Efficient Scaling-Invariant Checking of Timed Bisimulation

Carsten Weise and Dirk Lenzkes

Lehrstuhl für Informatik I,
Aachen University of Technology, Germany
{carsten|dlen}informatik.rwth-aachen.de

Keywords: algorithms and data structures, automata and formal languages, program specification and verification, decidability, real-time systems.

Abstract. Bisimulation is an important notion for the verification of distributed systems. Timed bisimulation is its natural extension to real time systems. Timed bisimulation is known to be decidable for timed automata using the so-called region technique. We present a new, top down approach to timed bisimulation which applies the zone technique from the theory of hybrid systems. In contrast to the original decision algorithm, our method has a better space complexity and is scaling invariant: altering the time scale does not effect the space complexity.

1 Introduction

Strong and weak bisimulation ([Mil89]) are useful notions of equivalence for the verification and analysis of distributed systems. Timed bisimulation is the suitable notion of bisimulation for real-time systems. We use timed graphs (or timed automata) ([NSY93,HNSY92,AD94,AC+95]) as the specification formalism for real time systems. Timed graph use the positive reals as time domain. While trace-based equivalences for timed automata are in general undecidable ([AD94]), timed bisimulation is known to be decidable ([Čer92]). The region-technique ([ACD93]) used in the algorithm leads to a high space-complexity, demanding for more efficient approaches. More recent publications on timed automata (e.g. [AC+95]) use a technique we shall refer to as *zone technique.* A zone is a union of regions. The space used by representations of convex zones (i.e. polyhedra) depends on the number of clocks of the automaton, but not on the size of the zone. Therefore keeping zones as large as possible will generally reduce the space complexity of an algorithm.

We will present a decision algorithm for timed bisimulation using the zone technique, which in practice will be more space efficient than the original algorithm. Our algorithm turns out to be *scaling invariant*: the number of zones computed by the algorithm is invariant against re-scaling the involved timed graphs. Re-scaling a timed graph is multiplication of all constants with a fixed factor, as e.g. used to change the resolution of the time scale.

Reischuk, Morvan (Eds.): STACS'97 Proceedings, LNCS 1200

For simplicity, we restrict our presentation to timed simulation (which is "half a bisimulation"). All challenging algorithmic problems are already present within this framework. Generalization to bisimulation is straightforward – though it has to be done with a certain care – and is discussed at the end of the paper together with possible applications and extensions of our algorithm.

We proceed as follows: after recalling the definitions of timed graphs and timed simulation, we give a new characterization of timed simulation. Then we introduce zones and zone graphs, which are the basis of our algorithm. We explain a general algorithm for strong and weak simulation, and discuss correctness and termination. The paper ends with a discussion of the implementation, the complexity, related work, applications and future work.

Due to lack of space we can only hint at how to prove our propositions. The reader is referred to [WL96] for a detailed treatment. In the formalism, impl. denotes implication, as we already use $\Rightarrow$ for the weak transition relation.

2 Timed Graphs

This section recalls the definition of timed graphs, the framework for the presentation of our algorithm. *Timed graphs* model the timing behavior of a real time system using a finite set of clocks, ranging over the positive reals. Timing constraints are expressed by formulae over these clocks. Legal formulae are tt (the value true) and finite conjunctions of inequalities comparing a natural number to a clock or a clock difference. All usual comparisons $(<, \leq, =, \geq, >)$ are admitted. For a given clock set $\mathcal{C}$, the set of these *simple linear formulae* (SLF) is denoted by $\Phi(\mathcal{C})$, with typical elements ϕ, ψ, etc. For a subset $R \subseteq \mathcal{C}$, $\phi_0(R)$ is the formula which requires all clocks in R to be zero, i.e. $\phi_0(R) = \bigwedge_{C \in R} C = 0$.

A timed graph is a directed, labeled graph, whose nodes (called *locations*) have an associated *invariant* and whose edges (called *transitions*) are labeled with an action, a *guard* and a *reset set*. Invariants and guards are SLF, a reset set is a subset of the clocks.

Definition 1 (Timed Graphs). A *Timed Graph* is a tuple $P = (N, n_0, \mathcal{C}, A, \rightarrow, \mathsf{Inv})$, where N is a finite set of *locations*, $n_0 \in N$ is the *initial location* of P, $\mathcal{C}$ is a finite set of *clocks*, A is a finite set of (synchronization) actions, $\rightarrow \subseteq N \times (A \times \Phi(\mathcal{C}) \times 2^{\mathcal{C}}) \times N$ is the transition relation, and $\mathsf{Inv} : N \rightarrow \Phi(\mathcal{C})$ are the *invariants* of the locations.

