Agata Janowska Paweł Janowski

Slicing of Timed Automata with Discrete Data*

Nr 990

Warszawa, February 2006

*The authors acknowledge support from the Polish grant No. 3T11C01128.

Abstract

The paper proposes how to use static analysis to extract an abstract model of a system. The method uses techniques of program slicing to examine syntax of a system modeled as a set of timed automata with discrete data, a common input formalism of model checkers dealing with time. The method is property driven. The abstraction is exact with respect to all properties expressed in the temporal logic CTL_{-X}*.

Keywords: timed systems, timed automata, static analysis, program slicing

Streszczenie

Plastrowanie automatów czasowych z danymi dyskretnymi.
Praca przedstawia zastosowanie analizy statycznej do uzyskania abstrakcyjnego modelu systemu. Proponowana metoda jest oparta na technice plastrowania programów. Analizie podlega system przedstawiony jako zbiór automatów czasowych rozszerzonych o dane dyskretne, który to formalizm jest stosowany przez weryfikatory modelowe dla systemów czasowych. W przedstawionej metodzie abstrakcja systemu zachowuje prawdziwość formuł logiki temporalnej CTL_{-X}* i jest uzależniona od weryfikowanej własności.

Słowa kluczowe: systemy czasowe, automaty czasowe, analiza statyczna, plastrowanie programów

1 Introduction

In recent years many verification methods of time-dependent, concurrent systems have been presented. One of the most important due to its practical applicability seems to be model checking. Essentially, the method determines whether a temporal formula stating a property of a system is valid on a model representing its all possible executions. This technique has the advantage of being automatic but suffers from the *state explosion problem, i.e* the number of states in the whole state space of a system can be enormous. During the last two decades many methods have been suggested to reduce the state space that need to be considered in the verification process [24]. One of the solutions to cope with the problem is to construct an *abstract* model of the system [10]. Model checking using abstractions does not explore the concrete state space, but an abstract one, which is potentially smaller.

To extract an abstract model of a system we propose to use static analysis, namely a technique called *program slicing* [25]. The method (see [22] for a survey) consists in pointing the so-called *slicing criterion*, which identifies program points of interest and then tracing an items of code on which it depends. We present a slicing algorithm for a system composed of a set of timed automata with discrete data. The method is property driven since a slicing criterion is derived from atomic propositions of a formula in question. An abstractions is *exact* with respect to all properties expressed in the temporal logic $\mathrm{CTL}_{-\mathrm{X}}{}^*$ [9], which means that the abstract system satisfies a given property if and only if the concrete one does. We show that a certain relation between the original and the sliced product of timed automata with discrete data is a visible bisimulation [12]. The correctness of the method follows from [12] stating that two structures are equivalent w.r.t. $\mathrm{CTL}_{-\mathrm{X}}{}^*$ formulas if there exists a visible bisimulation between the states of those structures.

Related works. *Time automata* [2] are finite state Büchi automata extended with real-valued variables modeling clocks. *Time safely automata* [15] are time automata specifying progress properties using local invariants instead of Büchi accepting conditions. *Timed automata with discrete data* are timed safety automata enriched by additional integer variables (not clocks), that can be updated in transitions and referenced in enabling conditions. The formalism is an input of model checkers dealing with time such as UppAal [21] or recent version of Verics [20]. The definition of timed automata with discrete data used in UppAal is slightly different than ours. We point out the details in the paper.

Program slicing as an abstraction method has been successfully applied in the context of model checking of untimed systems. Millett and Teitelbaum [19] study slicing of Promela, the input language of SPIN model checker [16]. They obtain the so called *imprecise slice* and they do not formalize their slicing methods. Hatcliff et al. [14] present a formal study of slicing sequential programs preserving LTL and extend their techniques to multi-threaded Java programs [13]. Slicing is also present in the IF project [7] concerning timed systems, however, it is defined for its untimed subset only [6].

The closest related work on using static analysis in timed system verification concerns the concept of *influence information* [8]. The technique can be understood as slicing I/O Timed Components, timed safely automata extended with interfaces. The approach does not use the notion of a slicing criterion, instead it preserves the branching structure of a transition system up to the propositional assignment given over the external observer. Another related methods are the *active-clock reduction* technique [11] and more general *relevant guard abstraction* [4] for timed safety automata. Since they focus on clocks reduction they are orthogonal to ours and can be combined with it.

Our first attempt of using slicing in context of timed systems concerns reduction of intermediate language of Verics [17]. The formalism is a specification language with no explicit clock variables, but restricting the time of transitions executions by means of delays.

Structure of the paper. In Section 2 we present an example of a time dependent system and give an informal introduction to our method. Section 3 contains the definition of the syntax and operational semantics of timed automata with discrete data. Section 4 provides a detailed description of the slicing technique and its application to a system composed of a set of automata[1]. The last section shortly concludes the paper. The proof of the correctness of our approach can be found in the appendix.

2 Example

As the example we present a version of the well known *alternating bit protocol* ([3]) that provides reliable communication over a network service that sometimes looses messages. This simple protocol uses a one-bit sequence number (which alternates between 0 and 1) in each message and an acknowledgment to determine whether the message must be retransmitted. The system consists of three automata running in parallel, Sender, Receiver and Faulty Buffer, as shown in Fig. 1.

The task of Sender is to transmit portions of data, which represents some computations performed in real systems. In our example they are succeeding numbers modulo N. Sender sends each portion accompanied with the bit to Faulty Buffer. Then it waits for an acknowledgment. If the value of the acknowledgment is the same as the value of the bit, then the message is treated as delivered and the value of the bit flips. Otherwise the message is retransmitted. The message is also retransmitted, if no acknowledgment comes within T units of time (the timeout is modeled by the clock x). Receiver waits until it gets the message from Faulty Buffer, then it acknowledges receipt of the message and compares its sequence number with the bit value. If they are equal, it changes the value of the bit and accepts the message. Otherwise it waits for another message. Faulty Buffer accepts a message from Sender or an acknowledgment

[1] In the rest of the paper by automata we always mean time automata with discrete data.

