Fundamenta Informaticae XX (2007) 1–21
IOS Press

Path Compression in Timed Automata[*]

Agata Janowska
Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warsaw, Poland
email: janowska@mimuw.edu.pl

Wojciech Penczek
Institute of Computer Science, PAS, Ordona 21, 01-237 Warsaw, Poland
Institute of Informatics, Podlasie Academy, 3 Maja 54, 08-110 Siedlce, Poland
email: penczek@ipipan.waw.pl

Abstract. The paper presents a method of abstraction for timed systems. To extract an abstract model of a timed system we propose to use static analysis, namely a technique called path compression. The idea behind the path compression consists in identifying a path (or a set of paths) on which a process executes a sequence of transitions that do not influence a property being verified, and replacing this path with a single transition. The method is property driven since it depends on a formula in question. The abstraction is exact with respect to all the properties expressible in the temporal logic $\mathrm{CTL}^*_{-\mathrm{X}}$.

1. Introduction

Many verification methods of time-critical systems have been defined over the last ten years. Model checking seems to be one of the most important methods due to its practical applications. Essentially, the method consists in determining whether a temporal formula expressing a property of a system is valid on its model (the state space) representing all the possible executions. Model checking has an advantage of being automatic but suffers from the so-called *state explosion problem*, i.e., the number of states in the model of a system can grow exponentially in the size of the system [24]. Many methods have been recently suggested to reduce state spaces that are considered in the verification process (e.g. [4, 9, 10, 21]).

In this paper we deal with abstract models for timed automata [1, 15] enriched with additional integer variables. Such automata are a common input formalism of model checkers for timed systems (e.g.

[*]The work is partly supported by the Polish grant No. 3T11C01128.

UppAal [22][1] and HyTech [14]). To extract an abstract model of a system we propose to use a static analysis, namely a technique called *path compression*. The idea behind it consists in identifying a path (or a set of paths) on which an automaton executes a sequence of transitions that do not influence a property being verified, and replacing this path with a single transition. The path compression has the similar effect to partial order reductions of excluding some possible interleavings, but in our approach several steps can be replaced by a single one, which partial order reductions do not do.

Our method is property driven since it depends on a formula in question. An abstraction is *exact* with respect to all the properties expressible in the temporal logic $\mathrm{CTL}^*_{-\mathrm{X}}$. which means that a given property holds in the abstract model if and only if it holds in the concrete one. To this aim we show that the original and the reduced model are in a stuttering bisimulation relation [7].

Path compression as an abstraction method is known for untimed systems. In [26] a method called *path reduction* is used to reduce state spaces by compressing computation paths of parallel while programs. The authors of [18] use the concept of *path slicing* to reduce control flow automata. The technique determines which subset of transitions along a given path to some particular location is relevant to reachability of the location at the path. We present the path compression technique for timed systems modeled in Intermediate Language of the verification tool VerICS in [17].

The closest to our work is [6], which deals with static analysis in verification of timed systems concerning the concept of *influence information*. The technique can be understood as slicing I/O Timed Components, i.e., timed automata extended with interfaces. The approach preserves the branching structure of a transition system up to the propositional assignment given over the external observer. Other related methods include the *active-clock reduction* technique [12] and a more general *relevant guard abstraction* [2]. Both the above works focus on reduction of the number of clocks and clock constraints. Statical analysis of control structure of timed automata is also used in [20] to minimize the total number of states to be saved during reachability analysis. In [27] the idea of cumulation of transition delays – similar to ours – is used to reduce timed Petri nets.

Our first attempt of using static analysis for timed automata concerns the slicing technique [16]. In short, the method of *program slicing* [25] consists in pointing the so-called *slicing criterion*, which identifies program points of interest and then tracing items of code on which it depends. The technique allows to eliminate data irrelevant to the property in question. In the approach presented in [16] a path compression is also possible, but only for paths along which the time cannot elapse. The current paper presents a general method for compressing timed and untimed paths. It is worth pointing out that the best results are achieved by combining slicing and path compression.

The rest of the paper is structured as follows. Section 2 recalls the formalism of timed automata. In Section 3 we present a motivating example of a timed system and give an informal overview of our method. Section 4 introduces the path compression technique for a network of timed automata. Experimental results are given in Section 5. The last section shortly concludes the paper.

2. Model and its semantics

In this section we introduce timed automata extended with integer variables. To this aim, we first define arithmetic and boolean expressions, as well as clock constraints.

[1]The definition of timed automata used in UppAal [5] is slightly different than ours, but this is insignificant to our method.

Variables. Let V be a finite set of integer variables. The set $Expr(V)$ of all the *arithmetic expressions* over V is defined by the following grammar:

$$expr ::= m \mid y \mid expr \ \oplus \ expr \mid -\,expr \mid (expr)$$

where $m \in \mathbb{Z}$, $y \in V$, and $\oplus \in \{-, +, *, /\}$.[2] The set $\Phi(V)$ of all the *boolean expressions* over V is defined inductively as follows:

$$\phi ::= true \mid expr \ \sim \ expr \mid \phi \ \wedge \ \phi \mid \phi \ \vee \ \phi \mid \neg\phi \mid (\phi)$$

where $expr \in Expr(V)$ and $\sim \in \{=, \neq, <, >, \leq, \geq\}$. The set $Act(V)$ of all the *actions* over V is defined as follows:

$$\alpha ::= \ \varepsilon \mid y := expr \mid \alpha; \alpha$$

where ε denotes an empty sequence, $y \in V$, and $expr \in Expr(V)$.

Clocks. Let X be a finite set of real variables, called *clocks*. The set $\Psi(X)$ of all the *clock constraints* over X is defined by the following grammar (we deal with diagonal-free automata):

$$\psi ::= true \mid x \sim c \mid \psi \wedge \psi$$

where $x \in X, c \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$.

Definition 2.1. (Syntax). A *timed automaton* is a tuple $\mathcal{TA} = (\Sigma, Q, q^0, V, X, T, \mathcal{I})$, where

- Σ is a finite set of *labels*,
- Q is a finite set of *locations*,
- q^0 is the *initial location*,
- V is a finite set of integer variables,
- X is a finite set of clocks,
- $T \subseteq Q \times \Sigma \times \Phi(V) \times \Psi(X) \times \{true, false\} \times Act(V) \times 2^X \times Q$ is a *transition relation*, and
- $\mathcal{I} : Q \longrightarrow \Psi(X)$ is an *invariant* function.

An element $t = (q, l, \phi, \psi, u, \alpha, Y, q')$ of T denotes a transition from the location q to the location q', where l is the label of transition t, ϕ and ψ define the enabling conditions (the guard and the clock constraint) for t, α is the action to be executed, Y is the set of clocks to reset (reset set), and u is the urgency attribute – when it is set, the transition t has to be executed as soon as it is enabled (t is enabled if the automaton is in the location q and the guard ϕ evaluates to $true$). We assume that clock constraints of urgent transitions are equal to $true$[3]. The invariant function assigns to each location $q \in Q$ a clock constraint defining the condition under which the automaton can stay in q. We write *source(t), label(t), guard(t), delay(t), urgency(t), action(t), reset(t),* and *target(t)* for q, l, ϕ, ψ, u, α, Y, and q', respectively. Several examples of timed automata are presented in Fig. 1-3 in Section 3.

[2]By / we denote integer division and $\mathbb{Z}$ denotes the set of integer numbers, $\mathbb{N}$ — the set of natural numbers, and $\mathbb{R}_+$ — the set of non-negative real numbers.

[3]The assumption is necessary to keep the clock constraint representable by convex zones (see [5] for details).

Variable Valuation. We define a *variable valuation* to be a total mapping $v : V \longrightarrow \mathbb{Z}$. We extend this mapping to expressions in the usual way. Satisfiability of a boolean expression $\phi \in \Phi(V)$ by a valuation v (we write $v \models \phi$) is defined as follows: $v \models true$, $v \models e_1 \sim e_2$ iff $v(e_1) \sim v(e_2)$, $v \models g_1 \wedge g_2$ iff $v \models g_1$ and $v \models g_2$, $v \models g_1 \vee g_2$ iff $v \models g_1$ or $v \models g_2$, and $v \models \neg g$ iff $v \not\models g$. Let α be an action, that is a sequence of assignments. By $v(\alpha)$ we denote the valuation after execution of an action α on a valuation v. We define $v(\alpha)$ to be the valuation v' defined inductively as follows:

- if $\alpha = \varepsilon$, then $v' = v$,
- if $\alpha = (y := expr)$, then $v'(y) = v(expr)$ and $v'(z) = v(z)$ for $z \in V \setminus \{y\}$,
- if $\alpha = \alpha_1; \alpha_2$, then $v' = v''(\alpha_2)$, where $v'' = v(\alpha_1)$.

Clock Valuation. A *clock valuation* is a total mapping $\tau : X \longrightarrow \mathbb{R}_+$. Satisfiability of a clock constraint $\psi \in \Psi(X)$ by a given clock valuation τ (denoted as $\tau \models \psi$) is defined inductively as follows: $\tau \models true$, $\tau \models x \sim c$ iff $\tau(x) \sim c$, $\tau \models \psi_1 \wedge \psi_2$ iff $\tau \models \psi_1$ and $\tau \models \psi_2$.

