polyhedra* [15], *Presburger arithmetics* [4] and *Parametric Difference Bound Matrices* [5] can be used to represent these parametric zones. A hybrid approach presented in [24] uses linear programming to compute conservative constant bounds on the values of the parameters, reducing the problem to regular timing analysis at the expense of computing metric timing constrants. Finally, other approaches are based on decision diagrams [23]. All these approaches have a very high complexity, which is very sensitive to the number of parameters.

This section presents a formalism to represent parametric timed zones called *octahedra* [9]. An octahedron is a special class of convex polyhedra, defined by linear inequalities with the following restriction: all coefficients except the constant term must be unitary, i.e. $\{-1, 0, +1\}$. The original implementation of octahedra was based on a decision diagram data structure called Octahedron Decision Diagram (OhDD). This implementation reduces the memory usage with respect to convex polyhedra; this reduction is achieved by sacrificing precision in the analysis, and also leads to large increase in CPU time. In this paper, a new set-based implementation is proposed, taking advantage of the time and memory efficiency of bit-vectors. Again, the improvements in time and memory will require a loss of precision with respect to convex polyhedra.

3.2. Unit inequalities

Definition 1 (Unit inequality) *A* unit inequality *over a set of variables X is a constraint of the form*

$$\sum_{x \in P} x - \sum_{y \in N} y \geq k$$

where P and N are sets of variables ($P \subseteq X, N \subseteq X$) and k is the constant term ($k \in \mathbb{Q}$). Any unit inequality can be characterized by the triple $\langle P, N, k\rangle$.

Example: The constraint $(x+z-y \geq 2)$ is a unit inequality than can be characterized as the triple $\langle \{x, z\}, \{y\}, 2\rangle$. Only well-formed unit inequalities where the sets P and N are disjoint will be considered. For instance, $(a+b-b-d \geq 3)$ can be rewritten into the equivalent $(a - d \geq 3)$.

In the remaining of the paper, we will only consider unit inequalities over non-negative values $(\forall x_i \in X : x_i \geq 0)$. This restriction can be imposed because the variables of X model clocks and delays which cannot have negative values. Using this restriction will allow convenient definitions and an efficient implementation of the underlying operations.

Definition 2 (Implication) *A unit inequality $A = \langle P_A, N_A, k_A\rangle$* implies *a unit inequality $B = \langle P_B, N_B, k_B\rangle$, noted $A \rightarrow B$, if B is true whenever A is true. If both inequalities are defined over non-negative variables, then A implies B if and only if $P_A \subseteq P_B$, $N_B \subseteq N_A$ and $k_A \geq k_B$.*

Example: The inequality $(x - y - z \geq 7)$ implies the inequality $(x + t - y \geq 0)$ because

$$\begin{aligned}(x - y - z \geq 7) \wedge (z \geq 0) &\quad\rightarrow\quad (x - y \geq 0)\\ (x - y \geq 0) \wedge (t \geq 0) &\quad\rightarrow\quad (x + t - y \geq 0)\end{aligned}$$

However, the inequality $(x + t - y \geq 0)$ does not imply $(y \geq 3)$, for example.

Definition 3 (Trivial and infeasible inequalities) *A unit inequality $I = \langle P_I, N_I, k_I\rangle$ over a set of non-negative variables is* trivial *(always true) if and only if $N_I = \emptyset$ and $k \leq 0$. Conversely, it is* infeasible *(always false) if and only if $P_I = \emptyset$ and $k > 0$.*

Example: The unit inequality $(-x \geq 2)$ is infeasible because $(x \geq 0)$. On the other side, a unit inequality like $(x + y \geq -1)$ will always be true as $(x \geq 0)$ and $(y \geq 0)$ imply $(x + y \geq 0)$.