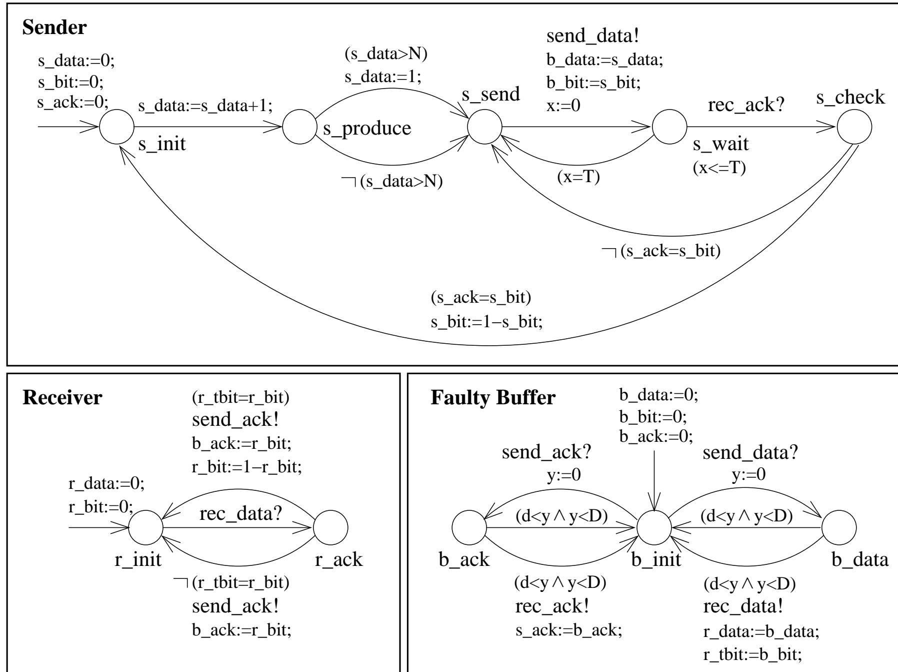


Figure 1: Alternating Bit Protocol

from Receiver and forwards it respectively or looses it. The clock y is used to model transmission delays, which are between d and D units of time.

Before we show in details how to reduce the protocol using techniques of program slicing, we would like to introduce some intuitions of key notions and sketch our method.

Synchronization. Transitions with a common label are executed *synchronously* and an order of variables updating is determined by additional marks ! (first) and ? (next). For example, Sender's transition with label *send_data*! is performed together with Faulty Buffer's transition with label *send_data*?.

Property. For the alternating bit protocol we want to verify the formula $\varphi = AG(Sender.s_init \Rightarrow (s_bit = r_bit))$ stating that when Sender is in its location *s_init* (at the beginning and always when it accepts an acknowledgment), values of Sender's bit and Receiver's bit are equal. The important point to note here is that the truthfulness of the formula does not depend on values of variables *s_data*, *b_data* and *r_data*.

Sketch of the method. Computing the slice of a system requires a few steps. First, given properties to be verified, a slicing criterion is extracted. Next, the slicing algorithm traces dependencies between items transitively, starting with

the slicing criterion. Finally, the slice of the system is constructed with relevant fragments of the system structure.

Slicing criterion. A slicing criterion is composed of two sets. The first one contains all operations defining observable variables and the second one contains the observable locations and their immediate predecessors (to ensure that an observable location would not be identified with non-observable one). For the alternating bit protocol and the formula φ a slicing criterion is the pair of sets $A^0 = \{s_bit = 1 - s_bit;, r_bit = 1 - r_bit;\}$ and $R^0 = \{s_init, s_check\}$.

Dependencies. In the slicing literature there are two basic notions of dependence: *data dependence* and *control dependence*. Intuitively, an item of a program data depends on another item, if the latter one can influence the value of some variable used at the first one. For example Sender's operation $(s_ack = s_bit)$ is data dependent on Faulty Buffer's operation $s_ack := b_ack;$. For control dependence, an item control depends on a conditional, if the conditional may affect execution of the item. We say that a location q is control dependent on a location q', if the reachability of the location q depends on a decision made in the location q', namely it depends on enabling conditions of transitions going out of location q'. For example Sender's location s_init is control dependent on its location s_check and on operations $(s_ack = s_bit)$ and $\neg(s_ack = s_bit)$, because there is a cycle from the location s_check in which the location s_init is not contained.

For timed automata with discrete data we consider two additional kinds of dependencies: *clock* and *time dependencies*. Clock dependence is an analogue of data dependence for clocks instead of data variables. The idea of time dependence is as follows: a location q is time dependent on another location q' from which it is reachable, if in the location q' time can elapse. For example all Sender's locations are time dependent on its locations s_wait and s_send and there is no more time dependencies for Sender.

Slicing algorithm. Based on the dependence notions the slicing algorithm is developed to construct the sets of *relevant* operations, locations and labels. Intuitively, *relevant* elements are those that have impact on a property of interest. The algorithm starts with a slicing criterion and successively marks as relevant new items, if any other relevant items depend on them. The reduced system is composed exclusively of relevant parts of the original one. The slice of the alternating bit protocol constructed according to the slicing algorithm is shown in Fig. 2. Comparing to the original, the location $s_produce$ disappears in the reduced protocol, because it appears to be non-relevant as no location depends on it. There are no variables s_data, r_data and b_data as they do not occur in any of relevant operations.

Experimental results. Verifying that the protocol satisfies the formula φ can be done by one of the model checkers mentioned earlier. To this end constants d, D, T and N must be instantiated with concrete values. We compare some aspects of the original and the sliced protocol and focus on discrete parts of states of the transition systems being verified (that is the set of current locations

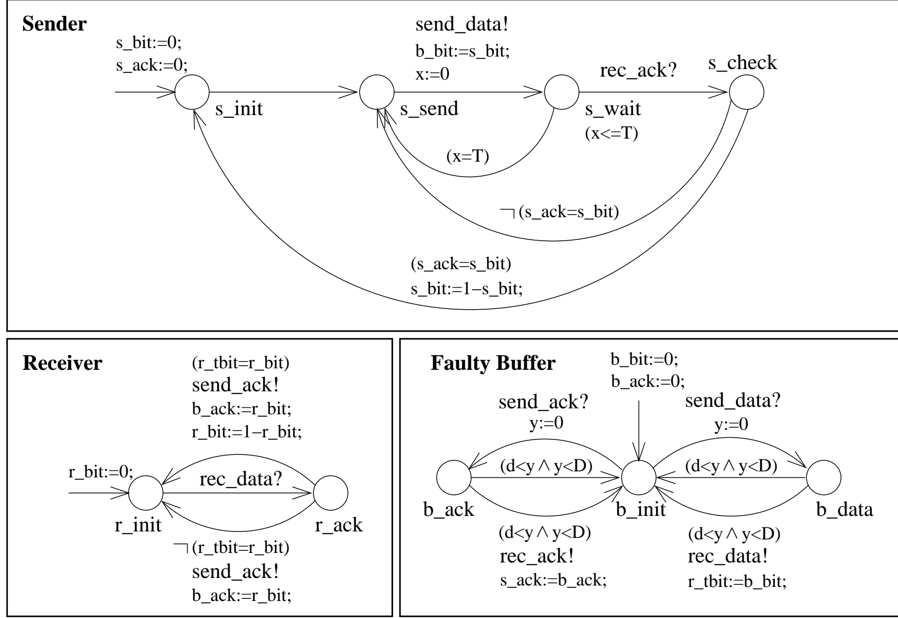


Figure 2: Sliced Alternating Bit Protocol

of the automata and a variables valuation) as our reduction method concerns mainly these parts. For example, in case of the original protocol we obtain 97 various discrete configurations for $N = 1$, but 361 configurations for $N = 10$ and 3331 configurations for $N = 100$. The sliced protocol, which is insensitive to the value N, in all cases has 93 configurations.