For $\delta \in \mathbb{R}_+$, $\tau + \delta$ denotes the clock valuation τ' such that $\tau'(x) = \tau(x) + \delta$ for all $x \in X$. Moreover, by $reset_Y(\tau)$ we denote the clock valuation τ' such that $\tau'(x) = 0$ for $x \in Y$ and $\tau'(x) = \tau(x)$ for $x \in X \setminus Y$. Finally, by τ^0 we denote the *initial* clock valuation such that $\tau^0(x) = 0$ for all $x \in X$.

Definition 2.2. (Semantics). *Semantics* of a timed automaton $\mathcal{TA} = (\Sigma_A, Q, q^0, V, X, T, \mathcal{I})$ for the initial valuation $v^0 : V \rightarrow \mathbb{Z}$ is a labeled transition system $\mathcal{S} = (S, s^0, \Sigma, \longrightarrow)$, where:

- $S = \{(q, v, \tau) \mid q \in Q \wedge v \in \mathbb{Z}^{|V|} \wedge \tau \in \mathbb{R}_+{}^{|X|} \wedge \tau \models \mathcal{I}(q)\}$ is the set of *states*,
- $s^0 = (q^0, v^0, \tau^0) \in S$ is the *initial state*,
- $\Sigma = \Sigma_A \cup \mathbb{R}_+$ is the set of labels,
- $\longrightarrow \subseteq S \times \Sigma \times S$ is the smallest transition relation defined by the following rules:
  - $(q, v, \tau) \xrightarrow{l} (q', v', \tau')$ iff there exists a transition $t = (q, l, \phi, \psi, u, \alpha, Y, q') \in T$ such that $v \models \phi$, $\tau \models \psi$, $v' = v(\alpha)$, $\tau' = reset_Y(\tau)$, and $\tau' \models \mathcal{I}(q')$
  - $(q, v, \tau) \xrightarrow{\delta} (q, v, \tau + \delta)$ iff $\delta \in \mathbb{R}_+$, $\tau + \delta \models \mathcal{I}(q)$, and for all transitions $t = (q, l, \phi, \psi, u, \alpha, Y, q') \in T$, $u = false$ or $v \not\models \phi$.

Initially, all the variables have their initial values set and all the clocks are set to 0. At a state $s = (q, v, \tau)$ the system can either:

- execute an enabled transition t and move to the state $s' = (q', v', \tau')$, where q' is the target of t, the variable valuation is changed according to the action of t and the clocks from the reset set of t are set to 0, or
- let time δ pass and move to the state $(q, v, \tau + \delta)$, as long as no urgent transition is enabled and $\tau + \delta$ satisfies $\mathcal{I}(q)$.

Runs. For $(q, v, \tau) \in S$, let $(q, v, \tau) + \delta$ denote $(q, v, \tau + \delta)$. An $s-$*run* of $\mathcal{T}A$ is a sequence of states: $s_0 \overset{a_0}{\to} s_1 \overset{a_1}{\to} s_2 \overset{a_2}{\to} \ldots$, where $s_0 = s \in S$ and $a_i \in \Gamma \cup \mathbb{R}_+$ for each $i \geq 0$. A run is called *progressive* (or *divergent*) iff $\sum_{\{i \in \mathbb{N} \mid a_i \in \mathbb{R}_+\}} a_i$ is infinite. $\mathcal{T}A$ is *progressive* if all its s^0-runs are progressive.

Parallel Composition. We assume that a system to be verified is described as a set of automata running parallel. Automata communicate with each other via shared variables. We assume *multi-synchronization* which requires that the transitions with a shared label have to be executed synchronously by all the automata containing this label.

Let $\{\mathcal{T}A_1, \ldots \mathcal{T}A_n\}$ be a set of timed automata and $\Sigma(l) = \{ 1 \leq i \leq n \mid l \in \Sigma_i \}$ denote the set of the numbers of the automata containing the label l. By $\{ a_j \}_{j \in Z}$ we denote the sequence of the actions a_j, where $j \in Z \subseteq \{1, \ldots, n\}$, such that the actions are arranged according to the numerical order of their indices. We assume additionally that synchronous transitions do not update shared[4] variables.

Definition 2.3. (Parallel Composition). Let $\mathcal{T}A_i = (\Sigma_i, Q_i, q_i^0, V_i, X_i, E_i, \mathcal{I}_i)$ for $i = 1, 2, \ldots, n$ be timed automata such that $X_i \cap X_j = \emptyset$ for all $i \neq j$. A parallel composition of the above automata (denoted as $\mathcal{T}A_1 \parallel \mathcal{T}A_2 \parallel \ldots \parallel \mathcal{T}A_n$) is the timed automaton $\mathcal{T}A = (\Sigma, Q, q^0, V, X, E, \mathcal{I})$, where:

- $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$,
- $Q = \prod_{i=1}^{n} Q_i$,
- $q^0 = (q_1^0, \ldots, q_n^0)$,
- $V = \bigcup_{i=1}^{n} V_i$,
- $X = \bigcup_{i=1}^{n} X_i$,
- $((q_1, \ldots, q_n), l, \bigwedge_{i \in \Sigma(l)} \phi_i, \bigwedge_{i \in \Sigma(l)} \psi_i, \bigvee_{i \in \Sigma(l)} u_i, \{ a_i \}_{i \in \Sigma(l)}, \bigcup_{i \in \Sigma(l)} Y_i, (q_1', \ldots, q_n')) \in E$ iff for all $i \in \Sigma(l)$, $(q_i, l_i, \phi_i, \psi_i, u_i, a_i, Y_i, q_i') \in E_i$, and for all $j \in \{1, \ldots, n\} \setminus \Sigma(l)$, $q_j' = q_j$,
- $\mathcal{I}(q_1, \ldots, q_n) = \bigwedge_{i=1}^{n} \mathcal{I}_i(q_i)$.

Model. Let $\mathcal{S} = (S, s^0, \Sigma, \longrightarrow)$ be the labeled transition system of the parallel composition of the automata $\mathcal{T}A_1, \ldots, \mathcal{T}A_n$. The *propositional variables* PV are of the form: $p_{i.q}$ for some $1 \leq i \leq n$, where $q \in Q_i$ and $p_{e_1 \sim e_2}$, where $e_1, e_2 \in Expr(V)$, and $\sim$ is a relational operator. In order to reason about systems represented as a set of automata we define a *labeling* function $\mathcal{V} : S \to 2^{PV}$. For $s = ((q_1, \ldots, q_n), v, \tau) \in S$, $\mathcal{V}(s)$ is defined as follows: $p_{e_1 \sim e_2} \in \mathcal{V}(s)$ iff $v \models e_1 \sim e_2$ and $p_{i.q} \in \mathcal{V}(s)$ iff $q_i = q$. The *model* for the set of automata $\{\mathcal{T}A_1, \ldots, \mathcal{T}A_n\}$ is the pair $M = (\mathcal{S}, \mathcal{V})$.

Let $PV_\varphi \subseteq PV$ be the set of propositional variables used in the formula φ. We call a location $q \in Q_i$ *observable* (for φ) if it appears in some proposition of PV_φ, i.e., $p_{i.q} \in PV_\varphi$.

[4]These variables are used by more than one process.

3. Motivating example

As the example we present a system that exploits the well known *Fischer's mutual exclusion protocol* [3] to ensure mutual exclusion. The system consists of n processes using the same shared resource. Each of the processes is modeled as a timed automaton shown in Fig. 1. The global variable Z is used to schedule an access to the resource. The shared resource is represented by another global variable Y. Besides competing for the resource the processes can perform some local actions. For the sake of clarity, we skip the labels of the transitions since all of them are local (i.e., non synchronous). The urgency attribute is not set for any transition.

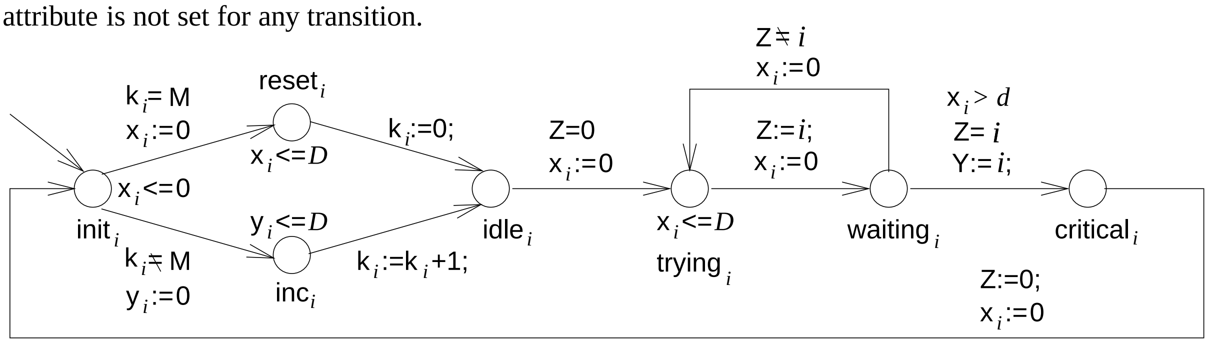

Figure 1. The i-th automaton of Fischer's mutual exclusion protocol

Before we show in detail how to reduce a system using the technique of path compression, we give some intuition and sketch our method. The method is property driven, i.e., it depends on a property to be verified. The property can be expressed by an additional timed automaton or by a temporal formula. For the system presented we verify the formula $\varphi = EF(\bigvee_{1 \leq i < j \leq n}(p_{i.critical_i} \wedge p_{j.critical_j}))$ saying that eventually the mutual exclusion property is violated. The important point to note here is that the validity of the formula in the model does not depend on values of the variables Y and k_i.