Definition 4 (Unit combination) *The* unit combination *of two unit inequalities A and B (noted $A \oplus B$) is the inequality obtained by adding the left-hand sides and the right-hand sides of A and B, e.g.*

$$\begin{array}{rrcl} & \sum_{x\in P_A} x - \sum_{y\in N_A} y & \geq & k_A \\ \oplus & \sum_{x\in P_B} x - \sum_{y\in N_B} y & \geq & k_B \\ \hline \multicolumn{2}{r}{\sum_{x\in P_A} x + \sum_{x\in P_B} x - \sum_{y\in N_A} y - \sum_{y\in N_B} y} & \geq & k_A + k_B \end{array}$$

$A \oplus B$ will be a unit inequality iff $(P_A \cap P_B = \emptyset)$ and $(N_A \cap N_B = \emptyset)$. However, if A and B are defined over non-negative values, the restriction $(N_A \cap N_B = \emptyset)$ is not required (see the example below). If $A \oplus B$ is a unit inequality, then it can be characterized as the triple $\langle (P_A \setminus N_B) \cup (P_B \setminus N_A), (N_A \setminus P_B) \cup (N_B \setminus P_A), k_A + k_B\rangle$.

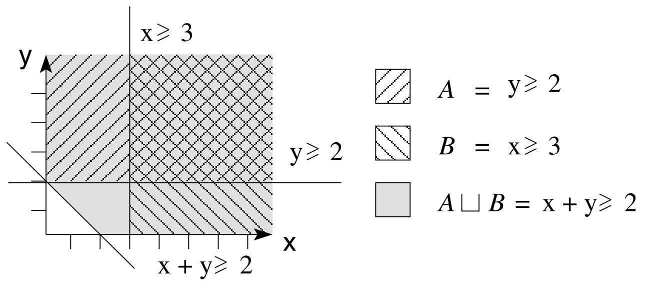


Figure 5. A graphical example of the semantics of a strongest common constraint

Example: The unit combination is a restricted version of the widely used linear combination of inequalities. For instance, the unit combination of inequalities $(x + w - t \geq 2)$ and $(t-y-z \geq 4)$ is $(x+w-y-z \geq 6)$. In some cases, the unit combination will lead to non-unit inequalities. For example, the unit combination of $(x + y \geq 2)$ and $(x - z \geq 0)$ is the inequality $(2x + y - z \geq 2)$, which is not a unit inequality. When the non-unit coefficient is negative, the non-negativity of the variables can be used to remove the non-unit coefficient. For example, the unit combination of $(x - y \geq 2)$ and $(t + w - y \geq 7)$ is the inequality $(x + t + w - 2y \geq 9)$ which is not unit. However, as $(y \geq 0)$:

$$\begin{array}{rrcl} & x + t + w - 2y & \geq & 9 \\ \oplus & y & \geq & 0 \\ \hline & x + t + w - y & \geq & 9 \end{array}$$

a unit inequality can be obtained. Notice that this strategy cannot be used when the non-unit coefficient is positive, as a constraint of the form $(-y \geq 0)$ is not available.

Definition 5 (Strongest common constraint) *The* strongest common constraint *of two unit inequalities A and B (noted as $A \sqcup B$) is another inequality C such that:*

- $(A \rightarrow C) \wedge (B \rightarrow C)$
- $\forall D : (A \rightarrow D \wedge B \rightarrow D) \Rightarrow (C \rightarrow D)$.

If the two unit inequalities $A = \langle P_A, N_A, k_A\rangle$ and $B = \langle P_B, N_B, k_B\rangle$ are defined over non-negative variables, then the strongest common constraint C can be defined as $C = \langle P_A \cup P_B, N_A \cap N_B, \min(k_A, k_B)\rangle$.

Example: Given $(x \geq 3)$ and $(y \geq 2)$, the strongest common constraint is $(x + y \geq 2)$, as:

$$\begin{aligned}(x \geq 3) \wedge (y \geq 0) &\quad\rightarrow\quad (x + y \geq 2)\\ (y \geq 2) \wedge (x \geq 0) &\quad\rightarrow\quad (x + y \geq 2)\end{aligned}$$

The values represented by these constraints can be seen graphically in Figure 5. Notice that $A \sqcup B$ does not compute a exact union of the inequalities, but an upper approximation of that union similar to a convex hull. Contrary to a convex hull, the resulting area can be described using only unit inequalities. This notion will be extended in the following section as the *octahedral hull*. Another small example is $(x+z-y-t \geq 9)$ and $(y+z-t \geq 5)$, whose strongest common constraint is $(x+y+z-t \geq 5)$.