3 Timed automata with discrete data

3.1 Syntax

Variables. Let V be a finite set of integer variables. The set of *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]. By $Expr(V)$ we denote the set of all arithmetic expressions. The set of *boolean expressions* over V is defined inductively as follows:

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

[2]By / we denote integer division and by % – modulo operation 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.

where $expr \in Expr(V)$ and $\sim \in \{=, \neq, <, >, \leq, \geq\}$. The set of all boolean expressions is denoted by $\Phi(V)$. The set of *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 $expr \in Expr(V)$. The set of all actions is denoted by $Act(V)$.

Clocks. Let X be a finite set of real variables, called *clocks*. The set of *clock constraints* over X is defined by the following grammar:

$$\psi := true \mid x \sim c \mid x_1 - x_2 \sim c \mid \psi \wedge \psi$$

where $x, x_1, x_2 \in X, c \in \mathbb{N}$ and $\sim \in \{\leq, <, =, >, \geq\}$. By $\Psi(X)$ we denote the set of all clock constraints. Moreover, let X^* be the set $X \cup x_0$, where x_0 is a clock whose value is always 0 (its value does not increase with time as values of the other clocks). Then, an *assignment* is a function $\varrho : X \rightarrow X^*$. $Asg(X)$ denotes the set of all assignments over X.

Definition 1 (Syntax) *A* timed automaton with discrete data *is a tuple* $\mathcal{T}A = (\Sigma, Q, q^0, V, X, T, \mathcal{I})$*, where*

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

Each element $t = (q, l, \phi, \psi, u, \alpha, \varrho, q')$ 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 enabling conditions for t, u is the urgency attribute – when it is set, the transition t has to be executed as soon as it is enabled (we assume that clock constraints of urgent transitions are equal *true*[3]), α is the action to be executed and ϱ is the clocks assignment. The invariant function assigns to each location $q \in Q$ a clock constraint defining the condition under which an automaton can stay in q. We write *source(t), label(t), guard(t), delay(t), urgency(t), action(t), asgn(t)* and *target(t)* for q, l, ϕ, ψ, u, α, ϱ and q', respectively.

In our example in Section 2, for the sake of clarity, we skip labels of local (non synchronous) transitions and clock assignments of the form $x = x$. We assume

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

that all transitions are urgent except those having clock constraints different from *true*.

The definition of automata used as an input of UppAal [5] is slightly different than ours. Instead of urgent transitions, it introduces urgent labels of synchronous transitions. Our approach is a little bit more general as it allows any transition to be urgent, not necessary synchronous one. Also, UppAal supports so-called committed locations. In such locations time cannot elapse. We model such a behavior by setting as urgent all transitions going out of the location.

3.2 Semantics

This section shows how to associate a labeled transition system with a timed automaton with discrete data.

Variables Valuation. We define a *variables valuation* to be a total mapping $v : V \to \mathbb{Z}$. We extend this mapping to expressions in the usual way. Satisfiability of a boolean expression $\phi \in \Phi(V)$ by a valuation v (denoted as $v \models \phi$) 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 $v \models g_2$, $v \models \neg g$ iff $v \not\models g$. Let $v(\alpha)$ denote a valuation v' defined inductively as follows:

- for $\alpha = \varepsilon$, $v' = v$,
- for $\alpha = (y := expr;)$, for all $z \in V$, if $z = y$, then $v'(z) = v(expr)$, otherwise $v'(z) = v(z)$,
- for $\alpha = \alpha_1\ \alpha_2$, $v' = (v(\alpha_1))(\alpha_2)$.

Clock Valuation. A *clock valuation* is a total mapping $\tau : X \to \mathbb{R}_+$. Satisfiability of a clock constraint $\psi \in \Psi(X)$ by a clock valuation τ (denoted as $\tau \models \psi$) is defined inductively as follows: $\tau \models true$, $\tau \models x \sim c$ iff $\tau(x) \sim c$, $\tau \models x_1 - x_2 \sim c$ iff $\tau(x_1) - \tau(x_2) \sim c$, $\tau \models \psi_1 \wedge \psi_2$ iff $\tau \models \psi_1$ and $\tau \models \psi_2$. For $\delta \in \mathbb{R}_+$, $\tau + \delta$ denotes a clock valuation τ' such that for all $x \in X$, $\tau'(x) = \tau(x) + \delta$. Moreover, by $\tau(\varrho)$ we denote a clock valuation τ' such that for all $x \in X$, if $\varrho(x) \in X$, then $\tau'(x) = \tau(\varrho(x))$, otherwise $\tau'(x) = 0$. Finally, by τ^0 we denote the *initial* clock valuation such that for all $x \in X$, $\tau^0(x) = 0$.

Definition 2 (Semantics) Semantics *of a timed automaton with discrete data $\mathcal{TA} = (\Sigma_A, Q, q^0, V, X, T, \mathcal{I})$ for an initial valuation $v^0 : V \to \mathbb{Z}$ 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* $t = (q, l, \phi, \psi, u, \alpha, \varrho, q') \in T$ *such that* $v \models \phi$, $\tau \models \psi$, $v' = v(\alpha)$ *and* $\tau' = \tau(\varrho) \models \mathcal{I}(q')$,
  - $(q, v, \tau) \xrightarrow{\delta} (q, v, \tau + \delta)$ *iff* $\tau + \delta \models \mathcal{I}(q)$ *and for all transitions* $t = (q, l, \phi, \psi, u, \alpha, \varrho, q') \in T$, $u = false$ *or* $v \not\models \phi$.

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

- either execute an enabled transition t and move to the state (q', v', τ'), where q' is the target of t, the variables valuation is changed according to the action of t and the clocks valuation is changed according to the assignment of t,
- or let time δ pass and move to the state $(q, v, \tau + \delta)$, as long as no urgent transition is enabled and $\tau + \delta$ satisfy $\mathcal{I}(q)$.