Sketch of the method. The first step is to use a slicing algorithm to obtain a set of the *relevant* variables, i.e., the variables that can influence a property in question. The slicing algorithm for timed automata with integer variables was defined in [16], so now we present its main ideas only. All the operations (i.e., the enabling conditions and the operations executed in the transitions) on irrelevant variables are removed. After completing the slicing, the path compression algorithm is executed.

Based on the notions of data, control and time dependence, the slicing algorithm is developed to construct the sets of the *relevant* operations, locations, and labels. Intuitively, the *relevant* elements are these that have an impact on a property of interest. The algorithm starts with a slicing criterion (a set of operations defining observable variables and a set of observable locations) which is derived from propositional variables of a formula in question and successively marks as relevant a new item if any other relevant item depends on it. The system reduced is composed exclusively of the relevant parts of the original one.

The slice of our example constructed according to the slicing algorithm, for the slicing criterion defined as the set of locations $critical_i$ for $1 \leq i \leq n$, is shown in Fig. 2. Comparing to the original one, the variables Y and k_i have been removed and so have been operations on them as they are irrelevant to the mutual exclusion property.

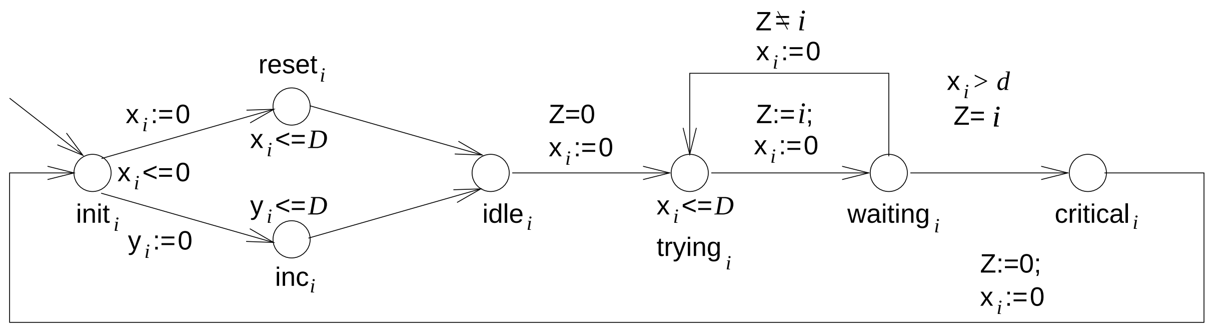

Figure 2. Automaton of Fig. 1 after slicing

Path compression. The idea behind the path compression consists in replacing a path (or a set of paths) with a single transition. Such a path which can be replaced, called *compressible*, must satisfy certain conditions. First of all, all the locations on the path are not observable. Secondly, the path cannot contain a cycle, which means that none of its locations occurs twice. Next, all the transitions of the path except for the last one have local labels, empty actions, and guards equal to $true$. Finally, all the clocks are reset immediately before they are used (in an invariant or in a transition enabling condition) and, dually, all the clock resets are immediately consumed, that is each clock which is reset at a transition going to some location q on the path is newly reset before it is used at any location different that q (in its invariant or in a enabling condition of one of its outgoing transitions).

If there are many compressible paths between a pair of locations, then all of them can be replaced by one transition provided that the last transitions of the paths are equal or at least: reset the same clocks, have the same enabling conditions, perform the same operations on variables, and either have the same labels, or all their labels are local.

In our example we have two compressible paths that fulfill all of the above conditions: $init_i \to reset_i \to idle_i \to trying_i$ and $init_i \to inc_i \to idle_i \to trying_i$. Both of them can be replaced by the transition $init_i \to trying_i$ as shown in Fig. 3.

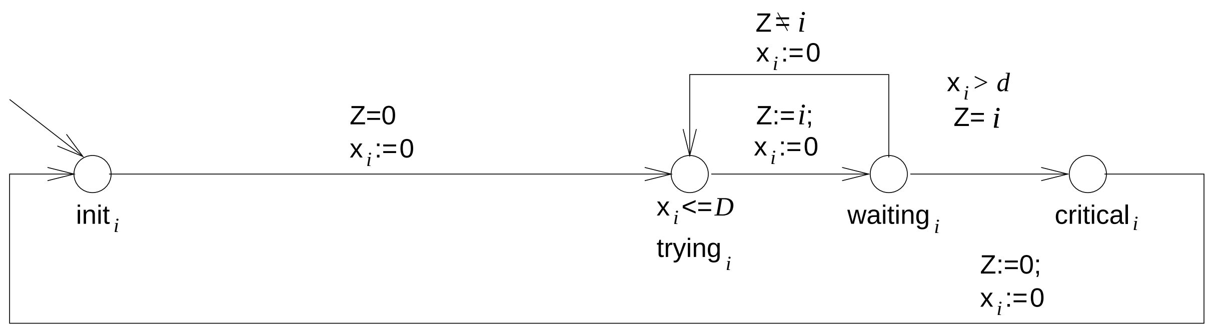

Figure 3. Automaton of Fig. 2 after path compression

It is easy to notice that the exact time of executing the transitions of the compressible paths is not important since the automaton can stay at the location $idle_i$ for an arbitrary amount of time. But, in general, this does not need to be the case. In our approach, if the time of a path traversal is finite, we replace the path with a transition of a finite execution time. In the rest of the paper we present our method in detail.

4. Path compression

In model checking applications we typically check whether a given temporal logic formula φ holds in a system $\mathcal{P}$. What we would like to do is to generate a reduced system $\mathcal{P}'$ such that φ holds in $\mathcal{P}'$ if and only if it holds in $\mathcal{P}$. It is clear that $\mathcal{P}'$ needs to preserve behavior of these parts of $\mathcal{P}$ only which can influence the validity of the formula φ.

As before let $\mathcal{TA}$ be a timed automaton, where Q and T are the sets of its locations and transitions, respectively. For $q \in Q$ let $out(q) \subseteq T$ be the set of its outgoing transitions and $in(q) \subseteq T$ be the set of its incoming transitions. A location $q \in Q$ is called an *ending location*, if $out(q) = \emptyset$.

Paths. A *path* in the automaton $\mathcal{TA}$ from a location $q' \in Q$ to a location $q'' \in Q$ is a sequence of locations and transitions of the form $q_1 t_2 q_2 \ldots t_m q_m$ such that $q_1 = q'$, $q_m = q''$, $m \geq 2$, $q_j \in Q$, and $t_j \in out(q_{j-1}) \cap in(q_j)$ for all $1 < j \leq m$. We say that a path $q_1 t_2 q_2 \ldots t_m q_m$ *contains* (or *goes through*) a location q if $q_j = q$ for some $1 \leq j \leq m$ (similarly for transitions). A path contains a *cycle* if it contains one of its locations twice, i.e., $q_i = q_j$ for some $1 \leq i < j \leq m$. We call a path *maximal* if it contains an ending location or a cycle. A path is *simple* if it does not contain a cycle. By $\Pi(q', q'')$ we denote a set of all simple paths from the location q' to the location q''. By $locs(\Pi(q', q'')) \subseteq Q$ and $trans(\Pi(q', q'')) \subseteq T$ we denote the set of all the locations and the set of all the transitions contained in the paths of $\Pi(q', q'')$, respectively.

Domination and post-domination. We say that a location $q \in Q$ *dominates* a location $q' \in Q$ if every simple path from the initial location q^0 to q' goes through q. Also, a location q' *post-dominates* a location q if every maximal path from q goes through q'. Notice that the above definitions are not symmetrical due to possibly missing an ending location. In the process in Fig. 1 the location $init_i$ dominates the location $reset_i$, but the location $reset_i$ does not dominate the location $idle_i$. Also the location $idle_i$ post-dominates the location $reset_i$, but the location $reset_i$ does not post-dominate the location $init_i$.

Reaching definitions for clocks. For a clock constraint $\psi \in \Psi(X)$, let $use(\psi) \subseteq X$ be the set of clocks that appear in ψ. The set of clocks *used* at a location $q \in Q$ is defined as follows: $use(q) = use(\mathcal{I}(q)) \cup \bigcup_{t \in out(q)} use(delay(t))$.