3.3. Octahedra

An octahedron is the set of solutions to a system of unit inequalities over non-negative variables. Octahedra can be seen as a restricted case of *convex polyhedra* (system of linear inequalities) or as a generalization of *octagons* [17] (unit inequalities with at most two variables). Figure 6(a) shows an example of a system of unit inequalities and the octahedron it represents.

Several operations required in the timing analysis algorithm can be defined over octahedra like a test of inclusion, the intersection or the widening (see Fig. 6(b)-(c)). Defining the union of octahedra is more complex, as octahedra are convex objects and therefore not closed under union. The result of the union operation is an upper approximation: the smallest octahedra that includes the union. This approximation is similar to the *convex hull* (C-hull) used in convex polyhedra, and it is called *octahedral hull* (O-hull). Figure 7 shows an example of this approximation. Notice that the convex hull is always an upper approximation of the union, and the octahedral hull is always an upper approximation of the convex hull, i.e. $A \cup B \subseteq \text{C-hull}(A,B) \subseteq \text{O-hull}(A,B)$.

A more detailed presentation of octahedra and its operations, together with proofs, can be found in [9]. Also in [9], an implementation based on decision diagrams is proposed. The next section presents another representation, based on bit-vectors, with a much better time/memory trade-off.

3.4. Implementation of Octahedra

An octahedron will be implemented as a finite list of unit inequalities, where each inequality $\langle P, N, k \rangle$ is represented by two bit-vectors (encoding P and N respectively) plus the constant term. All transformations and tests that operate with inequalities will use the set-based definitions from Table **??**. These definitions allow an efficient implementation using the bit-wise operations of bit-vectors.

The operations required in the timing analysis are: union ($\cup$), intersection ($\cap$), test for inclusion ($\subseteq$), widening (∇), unit assignment of a variable and existential quantification of a variable. In octahedra, these operations can be defined as transformations of the system of unit inequalities.

Most of these operations require a *satisfiability* test: given a unit inequality $I = \langle P_I, N_I, k_I \rangle$ and an octahedron O, does O satisfy I? A possible implementation of this test is the following:

1. If I is trivial, then O satisfies I.
2. If I is infeasible, then O does not satisfy I.
3. Let N_O be the union of all the sets N from the inequalities in O. If $N_I \not\subseteq N_O$ then O does not satisfy I.
4. If the inequality I is implied by any inequality from O, then O satisfies I.
5. If the inequality I is implied by a unit combination of up to n inequalities from O, then O satisfies I.

The intuitive meaning of step 3 is that a constraint with a variable that does not appear in any inequality of O will not be satisfied by O. For instance, $(x-y-z \geq 4)$ cannot be satisfied by $(x \geq 4) \wedge (t \geq 4)$ as there are no restrictions on y or z. However, this shortcut can only be used for variables appearing with a negative coefficient (N) in the inequality. The non-negativity of variables allows us to add new variables with a positive coefficient. For example, $(x+y \geq 4)$ is satisfied by $(y \geq 8)$ even though the variable x does not appear explicitly, as the constraint $(x \geq 0)$ is implicit.

Step 5 also deserves additional comments. If all the combinations of constraints in O are considered, then the satisfiability test is exact. However, considering all possible unit combinations is too computationally expensive. Instead, a good trade-off between precision and efficiency is achieved when $n = 2$, i.e. only the combinations of pairs of inequalities of O are considered. As a consequence, the satisfiability test will be approximate, while still being *conservative*: some satisfied constraints might be reported as unsatisfied, but not the other way around. In the timing analysis algorithm, this approximation may cause *false negatives* (inability to find sufficient timing constraints, even if they exist) but it will never cause *false positives* (timing constraints will always avoid all errors).

Intersection. The intersection of two octahedra $A = B \cap C$ is defined by the system of unit inequalities with all the inequalities from A and all the inequalities from B. This is the only *exact* operation on octahedra.

Union. The union of two octahedra $A \cup B$ can be approximated as a system of unit inequalities that contains:

- The inequalities from A satisfied by B.
- The inequalities from B satisfied by A.
- The strongest common constraint of all pairs of inequalities from A and B.