Runs. For $(q, v, \tau) \in S$, let $(q, v, \tau) + \delta$ denote $(q, v, \tau + \delta)$. An $s-$*run* π of $\mathcal{TA}$ is a sequence of states: $s_0 \xrightarrow{\delta_0} s_0 + \delta_0 \xrightarrow{a_0} s_1 \xrightarrow{\delta_1} s_1 + \delta_1 \xrightarrow{a_1} s_2 \xrightarrow{a_0} \ldots$, where $s_0 = s \in S$, $a_i \in \Sigma$ and $\delta_i \in \mathbb{R}_+$ for each $i \geq 0$. A run π is called *progressive* iff $\sum_{i \in \mathbb{N}} \delta_i$ is unbounded. $\mathcal{TA}$ is *progressive* if all its runs are progressive. For simplicity in our work we consider only progressive automata. The method of checking progressiveness for timed automata ([23]) can be use also for timed automata with discrete data.

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 and perform transitions with shared labels synchronously. There are various definitions of parallel composition. We choose the one determining *multi-synchronization* which requires that transitions with a shared label have to be executed synchronously by all automata containing this label. To obtain clear semantics of variables updating it is necessary to fix the order of actions in case of synchronous transitions. The transition whose action should be taken first is marked with ! and it is called an *output* transition. Transitions whose actions should be taken next are marked with ? and they are called *input* transitions. We assume additionally that input transitions do not update shared variables.

Let $\mathcal{TA}_1, \ldots \mathcal{TA}_n$ be a set of timed automata with discrete data and let $\Sigma(l) = \{ 1 \leq i \leq n \,|\, l \in \Sigma_i \}$ denote a set of numbers of automata containing the label l. Let also $\widetilde{\Sigma_i} = \bigcup_{l \in \Sigma_i} \{l!, l?, l\}$ and for $\widetilde{l} \in \widetilde{\Sigma_i}$ let $\widetilde{l} \simeq l$, iff $\widetilde{l} \in \{l!, l?, l\}$. By $\{ a_j \}_{j \in Y}$ we denote a sequence (superposition) of actions a_j for $j \in Y \subseteq \{1, \ldots, n\}$.

Definition 3 (Parallel Composition). *Let* $\mathcal{TA}_i = (\widetilde{\Sigma_i}, Q_i, q_i^0, V_i, X_i, E_i, \mathcal{I}_i)$ *for* $i = 1, 2, \ldots, n$ *be a set of timed automata with discrete data such that* $X_i \cap$

$X_j = \emptyset$ for all $i \neq j$. A parallel composition of n automata $\mathcal{TA}_i$ (denoted as $\mathcal{TA}_1 \parallel \mathcal{TA}_2 \parallel \ldots \parallel \mathcal{TA}_n$) is a timed automaton with discrete data $\mathcal{TA} = (\Sigma, Q, q^0, V, X, E, \mathcal{I})$, where:

- $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$,
- $Q = \prod_{i=1}^{n} Q_i$,
- $q^0 = (q_1^0, \ldots, q_n^0)$,
- $V = \bigcup_{i=1}^{n} V_i$,
- $X = \bigcup_{i=1}^{n} X_i$,
- *transition relation E is defined as follows:*
  $((q_1, \ldots, q_n), l, \phi, \psi, u, a, \rho, (q_1', \ldots, q_n')) \in E$ *iff*
  - *for all $i \in \Sigma(l)$ there exists $(q_i, l_i, \phi_i, \psi_i, u_i,\ a_i, \rho_i, q_i') \in E_i$ such that there exists $j \in \Sigma(l)$ for which $l_j = l!$ and $l_j = l?$ for all $i \in \Sigma(l) \setminus \{j\}$ and $\phi = \bigwedge_{i \in \Sigma(l)} \phi_i$, $\psi = \bigwedge_{i \in \Sigma(l)} \psi_i$, $u = \bigvee_{i \in \Sigma(l)} u_i$, $a = a_j\{\, a_i \,\}_{i \in \Sigma(l) \setminus \{j\}}$, $\rho = \bigcup_{i \in \Sigma(l)} \rho_i$ and for all $i \in \{1, \ldots, n\} \setminus \Sigma(l)$, $q_i' = q_i$, or*
  - *for all $i \in \Sigma(l)$ there exists $(q_i, l_i, \phi_i, \psi_i, u_i,\ a_i, \rho_i, q_i') \in E_i$, such that $l_i = l$ and $\phi = \bigwedge_{i \in \Sigma(l)} \phi_i$, $\psi = \bigwedge_{i \in \Sigma(l)} \psi_i$, $u = \bigvee_{i \in \Sigma(l)} u_i$, $a = \{\, a_i \,\}_{i \in \Sigma(l)}$, $\rho = \bigcup_{i \in \Sigma(l)} \rho_i$ and for all $i \in \{1, \ldots, n\} \setminus \Sigma(l)$, $q_i' = q_i$,*
- $\mathcal{I}(q_1, \ldots, q_n) = \bigwedge_{i=1}^{m} \mathcal{I}_i(q_i)$.

Model. Let $\mathcal{S} = (S, s^0, \Sigma, \longrightarrow)$ be the labeled transition system of the parallel composition of a set of automata $\mathcal{TA}_1, \ldots, \mathcal{TA}_n$. An *atomic proposition* is of the form: $\mathcal{TA}_i.q$ for some $1 \leq i \leq n$, where $q \in Q_i$ or $e_1 \sim e_2$, where $e_1, e_2 \in Expr(V)$ and $\sim$ is a relational operator. We denote the set of all atomic propositions by P. In order to reason about systems represented as a set of automata we define a *labeling* function $\mathcal{V} : S \to 2^P$. For $s = (q_1, \ldots, q_n, v, \tau) \in S$, $\mathcal{V}(s)$ is defined as follows: $e_1 \sim e_2 \in \mathcal{V}(s)$ iff $v \models e_1 \sim e_2$ and $\mathcal{TA}_i.q \in \mathcal{V}(s)$ iff $q_i = q$. A *model* is a pair $M = (\mathcal{S}, \mathcal{V})$.

4 Slicing

In model checking applications typically we check if a given temporal logic formula φ is satisfied for a system $\mathcal{P}$. We would like for a sliced system $\mathcal{P}'$ that we produce to satisfy formula φ if and only if the system $\mathcal{P}$ does. But the sliced system needs only to preserve behavior of those parts of the system which can influence the truth of the formula φ.

4.1 Preliminaries

As before let $\mathcal{T}A_1, \ldots, \mathcal{T}A_n$ be a set of timed automata with discrete data, where Q_i and T_i are the set of locations and the of set transitions of the i-th automaton, respectively. We use Q for $\bigcup_{i=1}^{n} Q_i$, $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$ and T for $\bigcup_{i=1}^{n} T_i$, to shorten the notation. For $q \in Q$ let $out(q) = \{t \ \in T \mid q = source(t)\}$ be the set of outgoing transitions and $in(q) = \{t \ \in T \mid q = target(t)\}$ be the set of incoming transitions. A location $q \in Q$ is called an *ending location*, if $out(q) = \emptyset$.