We say that $reset(t)$ of a transition $t \in T$ *affects* the clocks from the set $use(q)$ at $q \in Q$ if there exists a clock $x \in X$ such that $x \in reset(t) \cap use(q)$ and either $target(t) = q$, or there exists a path π from $target(t)$ to q such that $x \notin reset(t')$ for any transition t' contained in π. We say that the clock reset $reset(t)$ of a transition $t \in T$ is *locally consumed* if it does not affect $use(q)$ at any location $q \in Q \setminus \{target(t)\}$. Moreover, a clock $x \in use(q)$ is *locally used* at a location $q \in Q$ if $x \in reset(t)$ for all $t \in in(q)$. In the example in Fig. 1 the clock reset of each transition affects only the clock used at its target, this means that each clock reset is locally consumed. Also, clocks are locally used at each location.

Path delays. Let ψ be a time constraint of the form $d_1 \leq x \wedge x \leq d_2$,[5] where $d_1, d_2 \in \mathbb{N}$. We define the lower and upper bound of ψ (denoted as $lb(\psi)$ and $ub(\psi)$) as follows $lb(\psi) = d_1$ and $ub(\psi) = d_2$. We assume that $lb(\psi) = 0$ for ψ of the form $x \leq d_2$, and dually, $ub(\psi) = \infty$ for ψ of the form $d_1 \leq x$. Obviously, $lb(\psi) = 0$ and $ub(\psi) = \infty$ for ψ equal to $true$.

For a path $\pi = q_1 t_2 q_2 \dots t_m q_m$, where all the clocks used at q_j for all $1 \leq j < m$ are locally used and the time constraints of t_j for $1 < j \leq m$ include only one clock, we can define *minimal and maximal delay on the path* as follows:

$$min\_delay(\pi) = \sum_{j=2}^{m} lb(delay(t_j)),\ max\_delay(\pi) = \sum_{j=2}^{m} min(ub(delay(t_j)), ub(\mathcal{I}(q_{j-1})))^6.$$

In our example in Fig. 1 we have: $min\_delay(init_i \rightarrow reset_i \rightarrow idle_i) = 0$, and $max\_delay(init_i \rightarrow reset_i \rightarrow idle_i) = D$.

Definition 4.1. (Compressible paths). Let $q', q'' \in Q$ of $\mathcal{TA}$. We call a non-empty set of paths $\Pi(q', q'')$ *compressible* iff the following conditions are satisfied:

1. for each location $q \in locs(\Pi(q', q'')) \setminus \{q''\}$:
   (a) q is not observable and $q \neq q^0$ unless $q = q'$,
   (b) q' dominates q and q'' post-dominates q,
   (c) $source(t) \in locs(\Pi(q', q'')) \setminus \{q''\}$ for each transition $t \in in(q)$ with $q \neq q'$,
   (d) $|use(q)| \leq 1$ and if $use(q) \neq \emptyset$, then the clock of use(q) is locally used,
2. for each transition $t \in trans(\Pi(q', q'')) \setminus in(q'')$:
   (a) $label(t)$ is local (non-synchronous),
   (b) $guard(t)$ is equal to $true$,
   (c) $reset(t)$ uses only one clock and this clock is locally consumed,
   (d) $action(t)$ is equal to ϵ,
3. for each two transitions $t, t' \in in(q'')$:
   (a) $label(t) = label(t')$ or $label(t)$ and $label(t')$ are local,
   (b) $guard(t) = guard(t')$,
   (c) $reset(t) = reset(t')$,
   (d) $action(t) = action(t')$,
   (e) $urgent(t) = urgent(t') = false$,

[5]For sake of simplicity we only deal with not strict inequalities here.
[6]We define $d + \infty = \infty$ and $\infty + d = \infty$ for an arbitrary d.

4. for each $d \in \mathrm{I\!R}_+$ such that

$$min_{\pi \in \Pi(q',q'')}(min\_delay(\pi)) \leq d \leq max_{\pi \in \Pi(q',q'')}(max\_delay(\pi))^7$$

there exists a path $\pi \in \Pi(q', q'')$ such that $min\_delay(\pi) \leq d \leq max\_delay(\pi)$.

We call q' (q'') the *first* (*last*, resp.) location of Π. For two sets of compressible paths Π and Π', we say that $\Pi \subseteq_c \Pi'$ if $locs(\Pi) \subseteq locs(\Pi')$ and $trans(\Pi) \subseteq trans(\Pi')$. A set Π of compressible paths is *maximal* (*mcps*, for short) if $\Pi \not\subseteq_c \Pi'$ for any other set $\Pi' \neq \Pi$ of compressible paths.

Note that we do not require a transition $t \in trans(\Pi(q', q''))$, which does not enter the location q'', to have the urgency attribute equal to $false$. Since $guard(t)$ is equal to $true$, it follows that the transition t has to be executed as soon as the automaton enters the source location of t. Therefore, in such a case $urgent(t) = true$ is equivalent to $lb(delay(t)) = ub(delay(t) = 0$. The last condition of the above definition is necessary to keep a time constraint for a transition, which will replace a compressible path set, representable by a clock constraint of the form as defined in Section 2.

Lemma 4.1. For two different mcps $\Pi_1 = \Pi(q'_1, q''_1)$ and $\Pi_2 = \Pi(q'_2, q''_2)$ for some $q'_1, q''_1, q'_2, q''_2 \in Q$, $(locs(\Pi_1) \setminus \{q'_1, q''_1\}) \cap (locs(\Pi_2) \setminus \{q'_2, q''_2\}) = \emptyset$.

Proof:
Conversely, suppose that there exists $q \in (locs(\Pi_1) \setminus \{q'_1, q''_1\}) \cap (locs(\Pi_2) \setminus \{q'_2, q''_2\})$. Since by Def. 4.1.1(b) q'_1 and q'_2 dominate q, it follows that each path from q^0 to q passes through both q'_1 and q'_2. Without loss of generality we can assume that a path from q'_1 to q contains q'_2. Since Π_2 contains all the paths from q'_2 to q''_2 and by Def. 4.1.1(c) all the locations from which there are transitions to q are contained in Π_2, it follows that all the paths from q'_1 to q go through q'_2. Let us consider the path set $\Pi' = \Pi(q'_1, q''_2)$. Since all the paths from q'_1 to q'_2 are compressible and Π_2 is a mcps, it follows easily that the conditions of Def. 4.1 are satisfied for Π'. Therefore, Π' is a compressible path set containing Π_2, which contradicts maximality of Π_2. $\square$

Reduced Timed Automaton. We show how to construct a transition t that will replace a maximal compressible set of path $\Pi(q', q'')$ in the reduced timed automaton. Let $x \in X$ be a clock (denoted as $clock(\Pi(q', q'')))$ chosen in the following way. If there is a clock used at some location $q \in Q$, which is not included in any maximal compressible path set and it is reset on each path from q'' to q, then let x be such a clock. Otherwise, let x be an arbitrary clock used at some location $q \in locs(\Pi(q', q''))$. If there is no such clock, then no clock is needed for the transition t (then $clock(\Pi(q', q''))$ is undefined). Clearly, in case of many maximal compressible path sets we minimize the number of clocks by choosing the same clock for as many compressible path sets as possible.

Definition 4.2. (Reduced Timed Automaton). The reduced timed automaton is obtained from the original one, where each maximal compressible set of paths $\Pi(q', q'')$ is replaced by a fresh transition t. Let t' be an arbitrary transition entering q'' and $x = clock(\Pi(q', q''))$. We define t as follows:

- $source(t) = q'$,

[7] A maximal value exists since there is a finite number of finite paths.

- $target(t) = q''$,
- $label(t) = label(t')$,
- $guard(t) = guard(t')$,
- $delay(t) = min_{\pi \in \Pi(q',q'')} (min\_delay(\pi)) \leq x$[8] $\wedge\, x \leq max_{\pi \in \Pi(q',q'')} (max\_delay(\pi))$[9],
- $urgency(t) = false$,
- $action(t) = action(t')$,
- $reset(t) = reset(t')$.

Let $d = max_{\pi \in \Pi(q',q'')}(\sum_{q_j \in locs(\{\pi\}) \setminus \{q''\}} ub(\mathcal{I}(q_j)))$. If $d \neq \infty$, then the invariant $\mathcal{I}(q')$ is redefined to $x \leq d$, otherwise it is redefined to $true$. The reset set for each transition $t'' \in in(q')$ is redefined to $reset(t'') \cup \{x\}$.

By Def. 4.1.3(a), for the guard of the transition t we can choose the guard of an arbitrary t' entering q'' since the guards of all the transitions entering q'' are equal. Similarly we choose the label, the reset set, and the action of t (Def. 4.1.3(b)-(d)).

Complexity. In short, to find a mcps in a process we first mark the locations satisfying the condition 1(a) of Def. 4.1 and the transitions satisfying the condition 2 of Def. 4.1. Then, the rest of the conditions is checked for each pair of locations q' and q'', such that all the locations from $locs(\Pi(q',q'')) \setminus \{q''\}$ and all the transitions from $trans(\Pi(q',q'')) \setminus in(q'')$ are marked. The algorithm starts with the largest possible set of paths. If some condition of Def. 4.1 is not satisfied, then a path set for another pair of locations is checked.