We will refer to a timed graph as a *process*. The transition relation of a process P is usually denoted by $\rightarrow_P$. A timed graph is an abstract description of a real time system. The system starts in its initial location with all clocks set to zero. All clocks increase at the same speed, measuring exactly the elapse of time. As long as the clocks' values meet the invariant of a location, the process may choose to stay within the location. A transition can be taken if its guard is valid for the current values of clocks. If a transition is taken, the annotated action occurs with duration zero. Afterwards all clocks in the reset set are set to zero, while the other clocks retain their values. This intuition is made precise using *valuation graphs*, the semantic model of timed graphs.

A valuation is a mapping of the clocks into the positive reals. The set of all valuations for a given clock set $\mathcal{C}$ is $\mathcal{V}(\mathcal{C}) := \{v \mid v : \mathcal{C} \to \mathbb{R}^{\geq 0}\}$, with typical elements v, w, etc.. The following are useful operations on valuations:

time step: $\forall C \in \mathcal{C}.\, (v + d)(C) := v(C) + d, d \in \mathbb{R}^{\geq 0}$
future: $v^{\uparrow} := \{v + d \mid d \in \mathbb{R}^{\geq 0}\}$
past: $v^{\downarrow} := \{v' \mid \exists d \in \mathbb{R}^{\geq 0}.\, v = v' + d\}$
reset: $[R \to 0]v := \begin{cases} \forall C \in R.\ [R \to 0]v(C) := 0 \\ \forall C \notin R.\ [R \to 0]v(C) := v(C) \end{cases}$
restriction: $v|_R \in \mathcal{V}(R)$ where $\forall C \in R.\, v|_R(C) = v(C)$
embedding: $v|^{\mathcal{C}'} := \{w \in \mathcal{V}(\mathcal{C}') \mid w|_{\mathcal{C}} = v|_{\mathcal{C}}\}, \mathcal{C} \subseteq \mathcal{C}'$

The full valuation graph represents the semantics of a timed graph:

Definition 2 (Full Valuation Graph). A Timed Graph $P = (N, n_0, \mathcal{C}, A, \to_P, \mathsf{Inv})$ defines a valuation graph $G = (S, s_0, L, \to)$ where $S := N \times \mathcal{V}(\mathcal{C})$ is the set of states, $s_0 := (n_0, \phi_0(\mathcal{C}))$ is the start state, $L := A \cup \mathbb{R}^{\geq 0}$ is the set of transition labels, and $\to$ is the transition relation defined by

$$(A)\quad \frac{n \xrightarrow{a,\phi,R}_P n', \phi(v) = \mathtt{tt}, v \in \mathsf{Inv}(n), [R \to 0]v \in \mathsf{Inv}(n')}{(n, v) \xrightarrow{a} (n', [R \to 0]v)}$$

$$(T)\quad \frac{d \in \mathbb{R}^{\geq 0}, \forall 0 \leq d' \leq d.\, v + d' \in \mathsf{Inv}(n)}{(n, v) \xrightarrow{d} (n, v + d)}$$

Let $S_0 \subseteq S$ be the set of states reachable from the start state s_0 in G. Then the *full valuation graph of* P is the graph $G_f = (S_0, s_0, L, \to \cap\, S_0 \times L \times S_0)$.

We usually will denote the transition relation of the full valuation graph of a process P by $\to_{VP}$. Often the states of valuation graphs will be called points.

The full valuation graph describes the complete behavior of a process. In the sequel, we often will concentrate on a partial description of the process' behavior. For this we use *valuation graphs*, which are subgraphs of the full valuation graph. As usually, a labeled transition system $T' = (S_0, L, \to_0)$ is a *subgraph* of $T = (S, L, \to)$ (written $T' \leq T$), if $S_0 \subseteq S$ and $\to_0 \subseteq S_0 \times L \times S_0 \cap \to$.

Valuation graphs are *two phase transition systems*: continuous phases, where time passes, alternate with discrete steps, where actions are observable while no time passes.

3 Timed Simulation

This section recalls the definition of *timed (forward) simulation* ([LV91]). Intuitively, a process Q simulates a process P if Q can match every step of P by a step with the same label. Formally this is defined for valuation graphs by:

Definition 3 (Strong Simulation). Given two valuation graphs $V_i = (S_i, s_i, L, \rightarrow_i)(i \in \{1,2\})$, a relation $\mathcal{R} \subseteq S_1 \times S_2$ is a *strong simulation* if for all pairs $(p,v)\, \mathcal{R}\, (q,w)$ and all $\ell \in L.\, (p,v) \xrightarrow{\ell}_1 (p',v')$ impl. $\exists(q',w').\, (q,w) \xrightarrow{\ell}_2 (q',w')$ and $(p',v')\, \mathcal{R}\, (q',w')$.