Paths. A *path* in the automaton $\mathcal{T}A_i$ from a location $q_1 \in Q_i$ to a location $q_m \in Q_i$ is a sequence of locations and transitions of the form $q_1 t_2 q_2 \ldots t_m q_m$ such that $m \geq 2$, $q_j \in Q_i$ and $t_j \in out(q_{j-1}) \cap in(q_j)$ for all $j = 2, \ldots, m$. We say that a path $q_1 t_2 q_2 \ldots t_m q_m$ *contains* (or *goes through*) a location q, if there exists $j = 1, \ldots, m$ such that $q = q_j$ (similarly for transitions). A path $q_1 t_2 q_2 \ldots t_m q_m$ is a *cycle*, if $q_1 = q_m$. A cycle is called a *loop*, if $m = 2$. We say that a path contains a cycle if it contains one of its locations twice. We call a path *maximal* if it contains an ending location or a cycle. Finally, we call the location $q' \in Q_i$ *reachable* from the location $q \in Q_i$ (we write $q \Longrightarrow q'$), if there exists a path from q to q'. We assume that each location of an automaton is reachable from its initial location.

Reducible control flow. We say that a location $q_1 \in Q_i$ *dominates* a location $q_2 \in Q_i$, if every path from the initial location q_i^0 to q_2 goes through q_1. Also, a location q_1 *post-dominates* a location q_2, if every maximal path from q_2 goes through q_1.

Various definitions of a reducible flow graph are known. We adopt the one given in [1]. We say also that an automaton has a *reducible control flow*, if its transitions can be divided into two disjoint groups: one forms an acyclic graph and the other consists of transitions whose sources dominate their targets. Such an automaton has an important property that each cycle can be entered through exactly one location. In our work we consider only automata with reducible control flow. This is a secondary simplification since non-reducible control flows rarely occur in practice.

Operations. An atomic assignment of the form $y := e$, where $y \in V$ and $e \in Expr(V)$ and a boolean expression which is a guard of a transition are called *operations*. We use $opers(action(t))$ to denote the set of operations of the action of the transition t and $opers(t)$ for $opers(action(t)) \cup \{guard(t)\}$. $Ops(V) = \bigcup_{t \in T} opers(t)$ denotes the set of all operations of a set of automata. For a location $q \in Q$, $guards(q) = \bigcup_{t \in out(q)} guard(t)$ denotes the set of guards of all transitions going out of q.

Used and define variables. Let $vars(a) \subseteq V$ be the set of variables which occur in the operation $a \in Ops(V)$ and $vars(e) \subseteq V$ be the set of variables which appear in the expression $e \in Expr(V)$. For an operation $a \in Ops(V)$, $def(a) \subseteq V$ is the set of *defined* variables, where $def(a) = \{y\}$ for $a = (y := e)$ and $def(a) = \emptyset$ for $a \in \Phi(V)$. Also, $use(a) \subseteq V$ is the set of *used* variables, defined as $vars(e)$ for $a = (y := e)$ and $vars(a)$ for $a \in \Phi(V)$.

Reaching definitions. Let $v \in V$, $t_1 \in T_i$, $t_2 \in T_j$, where $i, j = 1, \ldots, n$. We say that the definition of v in an operation $a_1 \in \mathit{opers}(t_1)$ is *reaching* for an operation $a_2 \in \mathit{opers}(t_2)$, if $v \in \mathit{def}(a_2) \cap \mathit{use}(a_1)$ and one of the following holds:

- $t_1 = t_2$ and a_2 is followed[4] by a_1 and $v \notin \mathit{def}(a_3)$ for any operation $a_3 \in \mathit{opers}(t_1)$ between a_2 and a_1,
- $v \notin \mathit{def}(a)$ for any $a \in \mathit{opers}(t_2)$ following a_2 and $v \notin \mathit{def}(a)$ for any $a \in \mathit{opers}(t_1)$ preceding a_1 and $\mathit{target}(t_2) = \mathit{source}(t_1)$ or there exists a path from the location $\mathit{target}(t_2)$ to the location $\mathit{source}(t_1)$ such that $v \notin \mathit{def}(a)$ for any $a \in \mathit{opers}(t)$ for any t contained in the path,
- $i \neq j$.

Used and defined clocks. For an assignment $\varrho \in \mathit{Asg}(X)$, $\mathit{def}(\rho) = \{x \in X \mid \rho(x) \neq x\}$ is the set of *defined* clocks and $\mathit{use}(\varrho) = \{x \in X \mid \exists y \in X\ x \neq y \wedge\ \rho(y) = x\}$ is the set of *used* clocks. For a clock constraint $\psi \in \Psi(X)$, let $\mathit{use}(\psi) \subseteq X$ be the set of clocks which appear in ψ. Finally, we define $\mathit{clocks}(q)$ as $\mathit{use}(\mathcal{I}(q)) \cup \bigcup_{t \in \mathit{out}(q)} \mathit{use}(\mathit{delay}(t))$.

4.2 Slicing algorithm

In this part we define the slicing criterion, the dependence relations as well as the slicing algorithm introduced informally in Section 2.

Let $P^\varphi \subseteq P$ be the set of atomic propositions of a formula φ. Let $\mathit{locs}(P^\varphi)$ denote the set of *observable* locations, *i.e.*, locations which appear in propositions of P^φ. Let also $\mathit{vars}(P^\varphi)$ denote the set of *observable* variables, *i.e.*, variables which appear in propositions of P^φ. A slicing criterion for a set of automata is defined by the set of propositions P^φ.

Definition 4 (Slicing criterion) *The* slicing criterion *for a set of timed automata with discrete data $\mathcal{TA}_1, \ldots, \mathcal{TA}_n$ with respect to a set P^φ of atomic propositions is a pair (R^0, A^0), where:*

$$R^0 = \mathit{locs}(P^\varphi) \cup \{q \in Q \mid \exists_{t \in \mathit{out}(q)}\ \mathit{target}(t) \in \mathit{locs}(P^\varphi)\}$$
$$A^0 = \{a \in \mathit{Ops}(V) \mid \mathit{def}(a) \cap \mathit{vars}(P^\varphi) \neq \emptyset\}$$

Definition 5 (Data dependence) *An operation $a_1 \in \mathit{Ops}(V)$ is* data dependent *on an operation $a_2 \in \mathit{Ops}(V)$ (we write $a_1 \xrightarrow{dd} a_2$), if there is a variable $v \in V$ such that the definition of v at a_2 is reaching for a_1.*

Definition 6 (Control dependence) *For two locations q_1, q_2 of the same process Q_i, where $1 \leq i \leq n$, we say that q_1 is* control dependent *on q_2 (we write $q_1 \xrightarrow{cd} q_2$), if the location q_2 is not post-dominated by q_1 and there is a path π from q_2 to q_1 such that each location contained in π different from q_2 is post-dominated by q_1.*

[4] Actions are sequences of operations, so the terms: "follow" and "between" are well defined. Clearly, the action of a transition follows its guard.