Test of inclusion. An octahedra A is included in an octahedron B ($A \subseteq B$) if all the inequalities of B are satisfied by A. Notice that the approximation in the satisfiability test might lead to false negatives in the test of inclusion.

Widening. Widening is the extrapolation operator used to guarantee the termination of the analysis in the presence of loops [11]. The widening of two octahedra $A\nabla B$, where A is the initial property and B is the property after one iteration, extrapolates the result of future iterations based on A and B. The widening $A\nabla B$ can be defined as:

- If all the constraints in A and B have a constant term $k = 0$, then $A \cup B$ is a widening operator.
- Otherwise, $A\nabla B$ contains all the inequalities from A that are also satisfied by B.

$$\begin{array}{rcl} 1 \leq & x & \leq 7 \\ 0 \leq & y & \leq 7 \\ -5 \leq & x-y & \leq 6 \\ & x+y & \leq 9 \end{array}$$

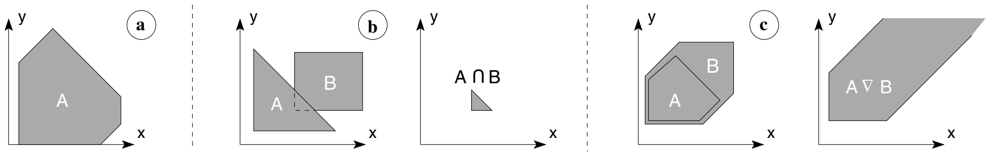
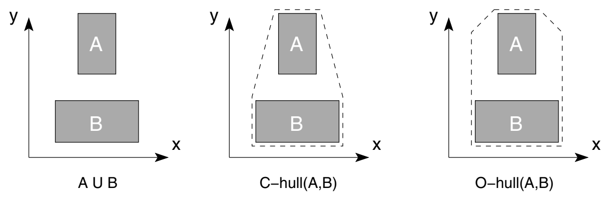


Figure 6. (a) An octahedron, (b) Intersection of octahedra and (c) Widening of octahedra.



$$\begin{array}{rcl} A & = & \{(4 \geq x \geq 2) \wedge (7 \geq y \geq 4)\} \\ B & = & \{(5 \geq x \geq 1) \wedge (3 \geq y \geq 1)\} \\ \text{C-hull} & = & \{(5 \geq x \geq 1) \wedge (7 \geq y \geq 1) \wedge \\ & & (4x - y \geq 1) \wedge (-4x - y \geq -23)\} \\ \text{O-hull} & = & \{(5 \geq x \geq 1) \wedge (7 \geq y \geq 1) \wedge \\ & & (x - y \geq 5) \wedge (-x - y \geq -11)\} \end{array}$$

Figure 7. Two upper approximations of the union: convex hull (C-hull) and octahedral hull (O-hull)

Unit assignment. Assignments of the form $x' := x + y$ are required in order to perform the timing analysis. After the assignment, we know that $(x \geq y)$ and we also know that the old value of x can be characterized as $x' - y$. Therefore, the assignment should add the constraint $(x \geq y)$ to the system of inequalities of O, and replace each instance of x in the system of inequalities by $x - y$. This replacement is implemented in the following way:

- Inequalities where $x \notin P$ and $x \notin N$ are not modified.
- The unit combinations of all pairs of inequalities $A \oplus B$ of O such that $x \in P_A$ and $x \in N_B$ are added to the system of linear inequalities. This step attempts to minimize the loss of precision: some inequalities might already contain x and y so replacing x by $x - y$ could produce a non-unit inequality. Considering these unit combinations reduces the loss of information.
- The inequalities $I = \langle P_I, N_I, k_I \rangle$ where $x \in P_I$ are transformed according to y:
  - If $y \in P_I$, then $P_I' = P_I \setminus \{y\}$.
  - If $y \in N_I$, then I is not modified.
  - Otherwise, $N_I' = N_I \cup \{y\}$.
- The inequalities $I = \langle P_I, N_I, k_I \rangle$ where $x \in N_I$ are transformed according to y:
  - If $y \in P_I$, then I is not modified.
  - If $y \in N_I$, then $N_I' = N_I \setminus \{y\}$.
  - Otherwise, $P_I' = P_I \cup \{y\}$.