The number of the pairs of locations (q', q'') is bounded by $|Q|^2$. The test of the condition 1 of Def. 4.1 is performed for each location of the path set in time $O(|Q| + |T|)$ (for 1 (b)) or in $O(1)$ (for 1 (a, c-d)). The next two conditions are checked in time $O(|T|)$. The test of the last condition requires $O(|Q| + |T|)$ time as it is based on the shortest path search algorithm for acyclic graphs [8]. Thus, the path reduction time is polynomial in the size of an automaton, that is in the number of its locations and transitions.

Correctness. We say that a location $q \in Q$ is *reducible* to a location q' if there exists a location $q'' \in Q$ such that $\Pi(q',q'')$ is a maximal compressible path set and $q \in locs(\Pi(q',q'')) \setminus \{q''\}$. For each $q \in Q$ we define $red(q)$ as follows: if there exists the location q' such that q is reducible to q', then $red(q) = q'$, otherwise $red(q) = q$. Note that a consequence of Lemma 4.1 is that for a location q there is at most one location q' to which q is reducible.

Let $\mathcal{TA}'$ be the reduced timed automaton obtained from $\mathcal{TA}$ according to Def. 4.2. Consider the clock renaming Γ such that for all $q \in Q$, where q is the first location of some mcps, replaces in both the automata the clock used at q by the same fresh clock in all the constraints appearing in the invariant of q and in the enabling conditions of the transitions going out of q as well as in the reset sets of the

[8]We skip the constraint if $min_{\pi \in \Pi(q',q'')} (min\_delay(\pi)) = 0$.
[9]We skip the constraint if $max_{\pi \in \Pi(q',q'')} (max\_delay(\pi)) = \infty$.

transitions entering q.[10] Denote by $\Gamma(\mathcal{T}A)$ and $\Gamma(\mathcal{T}A')$ the timed automata obtained by applying this renaming to $\mathcal{T}A$ and $\mathcal{T}A'$, resp. Since by Def. 4.1.1(d) the clock used at q is locally used and by Def. 4.1.2(c) it is locally consumed, it follows that the transition systems for $\Gamma(\mathcal{T}A)$ ($\Gamma(\mathcal{T}A')$) and $\mathcal{T}A$ ($\mathcal{T}A'$, resp.) are isomorphic.

Let $\mathcal{S} = (S, s^0, \Sigma, \longrightarrow)$ be the transition system for $\Gamma(\mathcal{T}A_1) \parallel \ldots \parallel \Gamma(\mathcal{T}A_n)$ and $\mathcal{S}' = (S', s^{0'}, \Sigma', \longrightarrow')$ be the transition system for $\Gamma(\mathcal{T}A'_1) \parallel \ldots \parallel \Gamma(\mathcal{T}A'_n)$ with respect to the set of propositional variables PV_φ. The labeling functions $\mathcal{V} : S \to 2^{PV_\varphi}$ and $\mathcal{V}' : S' \to 2^{PV_\varphi}$ are defined as in Section 2.

Let $M = (\mathcal{S}, \mathcal{V})$ and $M' = (\mathcal{S}', \mathcal{V}')$. Our aim is to show that M and M' are stuttering bisimilar according to the following definition.

Definition 4.3. (Stuttering bisimulation [7]). A relation $\cong_b \subseteq S \times S'$ is a stuttering simulation between M and M' if the following conditions hold:

1. $s^0 \cong_b s^{0'}$ and
2. if $s \cong_b s'$, then $\mathcal{V}(s) = \mathcal{V}'(s')$ and for every maximal path σ of M that starts at s, there is a maximal path σ' in M' that starts at s', a partition $B_1, B_2, \ldots$ of σ, and a partition $B'_1, B'_2, \ldots$ of σ' such that for each $j \geq 1$, B_j, and B'_j are nonempty and finite, and every state in B_j is related by $\cong_b$ to every state in B'_j.

A relation $\cong_b$ is a stuttering bisimulation if both $\cong_b$ and $\cong_b^T$ (the transpose of $\cong_b$) are stuttering simulations.

Definition 4.4. Let $\cong \subseteq S \times S'$ be the relation s.t. for each $s = ((q_1, \ldots, q_n), v, \tau) \in S$, and each $s' = ((q'_1, \ldots, q'_n), v', \tau') \in S'$ we have, $s \cong s'$ iff the following conditions hold:

1. $q'_i = red(q_i)$ for each $1 \leq i \leq n$,
2. $v' = v$, and
3. $\tau'(x) = \tau(x)$ for each $x \in use(q)$ for each $q \in \bigcup_{i=1}^{n} Q_i$ s.t. $red(q) = q$.

Lemma 4.2. The relation $\cong \subseteq S \times S'$ is a stuttering bisimulation between the two models $M = (\mathcal{S}, \mathcal{V})$ and $M' = (\mathcal{S}', \mathcal{V}')$.

The proof of Lemma 4.2 can be found the appendix. The following theorem is a consequence of Lemma 4.2.

Theorem 4.1. A CTL$_{\text{-X}}$* formula φ holds in the state s^0 of the model $M = (\mathcal{S}, \mathcal{V})$ iff it holds in the state $s^{0'}$ in the model of its reduct $M' = (\mathcal{S}', \mathcal{V}')$ with respect to the set of propositions PV_φ.

Proof:
By induction on the structure of φ. □

[10]If $use(q) = \emptyset$ in $\mathcal{T}A$, then the fresh clock is added to the reset sets of the transitions entering q in $\mathcal{T}A$.

	$n = 2$				$n = 4$				$n = 8$			
	bmc		minisat		bmc		minisat		bmc		minisat	
	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB
O	1.14	9.6	0.47	9.5	4.61	28.2	4.22	26.5	26.62	128.6	117.83	122.9
S	1.06	9.1	0.29	8.6	4.41	26.6	5.59	24.3	25.43	123.8	106.89	114.6
S+AC	0.73	6.9	0.13	7.3	2.99	18.0	6.75	6.7	16.74	77.5	67.58	72.1
S+PC	0.20	4.1	0.03	4.6	0.40	5.3	0.14	5.6	0.87	7.8	0.76	8.5

Figure 4. Experimental results for the example of Section 3 for $D = 2$ and $d = 1$

5. Experimental results

We present experimental results obtained with the verification tool VerICS [13], on the machine equipped with the processor Intel Pentium 4 – 3 GHz, 2 GB of main memory, and the Linux Red Hat operating system. We compare amounts of time and memory consumed by the translation of a network of timed automata and a given reachability property to a propositional formula (by the bounded model checking module [23] of VerICS) and satisfiability verification of the formula by MiniSat.

Let us first consider the example presented in Sect. 3. We have checked whether the formula φ defined in Sect. 3 holds for four systems: original (O, Fig. 1), sliced (S, Fig. 2), sliced and reduced according to the path compression technique (S+PC, Fig. 3), and sliced and reduced according to the active clock reduction (S+AC) [12]. The experiments have been performed for various numbers of processes (n) and values of parameters D and d. The preservation of mutual exclusion is ensured for $D < d$. The results are presented in Fig. 4.

To validate the technique we performed experiments with case studies known from the literature. In Fig. 5 the selected results are presented for *Manufacturing Plant* [11] and *Controller Area Network* [19]. For the *Manufacturing Plant* we check if the fall of any box is not possible (n is the number of boxes on the conveyor belt). In case of *Controller Area Network* we follow strictly the description of [19] and verify whether the request of the second application (with priority lower than the first one) is always accepted (n is the number of periodic application and m — sporadic ones). In all the cases presented, properties being verified are violated, that is erroneous scenarios' are found and the length of the shortest counterexample (depth) is shown.

6. Conclusions

In the paper, a method of abstraction exploiting static structures of timed automata has been presented. The experiments confirm that our method leads to significant reductions in the time and the memory consumed by the verification tool. The most important advantage of our approach is that it can be used prior to any existing tools analyzing timed automata including symbolic ones. It is also orthogonal to other abstraction methods and can be combined with them to yield a more powerful tool in terms of state space reduction.

				bmc		minisat	
example	parameters	version	depth	sec	MB	sec	MB
Manufacturing Plant	$n = 2$	**O**	20	3.88	20.6	2.20	18.9
		PC	16	2.81	16.2	0.84	14.4
	$n = 3$	**O**	22	5.85	28.5	6.92	24.4
		PC	18	4.00	21.2	2.31	19.8
	$n = 4$	**O**	24	7.83	35.5	71.96	50.5
		PC	20	5.27	27.5	14.22	27.6
Controller Area Network	$n = 2, m = 0$	**O**	40	23.70	90.8	44.67	85.7
		PC	36	17.65	71.2	21.60	61.2
	$n = 2, m = 1$	**O**	40	28.32	106.0	117.21	103.1
		PC	36	20.81	80.0	78.02	80.5
	$n = 3, m = 1$	**O**	40	33.20	121.3	566.89	189.5
		PC	36	25.49	96.5	274.14	127.0

Figure 5. Selected experimental results

References

[1] R Alur and D Dill. Automata for modelling real-time systems. In *Proc. of the Int. Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.