Definition 7 (Clock dependence) *A location $q_1 \in Q_i$ is* clock dependent *on a location $q_2 \in Q_i$, for $1 \leq i \leq n$, (we write $q_1 \xrightarrow{xd} q_2$), if there exists a clock $x \in X$ and a transition $t \in out(q_2)$ such that $x \in def(asgn(t)) \cap clocks(q_1)$ and $target(t) = q_1$ or there exists a path π from $target(t)$ to q_1 such that $asgn(t')(x) = x$ for all transitions t' contained in π.*

Definition 8 (Time dependence) *For two locations q_1, q_2 of the same process Q_i, where $1 \leq i \leq n$, we say that q_1 is* time dependent *on q_2 (we write $q_1 \xrightarrow{td} q_2$), if q_1 is reachable from q_2 and the following holds:*

$$\begin{array}{c} (\,\exists_{t \in out(q_2)}\,(urgency(t) \neq true \,\vee\, \Sigma(label(t)) > 1) \\ \vee\,(\bigvee_{t \in out(q_2)}\ guard(t)) \neq true \vee \mathcal{I}(q_2) \neq true\,) \\ \wedge\, \big(\mathcal{I}(q_2) = true \vee \forall_{x \in use(\mathcal{I}(q_2))}\,(\mathcal{I}(q_2) \not\Rightarrow (x = 0) \vee \exists_{t \in in(q_2)} asgn(t)(x) \neq x_0)\big) \end{array}$$

The definition of time dependence uses a safe syntactic approximation of its semantic intuition. If given conditions are violated, then in location q_2 time cannot elapse. Two cases are considered. In the first one, all transitions going out of q_2 are urgent and always one of them is enabled. To check the last condition some heuristic are necessary. We use simple comparison of strings to check, whether guards are of the form b and $\neg b$. In the second case the location invariant contains a constraint of the form $x = 0$, where $x \in X$ and assignments of all incoming transitions contain setting x to x_0.

We define the *dependence relation* $\xrightarrow{d}$ on locations of an automaton to be the union of previously defined relations: $\xrightarrow{cd}$, $\xrightarrow{xd}$ and $\xrightarrow{td}$. The set $A^\varphi \subseteq Ops(V)$ of *relevant operations* and set $R^\varphi \subseteq Q$ of *relevant locations* with respect to the set P^φ of atomic propositions are defined in terms of the transitive closure of the dependence relations by the algorithm shown in Fig. 3. The algorithm describes also how operations, locations and labels depend on each other. The set of relevant labels is denoted by L.

The construction of the sets R and A starts with the slicing criterion (R^0, A^0). Then, in each step we add to the set A the guards of the transitions going out of locations: on which depend the locations from R or having an outgoing transition with an operation from A in its action. Then, the transitive closure of the data dependence relation is computed. To the set R are added locations with at least one outgoing transition: executing an operation from A, guarded by an operation from A or having a label from L. Next, the transitive closure of the dependence relation on the set of locations obtained so far is computed. The set L is augmented by the synchronizing labels of transitions going out of locations from R. The loop ends when no new location, operation nor label is added. By Q^R we denote the set of locations from which there is a path to some location contained in R. Finally, the set of relevant locations Q^φ is composed of the set R and the locations from $Q \setminus Q^R$, which have immediate predecessors in Q^R.

Let us present how Algorithm 1 works for our example. At the beginning $R = \{s_init, s_check\}$, $A = \{s_bit = 1 - s_bit;, r_bit = 1 - r_bit;\}$ and $L = \emptyset$. According to the line 5 of the algorithm to the set A are added the operations

Slicing algorithm.

1. $A := A^0$; $R := R^0$; $L := \emptyset$;
2. $A' := \emptyset$; $R' := \emptyset$; $L' := \emptyset$;
3. **while** $A \neq A' \vee R \neq R' \vee L \neq L'$ **do**
4. $A' := A$; $R' := R$; $L' := L$;
5. $A'' := A \cup \{guards(q) \mid \exists_{q' \in R}\ q'(\xrightarrow{d})^* q \vee \exists_{t \in out(q)}\ opers(t) \cap A \neq \emptyset\}$;
6. $A := A'' \cup \{a \in Ops(V) \mid \exists_{a' \in A''}\ a'\ (\xrightarrow{dd})^*\ a\}$;
7. $R'' := R \cup \{q \in Q \mid \exists_{t \in out(q)}\ opers(t) \cap A \neq \emptyset \vee \exists_{l \in L}\ label(t) \simeq l\}$;
8. $R := R'' \cup \{q \in Q \mid \exists_{q' \in R''}\ q'\ (\xrightarrow{d})^*\ q\}$;
9. $L := L' \cup \{l \in \Sigma \mid |\Sigma(l)| > 1 \wedge \exists_{q \in R}\ \exists_{t \in out(q)}\ label(t) \simeq l\}$;
10. **od**
11. $Q^R := \{q \in Q \mid \exists_{q' \in R}\ q \Longrightarrow q'\}$;
12. $Q^\varphi := R \cup \{q \in Q \setminus Q^R \mid \exists_{t \in in(q)}\ source(t) \in Q^R\}$;
13. $A^\varphi := A$;

Figure 3: Algorithm of computing relevant locations and operations.

$(s_ack = s_bit)$ and $\neg(s_ack = s_bit)$ (as the location $s_init \in R$ control depends on the location s_check) and the operations $(r_tbit = r_bit)$ and $\neg(r_tbit = r_bit)$, (since $r_bit := 1 - r_bit; \in A$). Then, according to the line 6 the set A is successively augmented by the operations on which depend the operations included in A so far, that is $s_ack := b_ack;$ (on which $(s_ack = s_bit)$ depends), $r_tbit := b_bit;$ (on which $(r_tbit = r_bit)$ depends), $b_bit := s_bit;$ (on which $r_tbit := b_bit;$ depends) and both operations $b_ack := r_bit;$. (on which $s_ack := b_ack;$ depends). Next, to the set R are added locations: s_send, r_ack, b_data and b_ack as their outgoing transitions contain operations from A (line 8) and the locations s_wait, r_init and b_init since there are locations from R time dependent on them (line 9). The set L is composed of all synchronizing labels (line 10). The second iteration does not change any of the sets A, L and R, so the loop ends. As in each automata the locations from R are reachable from each location, the set $Q^R = Q$ and the set of the relevant locations $Q^\varphi = R$.