A state (q,w) *strongly simulates* (p,v) (written $(p,v) \lesssim (q,w)$) if there is a simulation $\mathcal{R}$ such that $(p,v)\, \mathcal{R}\, (q,w)$.

The process Q *strongly simulates* the process P if there is a strong simulation of their valuation graphs which includes their respective start states.

In practice, (weak) simulation is the more important notion. Weak simulation abstracts from internals of the individual processes. To define (weak) simulation, we need the notion of *weak timed transition relation*, which differs from the classical notion ([Mil89]) of the weak transition relation by the additonal laws $(W2)$ and $(W3)$. These laws guarantee *time additivity.*

Definition 4 (Weak Transition Relation). A *timed transition relation* is a transition relation with labels L where $\mathbb{R}^{\geq 0} \cap L \neq \emptyset$.

Given a timed transition relation $\rightarrow$ and a special silent action $\tau \in L$, the *weak timed transition relation* is the least relation $\Rightarrow$ satisfying:

$$(W1)\ \frac{s \xrightarrow{\tau^n}\xrightarrow{\ell}\xrightarrow{\tau^m} s', \ell \in L, n, m \in \mathbb{N}}{s \overset{\ell}{\Rightarrow} s'}$$

$$(W2)\ \frac{s \xrightarrow{\tau^n} s', n \in \mathbb{N}}{s \overset{0}{\Rightarrow} s'} \qquad (W3)\ \frac{s \overset{d}{\Rightarrow} s', s' \overset{d'}{\Rightarrow} s'', d, d' \in \mathbb{R}^{\geq 0}}{s \overset{d+d'}{\Rightarrow} s''}$$

Then *(weak) timed simulation* is defined by replacing $\rightarrow_\lambda$ by $\Rightarrow_\lambda$ $(\lambda \in \{P, Q\})$ in the definition of strong simulation. If Q simulates P, we write $P \lessapprox Q$. Note that weak ε-transitions of the classical definition are replaced by 0-transitions. We will abuse notation and write $n \xrightarrow{a(d)} n'$ if there is a n'' with $n \xrightarrow{d} n'' \xrightarrow{a} n'$, and analogously for $n \overset{a(d)}{\Rightarrow} n'$.

Our approach follows the idea of Čerāns to decide bisimulation of timed graphs by examining the product graph of the involved processes:

Definition 5 (Product of Timed Graphs). Given two Timed Graphs $P_i = (N_i, n_0^i, C_i, A, \rightarrow_i, \mathsf{Inv}_i)(i \in \{1,2\})$ with $C_1 \cap C_2 = \emptyset$, their *strong product* is the Timed Graph $G = P_1 \times P_2 = (N_1 \times N_2, n_0, C_1 \cup C_2, A, \rightarrow, \mathsf{Inv})$ where $n_0 = (n_0^1, n_0^2)$, and $\mathsf{Inv}(n_1, n_2) = \mathsf{Inv}(n_1) \wedge \mathsf{Inv}(n_2)$ for all (n_1, n_2), and $\rightarrow$ is defined by

$$(S)\ \frac{n_1 \xrightarrow{a,\phi_1,R_1}_1 n_1', n_2 \xrightarrow{a,\phi_2,R_2}_2 n_2'}{(n_1, n_2) \xrightarrow{a,\phi_1 \wedge \phi_2, R_1 \cup R_2} (n_1', n_2')}$$

The *weak product* is the graph $G = P_1 \times_w P_2 = (N_1 \times N_2, n_0, C_1 \cup C_2 \cup \{T\}, A \cup \{\tau_Q\}, \rightarrow, \mathsf{Inv})$ where $\rightarrow$ is defined by

$$(W)\ \frac{n_1 \xrightarrow{a,\phi_1,R_1}_1 n_1', n_2 \xrightarrow{a,\phi_2,R_2}_2 n_2', a \neq \tau}{(n_1,n_2) \xrightarrow{a,\phi_1\wedge\phi_2,R_1\cup R_2\cup\{T\}} (n_1',n_2')}$$

$$(P)\ \frac{n_1 \xrightarrow{\tau,\phi_1,R_1}_1 n_1'}{(n_1,n_2) \xrightarrow{\tau,\phi_1,R_1\cup\{T\}} (n_1',n_2)} \qquad (Q)\ \frac{n_2 \xrightarrow{\tau,\phi_2,R_2}_2 n_2'}{(n_1,n_2) \xrightarrow{\tau_Q,\phi_2,R_2} (n_1,n_2')}$$