After these changes, the constraint $(x \geq y)$ can be added to the system of inequalities.

Existential quantification. The quantification of a variable x will attempt to remove all the known restrictions on x while keeping as much information as possible on the rest of variables. This procedure is implemented using a process called Fourier-Motzkin elimination [13].

Inequalities where $x \notin P$ and $x \notin N$ are unaffected by this procedure. Regarding the remaining inequalities, Fourier-Motzkin proceeds by selecting one constraint where $x \in P$ and one constraint where $x \in N$. The unit combination of these constraints will not contain variable x; if the combination is a unit inequality, it is added to the system of inequalities of O. The final step is the removal of the inequalities of O that do not hold after the quantification. All inequalities where $x \in P$ must be removed, while those with $x \in N$ can be just modified so that $N' = N \setminus \{x\}$. Again, the different behavior of P and N appears from the implicit constraint $(x \geq 0)$ in O.

Figures 6(b) and (c) and Figure 7 show graphical examples of some of the operations on octahedra.

4. Experimental results

4.1. Asynchronous pipeline

In order to compare the results with the previous approaches, an example with a high degree of concurrency will be studied. The example is an asynchronous pipeline with a variable number of stages. The environment introduces data elements into the pipeline at a fixed rate $[d_{IN}, D_{IN}]$. Each stage i of the pipeline performs some computation on the data which takes some time $[d_i, D_i]$. After this computation, the stage i passes the data to the next stage $i + 1$, provided that it is empty. Otherwise, the data remains in stage i until stage $i + 1$ becomes empty. Finally, the environment reads the data from the last stage, a process that requires $[d_{OUT}, D_{OUT}]$ time units.

The correctness criterion for this example is that the environment should not be slowed down by the pipeline. More formally, this translates into the following safety property: *"whenever the environment sends new data to the pipeline, the first stage of the pipeline must be empty"*. Figure 8 shows an example of a pipeline with 4 stages, together with states that satisfy or fail to satisfy the safety property. A set of timing constraints that is sufficient to guarantee this safety property is the following:

$$d_{IN} > D_1 \wedge \ldots \wedge d_{IN} > D_N \wedge d_{IN} > D_{OUT}$$

These constraints are equivalent to:

$$d_{IN} > max(D_1, \ldots, D_N, D_{OUT})$$

which means that the pipeline works correctly if the gener-

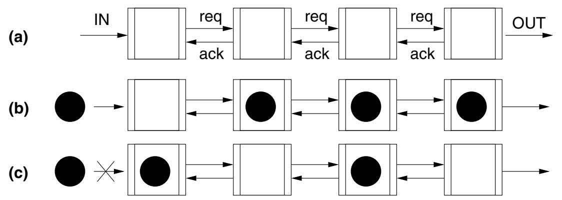


Figure 8. (a) Asynchronous pipeline with N=4 stages, (b) correct behavior and (c) incorrect behavior. Dots represent data elements.

ation of data elements from the environment is slower than the slowest stage of the pipeline.

These timing constraints can be computed using our approach for pipelines with a different number of stages. Table 2 compares the results obtained with the bit-vector representation of octahedra to those obtained the decision diagram representation (OhDD) and convex polyhedra. The bit-vector representation offers better memory and CPU time results than convex polyhedra. Remarkably, octahedra represented with bit-vectors can be used to analyze pipelines with state spaces one order of magnitude larger than those analyzable with convex polyhedra. When compared to OhDD, bit-vectors are only inferior in terms of memory for the larger examples, but the much inferior CPU times outweighs this disadvantage.