We say that there is an *invisible* path from the location $q_1 \in Q_i$ to the location $q_m \in Q_i$ (we write $q_1 \stackrel{inv}{\Longrightarrow} q_m$), if there is a path $q_1 t_1 q_2 t_2 \ldots q_m$ such that $m > 2$ and $q_j \in Q_i \setminus Q_i^\varphi$ for $j = 2, \ldots, m-1$ (all locations contained in the path except q_1 and q_m are not relevant).

An action reduced to relevant operations $action(t)|_{A^\varphi}$ is defined as $a_1 a_2 \ldots a_k$, where for all $j = 1, \ldots, k$, $a_j \in opers(action(t)) \cap A^\varphi$ and the order of operations is the same as in $action(t)$ (non-relevant operations are erased). We define also a guard reduced to relevant operations $guard(t)|_{A^\varphi}$ as $guard(t)$, if $guard(t) \in A^\varphi$ and $true$, otherwise.

Definition 9 (Slice) *Given a set of timed automata with discrete data* $\mathcal{T}A_i =$

$(\Sigma_i, Q_i, q_i^0, V_i, X_i, T_i, \mathcal{I}_i)$ for $i = 1, 2, \ldots, n$ and a valuation $v^0 : V \to \mathbb{Z}$, where $V = \bigcup_{i=1}^n V_i$, a slice with respect to the set P^φ of atomic propositions is a set of timed automata with discrete data $\mathcal{TA}_i' = (\Sigma_i', Q_i', {q_i^0}', V_i', X_i', T_i', \mathcal{I}_i)'$ for $i = 1, 2, \ldots, n$ and a valuation ${v^0}' : V' \to \mathbb{Z}$, where $V' = \bigcup_{i=1}^n V_i'$, ${v^0}'(V') = v^0(V')$ and for each i such that $Q_i \cap Q^\varphi \neq \emptyset$ the following holds:

- $Q_i' = Q_i \cap Q^\varphi$,
- ${q_i^0}' = \begin{cases} q_i^0, & \text{if } q_i^0 \in Q_i^\varphi \\ q \text{ such that } q_i^0 \stackrel{inv}{\Longrightarrow} q, & \text{otherwise} \end{cases}$
- $V_i' = V_i \cap \bigcup_{a \in A^\varphi} vars(a)$
- $X_i' = X_i \cap \bigcup_{q \in Q^\varphi} clocks(q)$
- *T_i' is the smallest set such that for each $q \in Q_i' \cap Q^R$ and for each $t \in out(q)$ there exists $t' \in T_i'$, where:*
  - $source(t) = source(t')$,
  - $label(t') = label(t)$,
  - $guard(t') = guard(t)|_{A^\varphi}$,
  - $delay(t') = delay(t)$,
  - $urgency(t') = urgency(t)$,
  - $action(t') = action(t)|_{A^\varphi}$,
  - $asgn(t') = asgn(t)|_{X_i'}$,
  - $target(t') = \begin{cases} target(t), & \text{if } target(t) \in Q_i' \\ q \text{ such that } target(t) \stackrel{inv}{\Longrightarrow} q, & \text{otherwise} \end{cases}$
- $\Sigma_i' = \bigcup_{t' \in T_i'} label(t')$,
- $\mathcal{I}_i'(q) = \begin{cases} \mathcal{I}(q), & \text{if } q \in Q_i^R \cap Q_i' \\ true, & \text{otherwise} \end{cases}$

The set of locations of each automaton of the sliced system consists of relevant locations of the original automaton. If an automaton has no relevant locations, it means that the whole automaton is not relevant in context of considered properties. The set of variables of the slice consists of variables of the original system which appear in relevant operations. Similarly for the clocks. In fact, the only clocks that are reduced are used exclusively to ensure that time cannot progress in some locations. For each automaton the set of transitions is composed of transitions of the original automaton going out of relevant locations. If an original transition goes to a non relevant location, then the target of its counterpart in the slice is the relevant location to which an invisible path exits. It can be shown that for a relevant location and each of its outgoing transitions there exists exactly one such location (see the appendix of the paper [18] for the proof).

4.3 Correctness

We conclude with notions of correctness of our slicing method. Let $\mathcal{S} = (S, s^0, \Sigma, \longrightarrow)$ be a labeled transition system for a set of timed automata with discrete data $\mathcal{T}A_1, \ldots, \mathcal{T}A_n$ and $\mathcal{S}' = (S', s^{0'}, \Sigma', \longrightarrow')$ be the labeled transition system for its slice $\mathcal{T}A'_1, \ldots, \mathcal{T}A'_n$ with respect to the set of atomic propositions P^φ. Labeling functions $\mathcal{V} : S \rightarrow 2^{P^\varphi}$ and $\mathcal{V}' : S' \rightarrow 2^{P^\varphi}$ are defined as in Section 3.2.

Definition 10 *Let $s = (q_1, \ldots, q_n, v, \tau) \in S$, $s' = (q'_1, \ldots, q'_n, v', \tau') \in S'$ and $\cong \subseteq S \times S'$. We say that the state s is related to the state s' (we write $s \cong s'$) iff:*

1. *for each pair $q_i \in Q_i$ and $q'_i \in Q'_i$, where $i = 1, \ldots, n$*
   (a) *if $q_i \in Q_i^R \cap Q'_i$, then $q_i = q'_i$, otherwise*
   (b) *if $q_i \in Q_i^R \setminus Q'_i$, then $q_i \stackrel{inv}{\Longrightarrow} q'_i$, otherwise*
   (c) *if $q_i \in Q_i \setminus Q_i^R$, then $q'_i \stackrel{inv}{\Longrightarrow} q_i$ or $q'_i = q_i$ and*
2. *$v(V') = v'(V')$ and*
3. *$\tau'(X') = \tau(X')$*

Lemma 1 *The relation $\cong \subseteq S \times S'$ is a visible bisimulation([12]) between two structures $M = (\mathcal{S}, \mathcal{V})$ and $M' = (\mathcal{S}', \mathcal{V}')$.*

The proof of Lemma 1 can be found in the appendix. It follows from [12] that two structures $M = (\mathcal{S}, \mathcal{V})$ and $M' = (\mathcal{S}', \mathcal{V}')$ are equivalent w.r.t. $\text{CTL}_{\text{-X}}{}^*$ ([9]) formulas if there exists a visible bisimulation between the states of two structures. Thus, the consequence of Lemma 1 is that a model of a system satisfies a $\text{CTL}_{\text{-X}}{}^*$ formula φ, if and only if a model of its slice with respect to the set of propositions P^φ satisfies φ.