The product graph requires actions to happen synchronously in both processes. The weak product will be used in the decision of weak simulation. It allows τ-transition to occur independently in each process. Note that the autonomous τ-moves of Q are marked by a special label τ_Q. Both τ and τ_Q are interpreted as internal actions of the product, but must be distinguishable in the algorithm. The fresh clock T will be explained in the Subsect. 5.1. Typically, points of the valuation graph of the product are written $(p \times q, v_P \times v_Q)$, the point which is composed from (p, v_P) and (q, v_Q). We introduce the notion of P-closedness in order to give a new characterization of timed simulation:

Definition 6 (P-closed). Let $\rightarrow_P, \rightarrow_V$ be the transition relations of the full valuation graphs of P and $P \times Q$. A point $s = (p \times q, v_P \times v_Q)$ is *strongly $P(0)$-closed* by definition, and *strongly $P(n+1)$-closed* if for all $\alpha \in A \cup \mathbb{R}^{\geq 0}$, $(p, v_P) \xrightarrow{\alpha}_P (p', v_P')$ implies there is q', v_Q' such that $s \xrightarrow{\alpha}_V s' = (p' \times q', v_P' \times v_Q')$ and s' is strongly $P(n)$-closed.
A point is *strongly P-closed*, if it is strongly $P(n)$-closed for every $n \in \mathbb{N}$. Replacing $\rightarrow_V$ by $\Rightarrow_V$ yields the definition of *(weakly) $P(n)$-closed* and *(weakly) P-closed.*

Intuitively, a point $s = (p \times q, v_P \times v_Q)$ is $P(n+1)$-closed if it can match each step required by (p, v_P), reaching a $P(n)$-closed point. Thus if s is (strongly) P-closed, (q, v_Q) can (strongly) simulate (p, v_P).

Let n be a $P(n+1)$-closed point, then a set M of points is called *matching closed* for n if it contains all the endpoints s' as in Def. 6 necessary to establish $P(n+1)$-closedness. M is matching closed for a set M' if it is matching closed for every point in M'. Using this notion, existence of a simulation relation can be reduced to $P(1)$-closedness:

Theorem 7. *$P \lesssim Q$ iff there is a valuation graph of $P \times Q$ where all points are strongly $P(1)$-closed. $P \lessapprox Q$ iff there is a valuation graph of $P \times_w Q$ and a subset M of its nodes (including the start node) such that all points in M are $P(1)$-closed and M is matching closed for M itself.*

The proof is straightforward, using Čerāns' decidability result. We will use this theorem to establish the correctness of our algorithm.

4 Zone Graphs

The *zone technique* represents valuation graphs by *zone graphs*. This section defines zone graphs and especially *backward stable zone graphs* which are the basis of our decision procedure.

Definition 8 (Zone). The *characteristic set* of a simple linear formula ϕ is the set of all valuations for which ϕ holds.
A *zone* is a finite union of characteristic sets.

All operations on valuations (see page 179) can be extended to zones by taking the union of the pointwise application. In the sequel, we will identify characteristic sets with their generating formulae. We will also write $\phi_1 \vee \ldots \vee \phi_n$ for the finite union of the characteristic sets of the ϕ_i. The set of all zones is written $\Phi_{\vee}(\mathcal{C})$.

A *zone graph* is a graph where each node consists of a location and a zone:

Definition 9 (Zone Graph). For a process $P = (N, p_0, \mathcal{C}, A, \rightarrow_P, \mathsf{Inv})$, a *zone-graph* is a transition system $(S, s_0, A, \rightarrow)$, where $S \subseteq N \times \Phi_{\vee}(\mathcal{C})$, $s_0 = (p_0, \phi_0(\mathcal{C}))$ and $\rightarrow \subseteq S \times A \times S$ is connected.

Zone graphs represent valuation graphs. If a node $n = (p, \psi)$ is present in a zone graph, then for every point s in n, all admissible time steps $s \xrightarrow{d} s'$ are in the valuation graph. If additionally an edge $n \xrightarrow{a} n'$ is present, then all transitions $s' \xrightarrow{a} s''$ with s'' in n' are in the valuation graph:

Definition 10 (Valuation Graph of a Zone Graph). Given a process P, its full valuation graph with transition relation $\rightarrow_{VP}$, and a zone graph $Z = (S, s_0, A, \rightarrow)$, the valuation graph of Z is defined by:

$$\frac{(p,\psi) \xrightarrow{a} (p',\psi'), (p,v) \xrightarrow{a}_{VP} (p',v'), v \in \psi^{\uparrow} \wedge \mathsf{Inv}(p), v' \in \psi'}{(p,v) \xrightarrow{a}_{V} (p',v')}$$

$$\frac{(p,\psi) \in S, v, v+d \in \psi^{\uparrow} \wedge \mathsf{Inv}(p)}{(p,v) \xrightarrow{d}_{V} (p, v+d)}$$

Note that the valuation graph of a zone graph Z cannot be constructed from Z alone, but knowledge of the underlying timed graph is necessary. We will abuse notation and write $G \leq G'$ for two zone graphs if the valuation graph of G is a subgraph of the valuation graph of G'. For two nodes $(n,\psi), (n',\psi')$ of a zone graph, we will write $(n,\psi) \subseteq (n',\psi')$ iff $n = n'$ and $\psi \subseteq \psi'$.