[2] G. Behrmann, P. Bouyer, E. Fleury, and K. Larsen. Static guard analysis in timed automata verification. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 254–277. Springer-Verlag, 2003.

[3] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems (SFM-RT'04)*, volume 3185 of *LNCS*, pages 200–236. Springer-Verlag, 2004.

[4] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *Proc. of the 9th Int. Conf. on Concurrency Theory (CONCUR'98)*, volume 1466 of *LNCS*, pages 485–500. Springer-Verlag, 1998.

[5] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lecture Notes on Concurrency and Petri Nets*, volume 3098 of *LNCS*. Springer-Verlag, 2004.

[6] V. Braberman, D. Garbervetsky, and A. Olivero. Improving the verification of timed systems using influence information. In *Proc. of Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 21–36. Springer-Verlag, 2002.

[7] M. C. Browne, E. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1/2):115–131, 1988.

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

[9] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. In *Proc. of the IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94)*. Elsevier, 1994.

[10] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Proc. of the 4th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *LNCS*, pages 313–329. Springer-Verlag, 1998.

[11] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with kronos. In *Proc. of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 66–75, 1995.

[12] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81. IEEE Computer Society, 1996.

[13] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Półrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying timed automata and Estelle specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 278–283. Springer-Verlag, 2003.

[14] T. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[15] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[16] A. Janowska and P. Janowski. Slicing timed automata with discrete data. *Fundamenta Informaticae*, 72(1-3):181–195, 2006.

[17] A. Janowska and W. Penczek. Static path compression in timed systems. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'06)*, volume 206(3) of *Informatik-Berichte*, pages 340–351. Humboldt University, 2006.

[18] R. Jhala and R. Majumdar. Path slicing. In *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation (PLDI'05)*, pages 38–47, 2005.

[19] J. Krákora and Z. Hanzálek. Timed automata approach for CAN verification. In *11th IFAC Symposium on Information Control Problems in Manufacturing*. Elsevier, 2004.

[20] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. of the 18th IEEE Real-Time System Symposium (RTSS'97)*, pages 14–24. IEEE Computer Society, 1997.

[21] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[22] P. Pettersson and K. G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.

[23] A. Półrola and A. Zbrzezny. Sat-based reachability checking for timed automata with discrete data. In *Proc. of the Int. Workshop on Concurency, Specification and Programming (CSP'06)*, volume 206(2) of *Informatik Berichte*, pages 207–218. Humboldt University, 2006.

[24] A. Valmari. The state explosion problem. In *Lecture Notes on Petri Nets I: Basic Models. Advances in Petri Nets*, volume 1491 of *LNCS*. Springer-Verlag, 1998.

[25] M. Weiser. Program slicing. *IEEE Trans. on Software Eng.*, 10(4), 1984.

[26] K. Yorav and O. Grumberg. Static analysis for state-space reductions preserving temporal logics. *Form. Methods Syst. Des.*, 25(1):67–96, 2004.

[27] H. Zheng, E. Mercer, and C. Myers. Modular verification of timed circuits using automatic abstraction. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(9):1138–1153, 2003.

7. Appendix

For sake of clarity we give a proof for a single time automaton first. Then we present the main result, that is a proof for a set of timed automata.

Let $\mathcal{S} = (S, s^0, \Sigma, \longrightarrow)$ be the labeled transition system for a timed automaton $\mathcal{TA} = (\Sigma, Q, q^0, V, X, T, \mathcal{I})$ and $\mathcal{S}' = (S', s^{0'}, \Sigma', \longrightarrow')$ be the labeled transition system for the reduced timed automaton $\mathcal{TA}' = (\Sigma', Q', q^{0'}, V', X', T', \mathcal{I}')$, which is constructed for $\mathcal{TA}$ according to Def. 4.2 with respect to the set of propositional variables PV_φ. The labeling functions $\mathcal{V} : S \to 2^{PV_\varphi}$ and $\mathcal{V}' : S' \to 2^{PV_\varphi}$ are defined as in Section 2.

Let us introduce a few notions, here. We say that a transition $t \in T$ and a transition $t' \in T'$ *correspond* to each other iff $guard(t') = guard(t)$, $action(t') = action(t)$, $label(t') = label(t)$ provided $|\Sigma(label(t))| > 1$, $delay(t') = delay(t)$, $urgency(t') = urgency(t)$, and $target(t') = target(t)$.

A transition $t \in T$ with a label $l \in \Sigma$ is *enabled* at a state $s = (q, v, \tau)$ (we write $enabled(t, s)$) if $source(t) = q$ and $v \models guard(t)$. A transition $t \in T$ is *fireable* at a state s (we write $fireable(t, s)$) if $enabled(t, s)$ and $\tau \models delay(t)$.

Proof of Lemma 4.2 for a single timed automaton

We show that the relation $\cong$ satisfies the conditions of Def. 4.3. Let $s^0 = (q^0, v^0, \tau^0)$ and $s'_l = (q^{0'}, v^{0'}, \tau^{0'})$. It is easy to check that $s^0 \cong s^{0'}$ since $q^{0'} = q^0$, $v^{0'} = v^0$, and $\tau^0(x) = 0$ for all $x \in X$ and $\tau^{0'}(x) = 0$ for all $x \in X'$. We proceed to prove that $\mathcal{V}(s) = \mathcal{V}'(s')$ for $s = (q, v, \tau) \in S$ and $s' = (q', v', \tau') \in S'$, where $s \cong s'$. By Def. 4.4.2 $v' = v$, hence $p_{e_1 \sim e_2} \in \mathcal{V}(s)$ iff $p_{e_1 \sim e_2} \in \mathcal{V}'(s')$ for $e_1, e_2 \in \Phi(V)$. It remains to show that $p_r \in \mathcal{V}(s)$ iff $p_r \in \mathcal{V}'(s')$ for $r \in Q$. By Def.4.4.1 $q' = red(q)$. There are two cases: either $red(q) = q$, or $red(q) \neq q$. If $red(q) = q$, then $q = r$ implies $q' = r$ and $q \neq r$ implies $q' \neq r$. Otherwise, by Def. 4.1.1(a) neither q nor $red(q)$ is observable. Clearly r is observable, so $q \neq r$ and $q' = red(q) \neq r$. By the above we see that $q = r$ iff $q' = r$, which completes the proof of $\mathcal{V}(s) = \mathcal{V}'(s')$.

($\Rightarrow$) Let σ be a maximal path of M starting at s. We proceed to show how to construct a maximal path σ' starting at s', a partition $B_1, B_2, \ldots$ of σ and a partition $B'_1, B'_2, \ldots$ of σ' as required in Item 2 of Def. 4.3. The construction is inductive. We assume that we have already constructed a finite prefix of σ: $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} s_k$ and a finite prefix of σ': $s'_1 \xrightarrow{a'_1} s'_2 \xrightarrow{a'_2} \cdots \xrightarrow{a'_{l-1}} s'_l$, where $a_i \in \Sigma \cup \mathbb{R}_+$, $s_1 = s$, $s'_1 = s'$, and $s_k \cong s'_l$[11] and that we have built partitions of σ and σ' into corresponding stuttering blocks. Let us assume that s_k belongs to the block B_m and s'_l belongs to the corresponding block B'_m. Let $s_k \xrightarrow{a_k} s_{k+1}$. There are two cases. Either s_{k+1} belongs to the new block B_{m+1}, or s_{k+1} belongs to the same block as s_k, namely B_m (see Fig. 6).

The main idea of the construction is as follows. The state s_{k+1} belongs to the same block as s_k, iff the transition labeled with a_k is contained in some mpcs and does not enter its last location. In all the other cases (other action transitions and time transitions) s_{k+1} belongs to the new block. The proof falls into two parts.

[11]Let $s_k = (q_k, v_k, \tau_k)$ and $s'_l = (q'_l, v'_l, \tau'_l)$ for arbitrary $k, l \in \mathbb{N}$.

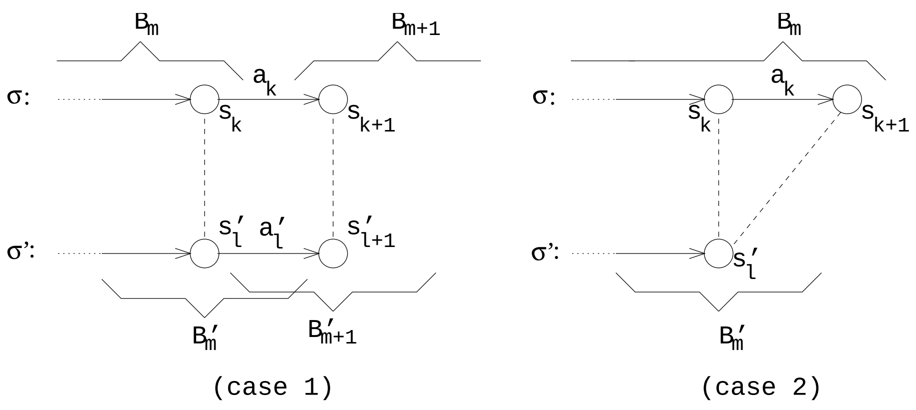

Figure 6. $a_k \in \Sigma \cup \mathbb{R}_+$

case 1 s_{k+1} belongs to the new block B_{m+1}. We show how to select s'_{l+1} such that $s'_l \xrightarrow{a'_l} s'_{l+1}$ and $s_{k+1} \cong s'_{l+1}$ (Fig.6(case 1)). The state s'_{l+1} will belong to a new block B'_{m+1}. Let us again consider two cases:

1. $a_k \in \Sigma$. Then, there are two cases (note that the situation where $red(q_{k+1}) \neq q_{k+1}$ belongs to **case 2**):

   (a) $red(q_k) = q_k$ and $red(q_{k+1}) = q_{k+1}$. Let t be a transition such that $t \in T$ and is labeled with a_k. In this case t is not included in any mcps. By construction of $\mathcal{TA}'$ (Def. 4.2) there exists a transition $t' \in T$ corresponding to t. We show that t' is fireable at s'_l.