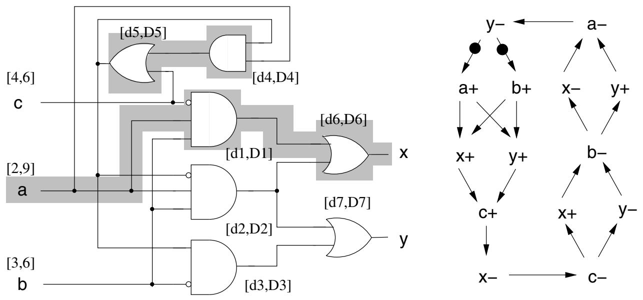
4.2. Asynchronous controllers

Several asynchronous controllers from the literature have also been verified with our timing analysis algorithm. Like the GasP example, these circuits are described as a netlist of gates, plus an environment specified with a STG. Gate and environment delays are described with a symbolic interval $[d, D]$, while wire delay is assumed to be negligible.

Figure 9 shows an asynchronous controller with the generated timing constraints for correctness. The highlighted areas in the implementation correspond to the first timing constraint. Notice that timing constraints enforce that a path in the circuit must be slower than another path.

Table 1 shows the experimental results for the verification of these controllers. For each circuit, the table describes the size of the circuit (number of signals and gates), the size of the STG that describes the interaction with the environment, the size of the untimed state space and the number of symbolic delays (Σ) in the example. Regarding the solution, polyhedra and octahedra implemented with bit-vectors are compared using two criteria: efficiency and precision.

With respect to efficiency, CPU time and peak memory usage are listed. The comparison is favorable to octahedra both in terms of memory and time. There is one example in the table with better CPU time results for convex polyhedra: the last entry, *converta*. In this specific circuit, timing constraints with non-unit coefficients are very useful, as some failures are reached when a specific path in the circuit is



$$\boxed{(D4 + D5 < d1 + d6 + 2)} \wedge (D1 < d2 + d7 + 4)$$

Figure 9. The *nowick* example.

traversed more than once. Even in this scenario, sufficient unit timing constraints can be found. Moreover, the analysis with convex polyhedra must use additional approximations for this example, as it generates too many constraints and runs out of memory (as it happens in the *desynch* example), while octahedra do not have this problem.

Quantifying the precision of the two approaches is not simple. Obviously, the timing constraints computed by convex polyhedra will be more precise and, therefore, less restrictive. Two indicators have been measured to quantify the difference of precision: the number of constraints required for correctness (C) and the number of states that satisfy these constraints (Sat). Intuitively, the second value hints the degree of restriction imposed by each set of constraints. In five examples, both approaches compute exactly the same constraints (noted as = in the Table). For the other examples, the constraints computed by octahedra are more restrictive. However, the collected data point out that the quality of the constraints computed by both methods is comparable: there are not many additional constraints, nor they are overly restrictive.

5. Conclusions

A technique for the generation of gate-level timing constraints in asynchronous circuits has been presented. Gate delays are parameters of the problem and the output timing constraints describe the linear inequalities that should be satisfied by the parameters to ensure correctness. Experimental results have shown that the kind of linear constraints that appear when analyzing timed circuits are represented more efficiently using octahedra than convex polyhedra. Still, the complexity is very dependent on the number of symbolic delays. Future work will attempt to improve the current representation so that it scales up for larger circuits.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, pages 3–34, 1995.

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

Table 1. Experimental results for the asynchronous controllers

Example	Circuit		STG		State space		Σ	Octahedra - This paper				Convex polyhedra			
	Wires	Gates	Places	Trans	States	Trans		C	Sat	CPU	Mem	C	Sat	CPU	Mem
nowick	10	7	19	14	60	119	20	=	=	0.0s	2.6Mb	2	45	0.8s	83Mb
gasp-fifo	9	7	10	8	66	209	12	11	22	4.1s	3.9Mb	10	28	8.1s	87Mb
sbuf-read-ctl	13	10	19	16	74	157	14	=	=	0.1s	2.9Mb	4	52	1.2s	83Mb
rcv-setup	9	6	14	15	72	187	12	=	=	0.4s	3.0Mb	8	49	2.1s	83Mb
alloc-outbound	15	11	21	22	82	161	19	4	61	0.1s	2.9Mb	3	62	1.3s	83Mb
ebergen	11	9	16	14	83	188	5	=	=	0.1s	2.9Mb	5	61	1,3s	83Mb
D flip-flop	6	4	16	22	146	448	8	=	=	1.6s	4.4Mb	7	112	5.8s	85Mb
mp-forward-pkt	13	10	24	16	194	574	12	8	82	0.3s	3.8Mb	6	89	1.9s	85Mb
chu133	12	9	17	14	288	1082	7	5	56	1.3s	5.5Mb	3	61	1.3s	85Mb
desynch	11	8	12	8	304	934	13	6	50	8.0s	4.4Mb	O/M	O/M	O/M	O/M
converta	14	12	16	14	396	1341	14	13	180	138s	15.0Mb	13	188	20.4s	92Mb

Table 2. Comparison of CPU time and peak memory in the asynchronous pipeline example.