A zone graph is *backward stable* (short: BS-graph) if along a transition $n \xrightarrow{a} n'$ every point in n' is reachable from some point in n:

Definition 11 (BS-Graph). A zone graph $Z = (N, n_0, A, \rightarrow)$ with valuation graph $V = (S, s_0, A \cup \mathbb{R}^{\geq 0}, \rightarrow)$ is called *backward stable* if $(p,\psi) \xrightarrow{a} (p',\psi')$ implies $\forall v' \in \psi'. \exists v \in \psi^{\uparrow}. (p,v) \xrightarrow{a}_{V} (p',v')$.

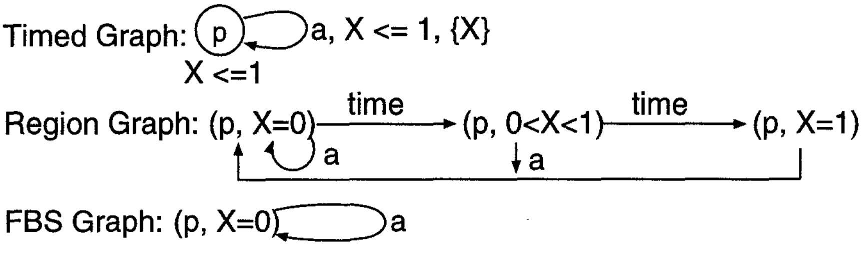

Fig. 1. Timed Graph with Region and FBS-graph

A zone graph is a *full BS-graph* (short: FBS-graph) if it represents the full valuation graph of a process. As FBS-graphs are used in the decision algorithm, we present a construction method for FBS-graphs. For a given zone ψ and an edge $e = p \xrightarrow{a,\phi,R} p'$ of P, the e-successor of ψ is the set of all valuations reachable from (p, ψ) via e: $\mathsf{succ}(\psi, e) := \{v' \mid \exists d \in \mathbb{R}^{\geq 0}.\ (p,v) \xrightarrow{d}_{VP} (p, v+d) \xrightarrow{a}_{VP} (p', v'), \phi(v+d) = \mathtt{tt}, v' = [R \to 0]v\}$.
The e-successor can be computed using the operations on zones: $\mathsf{succ}(\psi, e) = [R \to 0](\phi \wedge \psi^{\uparrow} \wedge \mathsf{Inv}(p)) \wedge \mathsf{Inv}(p')$. An FBS-graph is constructed by starting from the initial node n_0. For every node $n = (p, \psi)$ already in the graph and every outgoing edge e of p, the transition $n \xrightarrow{a} (p', \mathsf{succ}(\psi, e))$ is added, until no more new nodes are generated.

In order to compare BS-graphs to the region graphs used by the original algorithm, we recall the definition of regions:

Definition 12 (Region). For a constant $k \in \mathbb{N}$, $\Phi_k(C)$ is the set of all simple linear formula over C with constants less than k. A *region* is then the smallest zone describable by a formula from $\Phi_k(C)$.
By $\Gamma_k(C)$ we denote the set of all regions w.r.t. $\Phi_k(C)$.

For a given timed graph, the constant c is chosen to be the largest constant appearing in the guards and the invariants of the timed graph. Fig. 1 gives an example of a process and its region- and FBS-graph. In many cases, the FBS-graph constructed by the method given above will be much smaller than the corresponding region graph. The size of the FBS-graph is scaling invariant. Multiplying all constants by a factor c in Fig. 1 will result in an FBS-graph of the same size, while the region graph will have $2 * c + 1$ nodes. As any finite union of regions is a zone, the number of zones is greater than the number of regions. Thus in principal a zone graph can be larger than a region graph. However this can be avoided by modifying the construction so that all nodes are disjoint, i.e. $\psi \cap \psi' = \emptyset$ for any pair (p, ψ) and (p, ψ'). This is achieved by splitting a node (p, ψ) already in the graph if a node (p, ψ') with $\psi \cap \psi' \neq \emptyset$ is added.