   Since t is fireable at s_k, it follows that $source(t) = q_k$, $v_k \models guard(t)$, and $\tau_k \models delay(t)$. By Def. 4.4.1 we have $q_k = q'_l$. This gives $source(t') = source(t) = q_k = q'_l$. Next, $guard(t') = guard(t)$ and by Def. 4.4.2 $v_k = v'_l$, which gives $v'_l \models guard(t')$. Besides, $delay(t') = delay(t)$ and by Def.4.4.3 $\tau_l'(x) = \tau_k(x)$ for all $x \in use(q_k)$. Hence $\tau_l' \models delay(t')$. Therefore t' is fireable at s'_l.

   (b) $red(q_k) \neq q_k$ and $red(q_{k+1}) = q_{k+1}$. In this case by Def. 4.1.1.(b) the transition $t \in T$ labeled with a_k is a transition entering the last location of a path π of some mcps Π, where $red(q_k)$ is the first location of the path. We show that a transition $t' \in T$ constructed according to Def. 4.2 for a mcps Π is fireable at s'_l.

   First, by Def. 4.4.1 we have $q'_l = red(q_k)$ and by construction $source(t') = red(q_k)$, which gives $source(t') = q'_l$. Next, by Def. 4.4.2 $v_k = v'_l$ and by construction $guard(t') = guard(t)$ (by Def. 4.1.3(b) all the guards of transitions entering the last location of a mcps are equal), which gives $v'_l \models guard(t')$. Finally, we show that $\tau_l' \models delay(t')$, where $delay(t')$ is defined according to Def. 4.2. By Def. 4.4.3 $\tau_l'(x) = \tau_k(x)$, where $use(red(q_k)) = use(q'_l) = \{x\}$. The clock x is not reset since the automaton $\mathcal{TA}$ has visited the location $red(q_k)$. Therefore $\tau_k(x)$ reflects time of traversing the path π. From this it follows that $min\_delay(\pi) \leq \tau_k(x)$ and $\tau_k(x) \leq max\_delay(\pi)$. On the other hand by Def. 4.2 we know that $lb(delay(t')) \leq min\_delay(\pi)$ and $max\_delay(\pi) \leq ub(delay(t'))$. It gives $lb(delay(t')) \leq \tau_l'(x)$ and $\tau'_l(x) \leq ub(delay(t'))$, which shows that $\tau_l' \models delay(t')$.

What we have shown so far is how to choose a transition t' in the system $\mathcal{S}'$ such that $s'_l \xrightarrow{a_l} s'_{l+1}$, where a_l is the label of t'. We need to prove that $s_{k+1} \cong s'_{l+1}$.

It is easy to check that in both the above cases $q_{k+1} = target(t) = target(t') = q'_{l+1}$ as t corresponds to t'. This proves that Item 1 of the Def. 4.4 holds for s_{k+1} and s'_{l+1}. Next, in both the cases by Def. 4.1.3(d) $action(t) = action(t')$. Since $v_k = v'_l$, it follows that after the execution of the same sequence of operations $v_{k+1} = v'_{l+1}$, which satisfies Item 2. Finally, in both the cases Def. 4.1.3(c) $reset(t) = reset(t')$. Since Item 3 is satisfied for $\tau_l{}'$ and τ_k, after resetting the same set of clocks it remains satisfied for τ'_{l+1} and τ_{k+1}.

Combining these gives $s_{k+1} \cong s'_{l+1}$.

2. $a_k \in \mathbb{R}_+$, which represents time progress.

   In this case $q_{k+1} = q_k$, $v_{k+1} = v_k$, and $\tau_{k+1} = \tau_k + a_k$. Obviously, $q_{l+1}{}' = q_l{}'$ and $v_{l+1}{}' = v_l{}'$. Furthermore $\tau_{l+1}{}' = \tau_l{}' + a_k = \tau_k + a_k = \tau_{k+1}$. This proves $s_{k+1} \cong s'_{l+1}$, where $s'_l \xrightarrow{a_k} s'_{l+1}$. We need only to show that the progress of time of a_k is possible at the state s'_l. Let us consider three cases:

   (a) q_k is not included in any mcps or it is the last location of some mcps. In this case $red(q_k) = q_k$ and by Def. 4.4.1 $q'_l = red(q_k)$. Also, each transition $t \in T$ going out of q_k has a corresponding transition $t' \in T'$ going out of q'_l. Moreover, by construction of $\mathcal{TA}'$ we have $\mathcal{I}(q_k) = \mathcal{I}'(q'_l)$. Since a timed transition of duration a_k is possible at s_k, $\tau_k + a_k \models \mathcal{I}(q_k)$ and $urgent(t) = false$ or $v_k \not\models guard(t)$ for all $t \in out(q_k)$. Finally, by Def. 4.4.3 $\tau_k = \tau_l{}'$. Combining these gives $\tau'_l + a_k \models \mathcal{I}'(q'_l)$ and $urgent(t') = false$ or $v'_l \not\models guard(t')$ for all $t' \in out(q'_l)$, which proves our claim for this case.

   (b) q_k is a first location of a path π of some mcps Π. In this case also $red(q_k) = q_k$ and by Def. 4.4.1 $q'_l = red(q_k)$. By Def. 4.2 $use(q_k) = use(q'_l) = \{x\}$ and $ub(\mathcal{I}(q_k)) \leq ub(\mathcal{I}'(q'_l))$. Next, $\tau_k(x) + a_k \models \mathcal{I}(q_k)$. Finally, by Def. 4.4.3 $\tau_k(x) = \tau_l{}'(x)$. Combining these gives $\tau'_l(x) + a_k \models \mathcal{I}'(q'_l)$. By construction $urgent(t') = false$ for each transition $t' \in T'$ going out of q'_l which completes a proof for this case.

   (c) q_k is included in some mcps and is not its first location. According to Def. 4.2 $use(q'_l) = use(red(q_k)) = \{x\}$. As before $\tau_k(x)$ reflects time of traversing the path π. It follows that $\tau_k(x) + a_k \leq max\_delay(\pi)$. But from definition of max_delay, $max\_delay(\pi) \leq \sum_{q \in locs(\{\pi\}) \setminus \{q_m\}} ub(\mathcal{I}(q))$, where q_m is the last location of π. Then, by Def. 4.2 $\sum_{q \in locs(\{\pi\}) \setminus \{q_m\}} ub(\mathcal{I}(q))) \leq ub(\mathcal{I}'(q'_l))$. Finally, by Def. 4.4.3 $\tau_k(x) = \tau_l{}'(x)$. Combining these gives $\tau'_l + a_k \models \mathcal{I}'(q'_l)$. By construction $urgent(t') = false$ for each transition $t' \in T'$ going out of q'_l which completes a proof for this case.

   This completes our claim that from the state s'_l the timed transition of duration a_k can be performed.

case 2 s_{k+1} belongs to the same block as s_k, namely B_m. We show that $s_{k+1} \cong s'_l$. As we said before, this situation takes place when $a_k \in \Sigma$ and $red(q_{k+1}) \neq q_{k+1}$. In this case by Def. 4.1.1(b-c) we know that the transition $t \in T$ labeled with a_k is a transition of a path π of some mcps Π, which does not enter the last location of π. By Def. 4.4.1 $q'_l = red(q_k)$. It is clear that q_k and q_{k+1} are contained in the same path, thus $q'_l = red(q_k) = red(q_{k+1})$, which satisfied Item 1 of Def. 4.4 for s_{k+1} and s'_l. We see at once that Item 2 Def. 4.4 is also satisfied, since by Def. 4.1.2(d) $v_{k+1} = v_k$. Finally, by the fact that Item 3 is satisfied for $\tau_l{}'$ and τ_k it follows that Item 3 remains satisfied for τ_l' and τ_{k+1}, because by Def. 4.1.2(c) clocks, which are reset at t, are not included in the set $\bigcup_{\{q \in \bigcup Q_i \,|\, red(q)=q\}} use(q)$. By the above we have $s_{k+1} \cong s_l'$.