Pipeline example				Convex Polyhedra [10]		OhDD [9]		This paper	
Stages	Σ	States	Trans	CPU	Mem	CPU	Mem	CPU	Mem
2	8	36	88	0s	64Mb	1s	5Mb	0s	1Mb
3	10	108	312	2s	67Mb	17s	8Mb	2s	3Mb
4	12	324	1080	13s	79Mb	249s	39Mb	12s	9Mb
5	14	972	3672	259s	147Mb	1h5min	57Mb	123s	48Mb
6	16	2916	12312	O/M	O/M	39h44min	83Mb	18min	245Mb
7	18	8748	40824	O/M	O/M	T/O	T/O	2h6min	1183Mb

Σ = number of symbolic delays O/M = out of memory (> 1.5Gb) T/O = timeout (> 48h)

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

[4] T. Amon, G. Borriello, T. Hu, and J. Liu. Symbolic timing verification of timing diagrams using Presburger formulas. In *Proc. Design Automation Conf.*, pages 226–231, 1997.

[5] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Computer Aided Verification*, pages 419–434, 2000.

[6] W. J. Belluomini and C. J. Myers. Timed circuit verification using TEL structures. *IEEE Transactions on Computers*, 20(1):129–146, 2001.

[7] S. Chakraborty, D. L. Dill, and K. Y. Yun. Min-max timing analysis and an application to asynchronous circuits. *Proceedings of the IEEE*, 87(2):332–346, 1999.

[8] T.-A. Chu. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications*. PhD thesis, MIT, June 1987.

[9] R. Clarisó and J. Cortadella. The octahedron abstract domain. In *Proc. Static Analysis Symp.*, pages 312–327, 2004.

[10] R. Clarisó and J. Cortadella. Verification of timed circuits with symbolic delays. In *Proc. of Asia and South Pacific Design Automation Conf.*, pages 628–633, 2004.

[11] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. Symp. on Principles of Programming Languages*, pages 238–252, 1977.

[12] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. Symp. on Principles of Programming Languages*, pages 84–97, 1978.

[13] G. Dantzig and B. Eaves. Fourier-motzkin elimination and its dual. *Journal of combinatorial theory*, 14:288–297, 1973.

[14] D. L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212. Springer-Verlag, 1989.

[15] N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.

[16] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In *Proc. REX Workshop Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 226–251, 1992.

[17] A. Miné. The octagon abstract domain. In *Analysis, Slicing and Tranformation (in Working Conference on Reverse Engineering)*, IEEE, pages 310–319. IEEE CS Press, 2001.

[18] C. J. Myers, W. Belluomini, K. Killpack, E. Mercer, E. Peskin, and H. Zheng. Timed circuits: A new paradigm for high-speed design. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 335–340, 2001.

[19] M. A. Peña, J. Cortadella, A. Kondratyev, and E. Pastor. Formal verification of safety properties in timed circuits. In *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pages 2–11, 2000.

[20] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. I. ato, and K. Jenkins. Asynchronous Interlocked Pipelined CMOS Circuits Operating at $3.3 - 4.5$GHz. In *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, pages 292–293, Feb. 2000.

[21] K. Stevens, R. Ginosar, and S. Rotem. Relative timing. In *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pages 208–218, 1999.

[22] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pages 46–53, 2001.

[23] F. Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures. In *Computer Aided Verification*, July 2004.

[24] T. Yoneda, T. Kitai, and C. Myers. Automatic derivation of timing constraints by failure analysis. In *Proc. Int. Conference on Computer Aided Verification*, pages 195–208, 2002.