The result of this construction will no longer be backward stable, but has always a size less than or equal to the size of the region graph. Note that there is no way to prevent the worst case, where the zone and the region graph are identical.

5 The Decision Algorithm

Our algorithm uses a top down approach: starting from an FBS-graph G of $P \times Q$ (resp. $P \times_w Q$), all points which are not $P(1)$-closed in G are deleted. If a point is deleted, all its predecessors must be re-examined. The algorithm stops if either there is a set M of $P(1)$-closed points which is matching w.r.t. itself, or if the the start point is removed from G. In the latter case, Q cannot simulate P, while in the former the resulting graph represents a simulation relation.

For a point s, let $\Delta(s) := \{d \in \mathbb{R}^{\geq 0} \mid s \xrightarrow{d}\}$ be its admissible time steps. A point $s = (p \times q, v_P \times v_Q)$ is $P(1)$-closed if matching steps from s can be found for all $d \in \Delta(p, v_P)$ and for all action transitions leaving (p, v_P). For the matches of time steps it is sufficient to find matches which are on the same path in G (see Lemma 15 below). Two points are on the same path in a valuation graph if one of them is reachable from the other by a time step. To formalize this intuition we introduce the notion of *time sequences.* The points of a time sequence are the matches for the time steps of (p, v_P), and thus must stay within the control location p while the control location of Q may change. Time sequences are defined over a *grounded interval.* An interval of $\mathbb{R}^{\geq 0}$ is grounded if it is left-closed starting at zero, i.e. either an interval $[0, t]$ or $[0, t)$ or $\mathbb{R}^{\geq 0}$ itself:

Definition 13 (Time Sequence). Let $V = (S, s_0, L, \rightarrow)$ be a valuation graph of a (weak) product graph, and $s = (p \times q, v_P \times v_Q)$ a point in V. For a grounded interval I, a mapping $\delta : I \rightarrow S$ which fulfills $\delta(0) = s$ and $\forall t \in I. \exists q', v'_Q. \delta(t) = (p \times q', (v_P + t) \times v'_Q)$ and $\forall t < t' \in I. \delta(t) \xrightarrow{t'-t}_V \delta(t')$ is called a *strong time sequence* of s. The definition of a *(weak) time sequence* is yielded by replacing $\rightarrow_V$ by $\Rightarrow_V$ in the last requirement.

Assume a path $(p \times q, v) \xrightarrow{1} (p \times q, v+1) \xrightarrow{\tau} (p \times q', w) \xrightarrow{2} (p \times q', w+2)$ in V, then δ with $\delta(t) = (p \times q, v+t)(t \in [0,1])$ is a strong time sequence, and δ with $\delta(t) = (p \times q, v+t)(t \in [0,1))$ and $\delta(1+t) = (p \times q', w+t)(t \in [0,2])$ is a (weak) time sequence.

The straightforward approach to testing $P(1)$-closedness would be to find a time sequence of matching points for s, and additionally to find matches for the actions required in s. However we will require more: we call a point *good,* if we can find matches for the actions required by all the points in the time sequence:

Definition 14 (Good Points). Let V be a valuation graph of a product graph, $s = (p \times q, v_P \times v_Q)$ a point in V, and $I := \Delta(p, v_P)$. The point s is *strongly good* if there is a strong time sequence $\delta : I \rightarrow N$ such that $\forall d \in I, a \in A.(p, v_P + d) \xrightarrow{a}_P (p', v'_P)$ impl. $\exists q', v'_Q. \delta(d) \xrightarrow{a}_V (p' \times q', v'_P \times v'_Q)$.
The property *(weakly) good* is defined by replacing $\rightarrow_V$ by $\Rightarrow_V$.

```
INPUT: processes $P$ and $Q$

 1. compute the (weak) product of $P$ and $Q$
 2. compute an FBS-graph $G$ of the (weak) product
 3. find a subgraph of $G$ where all relevant nodes are good by
    (a) mark the start node as relevant
    (b) initialize the list $ex$ with the start node of $G$, and the list $pcl$ with the
        empty set
    (c) repeat
          i. remove node $n$ from $ex$
         ii. if $n$ is not marked relevant, put all predecessors into $ex$ (removing
             them from $pcl$ if necessary) and start with the next iteration, else
        iii. compute $n'$, the node containing the maximal subset of good points
             in $n$, marking all nodes in $\mathsf{rel}(n)$ as relevant,
         iv. if ($n' \neq n$) replace $n$ by $n'$ in $G$, and put all predecessors of $n$ into $ex$
             (removing them from $pcl$ if necessary)
          v. add $n'$ to $pcl$
        until $ex$ empty or the start node is no longer in $G$
```

Fig. 2. Generic Algorithm For Strong and Weak Simulation

A *good sequence* is a time sequence used to establish that a point is good. All points of the time sequence of a good point are $P(1)$-closed:

Lemma 15. *If a point s of a product graph is (strongly) good, then all points $\delta(t)$ of the good sequence of s are (strongly) $P(1)$-closed. If a point s and all points reachable by time steps from s are (strongly) $P(1)$-closed, then s is (strongly) good.*

The proof of the first implication is straightforward. In the proof of the second implication, the existence of matching points for every time step of s follows from the definition of $P(1)$-closedness. It it not completely obvious that these points form a time sequence. The proof in [WL96] uses the region graph of the product to give a finite construction method for the time sequence.