($\Leftarrow$) Let σ' be a maximal path of M' starting at s'. We proceed to show how to construct a maximal path σ starting at s, a partition $B_1, B_2, \ldots$ of σ and a partition $B_1', B_2', \ldots$ of σ' as required in Item 2 of Def. 4.3. As before, we assume that we have already constructed a finite prefix of σ: $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} s_k$ and a finite prefix of σ': $s_1' \xrightarrow{a_1'} s_2' \xrightarrow{a_2'} \cdots \xrightarrow{a_{l-1}'} s_l'$, where $s_1 = s$, $s_1' = s'$ and $s_k \cong s_l'$ and that we have built partitions of σ and σ' into corresponding stuttering blocks. Let us assume that s_k belongs to the block B_m and s_l' belongs to the corresponding block B_m'. The proof falls into two parts.

case 1 There is an action transition on the suffix of the path σ' starting at state s_l'. By the additivity of time all the timed transitions before the first action transition on the suffix of σ' are equivalent to one timed transition $s_l' \xrightarrow{\delta_l'} s_{l+1}'$. If there is no timed transition before the first action transition, then $\delta_l' = 0$.

We show that if $s_k \cong s_l'$ and $s_l' \xrightarrow{\delta_l'} s_{l+1}' \xrightarrow{\gamma_{l+1}'} s_{l+2}'$ in M', where $\delta_l' \in \mathbb{R}_+$ and $\gamma_{l+1}' \in \Sigma$, then there exists a path in M satisfying the conditions of Item 2 of Def. 4.3. Let us consider the following cases:

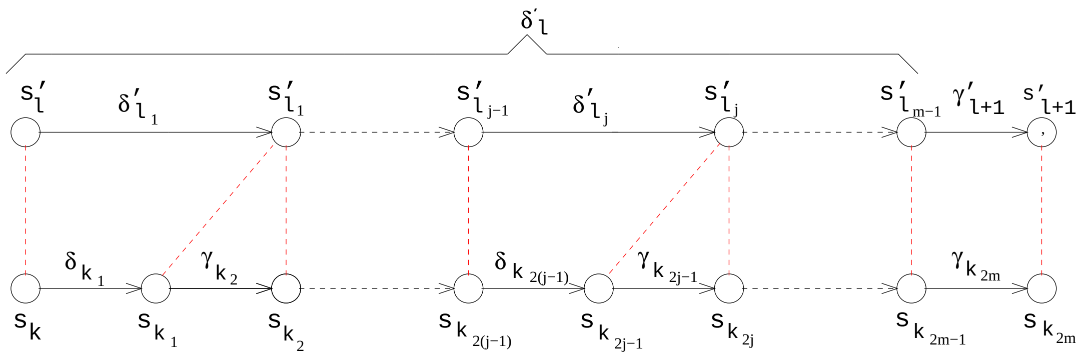

Figure 7. $\delta_{l_i}', \delta_{k_i} \in \mathbb{R}_+$, $\gamma_{l_i}', \gamma_{k_i} \in \Sigma$

1. q_k is not included in any mcps or it is the last location of some mcps. In this case for $s_l' \xrightarrow{\delta_l'} s_{l+1}'$ we construct $s_k \xrightarrow{\delta_k} s_{k+1}$ such that $s_{l+1}' \cong s_{k+1}$ in the same manner as in **case 1.2(a)** of the previous part of the proof. Then, for $s_{l+1}' \xrightarrow{\gamma_{l+1}'} s_{l+2}'$ we construct $s_{k+1} \xrightarrow{\gamma_{k+1}} s_{k+2}$ in the same way as described in **case 1.1(a)**.

2. q_k is a first location of a path of some mcps Π. By Def. 4.1.4 we can choose a path $\pi \in \Pi$, such that $min\_delay(\pi) \leq \delta_l \leq max\_delay(\pi)$.

   Let $\pi = q_1 t_2 q_2 \ldots t_m q_m$, where $q_1 = q_k$.

By the additivity of time the timed transition $s'_l \xrightarrow{\delta'_l} s'_{l+1}$ is equivalent to the sequence of timed transitions $s'_l \xrightarrow{\delta'_{l_1}} s'_{l_1} \xrightarrow{\delta'_{l_2}} s'_{l_2} \dots \xrightarrow{\delta'_{l_m}} s'_{l_m} = s'_{l+1}$, such that $\delta'_l = \sum_{j=1}^{m} \delta'_{l_j}$ (see Fig. 7). We choose δ'_{l_j} such that

$$lb(delay(t_j)) \leq \delta'_{l_j} \leq min(ub(delay(t_j)), ub(\mathcal{I}(source(t_j))))$$

This is possible since

$$\sum_{j=1}^{m} lb(delay(t_j)) \leq \delta'_{l_j} \leq \sum_{j=1}^{m} min(ub(delay(t_j)), ub(\mathcal{I}(source(t_j))))$$

We may now construct the path $s_k \xrightarrow{\delta_{k_1}} s_{k_1} \xrightarrow{\delta_{k_2}} \dots \xrightarrow{\delta_{k_{2m}}} s_{k_{2m}}$, such that $\delta_{k_{2j-1}} = \delta'_{l_j}$ and $\delta_{k_{2j}} = label(t_j)$ for $j = 1, \dots, m$.

Each timed transition $\delta_{k_{2j-1}}$ is possible at $s_{k_{2(j-1)}}$ for $j = 1, \dots, m-1$, since $\tau(x) = \delta_{k_{2j-1}} \leq ub(\mathcal{I}(q_{k_{2(j-1)}}))$, where $use(q_{k_{2(j-1)}}) = \{x\}$. From this $\tau_{k_{2(j-1)}}(x) \models delay(\mathcal{I}(q_{k_{2(j-1)}}))$.

It is straightforward that t_j is enabled at $s_{k_{2j-1}}$ for $j = 1, \dots, m-1$, since $source(t_j) = q_{k_{2j-1}}$ and by Def. 4.1.2(b) $guard(t_j) = true$. Also, from Def. 4.1.1(d) we have $\tau_{k_{2j-1}}(x) = \delta_{k_{2j-1}}$ where $use(q_{k_{2j-1}}) = \{x\}$. By the above, $lb(delay(t_j)) \leq \delta_{k_{2j-1}} \leq ub(delay(t_j))$. Therefore $\tau_{k_{2j-1}}(x) \models delay(t_j)$, that is t_j is fireable at $s_{k_{2j-1}}$.

The proof of $s_{k_{2j}} \cong s'_{l_j}$ for all $1 \leq j < m$ can be handled in much the same way as in **case 1.2(c)** of the previous part. The proof of $s_{k_{2j-1}} \cong s'_{l_j}$ for all $1 \leq j < m$ follows the same line of arguments as in **case 2**.

The proof of $s_{k_{2m}} \cong s'_{l_m} = s'_{l+1}$ can be handled similarly to **case 1.1(b)**.

3. (all other cases are sub-cases of the previous two).

case 2 There is no action transition on the suffix of the path σ' starting at state s'_l. Let $s'_l \xrightarrow{\delta'_l} s'_{l+1}$, where $\delta'_l \in \mathbb{R}_+$, be the next transition on σ'. Let us consider two cases.

1. If q_k is not included in any mcps or it is the last location of some mcps, then we construct $s_k \xrightarrow{\delta_k} s_{k+1}$ such that $s'_{l+1} \cong s_{k+1}$ in the same manner as in **case 1.2(a)** of the previous part of the proof.

2. Otherwise, q_k is included in some mcps. We may construct a sequence of transitions $\delta_{k_1}, \gamma_{k_2}, \dots, \delta_{k_{2m-1}}$ corresponding to the transition δ'_l in the similar fashion as describe in **case 1**.

According to the construction described above each of the blocks $B_1, B_2, \dots, B'_1, B'_2, \dots$ is finite, because there is a finite number of locations. Thus, the conditions of Def. 4.3 are satisfied and the proof is complete.

Proof of Lemma 4.2 for a set of timed automata (the skeleton)

The proof can be handled in much the same way as the previous one with the only differences that some transitions are executed synchronously by more than one automaton.

$(\Rightarrow)$ By Def 4.1.2(a) a synchronous transition is is not included in any mcps. In this case the proof follows by the same method as in **case 1.1(a)** for each automaton performing this transition. Obviously, for a local transition the proof is the same as before. Similarly, for a timed transition the same line of arguments is applied as in **case 1.2** for all automata in the set.

$(\Leftarrow)$ Since a path $s'_l \xrightarrow{\delta'_l} s'_{l+1} \xrightarrow{\gamma'_{l+1}} s'_{l+2}$ (a part of σ') can be interleaved with transition of other automata, we consider a possibly longer path $s'_l \xrightarrow{\delta'_l} s'_{l+1} \xrightarrow{\gamma'_{l+1}} \dots \xrightarrow{\gamma'_{l+p-1}} s'_{l+p}$ such that γ'_{l+p-1} is the label of a transition going from the location $q_l^{i'}$ to $q_{l+k}^{i\;\;'}$, where i is the number of an automaton performing the transition. The construction follows the previous one. Each subsequence (for other automata) is built in the same fashion.