Instead of testing all points of the valuation graph of G for $P(1)$-closedness, it is sufficient to test if the points of the nodes of G are good. In the case of weak simulation, we further need to keep track of the matching points. The matching points are the end points $(p' \times q', v'_P \times v'_Q)$ of the a-transitions in Def. 14. Our algorithm can now be described as in Fig. 2. The only problems left are how to compute the maximal set of good points and the set $\mathsf{rel}(n)$ in step *(iii)*. We will explain this briefly in the next subsection.

5.1 Computing Good Points

Details of the computation of good points can be found in [WL96]. All computations use the operations on zones as defined on page 179. The principal idea is to compute two sets of points: in the set ur are all the points which are required to be reachable by time steps by process P but which are not reachable in the product graph, while the set op has all points which are reachable by time steps, but which cannot match all required actions. A point is good if it is not in the past of these two sets.

In the strong case, these sets are easily determined as due to time-determinism all points of a good time sequence are uniquely determined. In the weak case, all maximal paths consisting of τ_Q-steps only may contain good time sequences. The good points are computed relative to these paths. This done by stepping backwards from the end points of the paths to their start points, iterating the computation used for the strong case along the nodes. The additional clock T is needed as timing information in Q may get lost due to clock resets along τ_Q-transitions. Note that the necessity of an additional clock for checking timed bisimulation was already shown in [ACH94].

In the strong case, all nodes can be marked as relevant. In the weak case, only the end nodes of paths needed to find weak action steps are marked relevant.

5.2 Correctness and Termination

The correctness of the algorithm follows from Theorem 7 and Lemma 15. With the definitions we have given so far termination of our algorithm is however not guaranteed, as the number of zones is in fact infinite. Our implementation uses a closure operation on zones which is also present in the region technique. Let c be the largest constant occurring in guards and invariants of the processes. Then there is no need to distinguish between the values of a clock which are greater than this constant c. Formally, the closure operation is defined by:

Definition 16 (Closure). Given a zone $\psi \in \Phi_{\vee}(C)$ and the set of regions $\Gamma_k(C)$, the *closure of* ψ *w.r.t.* $\Gamma_k(C)$ is defined by $\mathsf{cl}(\psi) := \bigcup\{\gamma \in \Gamma_k(C) \mid \gamma \cap \psi \neq \emptyset\}$. Let $\mathsf{cl}_k(C)$ be the set of the closures of the zones from $\Phi_{\vee}(C)$ w.r.t. $\Gamma_k(C)$.

As the number of regions is finite, so is $\mathsf{cl}_k(C)$. Thus using $\mathsf{cl}(\psi)$ instead of ψ for all zones in our algorithm ensures termination, while it can be shown that this does not disturb the correctness of the algorithm. Due to space limitations we cannot present the details here.

6 Conclusion

We presented a new algorithm for deciding timed simulation, improving the original result of Čerāns.. Timed bisimulation and its generalization, timed weak refinement ([CGL93]), can be decided as well. For bisimulation, nodes must be tested for P/Q-closedness, which is closedness for P and Q at the same time. As the operations on zones we use are scaling invariant, so is our algorithm. From the remarks on page 183 it follows that our algorithm always has a better space complexity than the region technique method. The example in Fig. 1 gives a

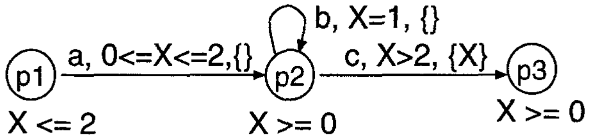

Fig. 3. Timed Graph with a small "simple" FBS-graph

hint on how to construct examples where it has a much better space complexity. We have implemented the algorithm in C++. Our implementation is on-the-fly, which even improves performance if Q cannot simulate P. It outperforms EPSILON ([CGL93]) – the only existing implementation of cerans's algorithm – by a factor of 80 even for simple examples, where the region graph is only double the size of the FBS graph.

In general, our implementation represents zones as union of polyhedra (i.e. convex zones), and polyhedra are implemented as difference bounds matrices (see [Dil89]). The zone ψ of a node n is instead stored as a list of zones $\psi_1, \ldots \psi_n$ such that $\psi = \psi_1 \setminus \psi_2 \setminus \ldots \psi_{n-1} \setminus \psi_n$. This representation comes natural, as the set of good points is in fact computed as a set difference between the original zone and the zone containing the "bad points". Whenever a node is diminished, a new ψ_i is added to the list. The representation has two advantages: first, the set difference of two polyhedra ψ_1, ψ_2 is in general a union of more than two polyhedra, so storing ψ_1, ψ_2 instead reduces the space used by the algorithm. Second, in the on-the-fly implementation to determine if a newly computed node is already in the graph we compare the new node's zone to ψ_1, the original zone of an old node. By this we avoid an infinite loop in which a node (p, ψ) is added, diminished to (p, ψ'), and then (p, ψ) is added again, and so forth. This again demonstrates that the main strenght of the algorithm lies in fact that we try to avoid splitting zones whenever possible.

The dual notion to backward stable is forward stable. Region graph are always forward stable. A node is stable if it is backward and forward stable. The algorithms given in [LY93,AC+95] for minimizing timed automata up to bisimulation use stable nodes. This indicates that our algorithm differs substantially from the known approaches to bisimulation, as we prefer to avoid splitting of zones instead of keeping (backward) stability of nodes. Note that the "improved" construction of an FBS-graph on page 183 can have a space complexity which is worse than the "simple" construction on top of page 183. Fig. 3 gives an example of a timed graph for which this is true. We plan to investigate this problem in more detail. While our algorithm reduces the space complexity in many practical cases, it cannot however be better than Čerāns' algorithm in the worst case. Note that our algorithm also tries to reduce space complexity in favor of time complexity, as advised in [HKV96]. We will use our implementation as a back end for the constraint oriented methodology ([LSW95]).

Acknowledgements We would like to thank Karlis Čerāns for pointing out that an efficient algorithm for timed bisimulation need to be scaling invariant.

References

[ACD93] R. Alur, C. Courcoubetis, D. Dill. Modelchecking in dense real-time. Information and Computation, 104(1):2-34, May 1993.

[AC+95] R. Alur,C. Courcoubetis, N. Halbwachs, T.A. Henzinger, et al. The algorithmic analysis of hybrid systems. TCS, February 1995.

[ACH94] R. Alur, C. Courcoubetis, T.A. Henzinger. The observational power of clocks. CONCUR 94, LNCS 836, Springer 1994, pp. 162-177.

[AD94] R. Alur, D.L. Dill. A Theory of Timed Automata. in: Theoretical Computer Science Vol. 126, No. 2, April 1994, pp. 183-236.

[AHV93] R. Alur, T.A. Henzinger, M.Y. Vardi. Parametric real-time reasoning. Proc. 25th STOC, ACM Press 1993, pp. 592-601.

[Čer92] K. Čerāns. Decidability of Bisimulation Equivalences for Parallel Timer Processes. in: Proc. CAV '92, LNCS 663, pp. 302 - 315.

[CGL93] K. Čerāns, J.C. Godsken, K.G. Larsen. Timed Modal Specification - Theory and Tools. in: Proc. CAV '93, LNCS 697, pp. 253–267.

[Dil89] D.L. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. in: LNCS 407, Springer Berlin 1989, pp. 197-212.

[HKV96] T.A. Henzinger, O. Kupferman, M.Y. Vardi. A Space-Efficient On-the-fly Algorithm for Real-Time Model Checking. in: Proc. CONCUR 96.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-time Systems. LICS '92, pp. 1-13.

[LY93] D. Lee, M. Yannakakis. An efficient algorithm or minimizing real-time transition systems. CAV '93, LNCS 697, Springer Berlin 1993, pp. 210–223.

[LSW95] K.G. Larsen, B. Steffen, C. Weise. Fischer's Protocol Revisited: A Simple Proof Using Modal Constraints. in: LNCS 1066, Springer 1996.

[LV91] N. Lynch, F. Vaandrager. Forward and Backward Simulations for Timing-Based Systems. in: REX Workshop, LNCS 600, pp. 397–446, 1991.

[Mil89] R. Milner. Communication and Concurrency. Prentice-Hall, 1989.

[NSY93] X. Nicollin, J. Sifakis, S. Yovine. From ATP to Timed Graphs and Hybrid Systems. Acta Informatica 30, 1993, S. 181-202.

[WL96] C. Weise, D. Lenzkes. A Fast Decision Algorithm for Timed Refinement. AIB 96-11, Technical Report University of Tech. Aachen, 1996.