5 Conclusions

The paper shows how the slicing technique can be adapted in model checking of systems modeled as a set of timed automata with discrete data. We have extended standard dependency relations by dependencies that arise when concerning time features and presented the method of constructing the sliced system with respect to the set of propositions of the formula to be verified. The presented method of reduction preserves the temporal logic $\text{CTL}_{\text{-X}}{}^*$.

The most important advantage of our approach is that it can be used prior any existing algorithm or tool analyzing timed automata with discrete data. Also it is completely orthogonal to any other abstraction methods and can be combined with them to yield a more powerful tool in terms of state space reduction. The method is efficient since it works on syntactic structure of a system.

References

[1] A Aho, R Sethi, and J Ullman. *Compilers*. Addison-Wesley, 1986.

[2] 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.

[3] K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.

[4] 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.

[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] M. Bozga, J-C. Fernandez, and L. Ghirvu. Using static analysis to improve automatic test generation. In *Proc. of the 6th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 235–250. Springer-Verlag, 2000.

[7] M. Bozga, J-C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier, and J. Sifakis. IF: An intermediate representation for SDL and its applications. In *Proc. of SDL Forum'99*, 1999.

[8] V. Braberman, D. Garbervetsky, and A. Olivero. Improving the verification of timed systems using influence information. In *Proc. 8th 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.

[9] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons for branching-time temporal logic. In *Proc. of Workshop on Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, 1981.

[10] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th ACM Symposium on Principles of Programming Languages*, 1977.

[11] 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 Comp. Soc. Press, 1996.

[12] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150:132–152, 1999.

[13] J. Hatcliff, J. C. Corbett, M. B. Dwyer, S. Sokolowski, and H. Zheng. A formal study of slicing for multi-threaded programs with JVM concurrency primitives. In *Proc. of the Int. Symposium on Static Analysis (SAS'99)*, pages 1–18, 1999.

[14] J. Hatcliff, M. B. Dwyer, and H. Zheng. Slicing software for model construction. In *Proc. of the ACM Workshop on Partial Evaluation and Program Manipulation (PEPM'99)*, volume BRICS NS-99-1 of *BRICS Note Series*, 1999.

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

[16] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Eng.*, 23(5), 1997.

[17] A. Janowska and P. Janowski. Slicing timed systems. *Fundamenta Informaticae*, 60(1-4):187–210, 2004.

[18] A. Janowska and P. Janowski. Slicing of timed automata with discrete data. Technical Report 990, ICS PAS, 2006.

[19] L. Millett and T. Teitelbaum. Slicing promela and its applications to model checking, simulation and protocol understanding. In *Proc. of the 4th Int. SPIN Workshop*, 1998.

[20] W. Nabiałek, A. Niewiadomski, W. Penczek, A. Półrola, and M. Szreter. Verics 2004: A model checker for real time and multi-agent systems. In *Proc. of the Workshop on Concurency, Specification and Programming (CSP'04)*, volume 170 of *Informatik Berichte*. Humboldt University, 2004.

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

[22] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3, 1995.

[23] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.

[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.

A Appendix (proof of Lemma 1)

We begin by defining a few notions and properties that will be useful in the proof. We say, that a transition with the label a is *invisible*, if for every pair of states $s_1, s_2 \in S$, such that $s_1 \xrightarrow{a} s_2$, it holds $\mathcal{V}(s_1) = \mathcal{V}(s_2)$.

Definition 11 (Visible bisimulation [12]). *A relation $\cong_{vb} \subseteq S \times S'$ is a visible simulation between two structures $M = (\mathcal{S}, \mathcal{V})$ and $M' = (\mathcal{S}', \mathcal{V}')$, where $\mathcal{S} = (S, s_0, \Sigma, \longrightarrow)$ and $\mathcal{S}' = (S', s'_0, \Sigma', \longrightarrow')$, if $s_0 \cong_{vb} s'_0$ and for $s \cong_{vb} s'$ the following conditions hold:*

1. *$\mathcal{V}(s) = \mathcal{V}(s')$.*
2. *Let $s \xrightarrow{a} s_1$. There are two cases:*
   - *a is invisible and $s_1 \cong_{vb} s'$, or*
   - *there exists a path $s'_0 \xrightarrow{c_0} s'_1 \xrightarrow{c_1} \dots \xrightarrow{c_{n-1}} s'_n \xrightarrow{c_n} s'_{n+1}$ in M', where $s' = s'_0$ and $s_1 \cong_{vb} s'_{n+1}$, such that $s \cong_{vb} s'_i$ for $0 \leq i \leq n$ and c_i is invisible for $0 \leq i < n$. Furthermore, if a is visible, then $c_n = a$. Otherwise, c_n is invisible.*
3. *If there is an infinite path $s = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots$ in M, where a_i is invisible and $s_i \cong_{vb} s'$ for $i \geq 0$, then there exists a transition $s' \xrightarrow{c} s_1$, such that c is invisible, and for some $j > 0$, $s_j \cong_{vb} s'$.*

A relation $\cong_{vb}$ is a visible bisimulation if both $\cong_{vb}$ and $\cong_{vb}^T$ (the transpose of $\cong_{vb}$) are visible simulations.

We say that a transition $t \in T_i$ and a transition $t' \in T'_i$ *correspond* to each other if and only if $source(t') = source(t)$, $guard(t') = guard(t)|_{A^\varphi}$, $action(t') = action(t)|_{A^\varphi}$, $label(t') = label(t)$ provided $|\Sigma(label(t))| > 1$, $delay(t') = delay(t)$, $urgency(t') = urgency(t)$ and $target(t') = target(t)$ provided $target(t) \in Q'_i$, otherwise $target(t) \overset{inv}{\Longrightarrow} target(t')$.

Let $s = (q_1, \dots, q_n, v, \tau) \in S$. A transition $t \in T_i$ with a label $l \in \Sigma_i$ is *enabled* at a state s (we write $enabled(t, s)$), if for every $j \in \Sigma(l)$ there exists $t \in T_j$ such that $source(t) = q_j$ and $v \models guard(t)$. A transition $t \in T_i$ is *fireable* at a state s (we write $fireable(t, s)$), if $enabled(t, s)$ and $\tau \models delay(t)$.

Proposition 1 *For each location $q \in Q'_i \cap Q^R_i$ and for each transition $t \in out(q) \subseteq T_i$ there exists a corresponding transition $t' \in out(q) \subseteq T'_i$. For each location $q' \in Q'_i$ and for each transition $t' \in out(q') \subseteq T'_i$ there exists a corresponding transition $t \in out(q') \subseteq T_i$.*

Proof:
Follows immediately from Def. 9. □

Proposition 2 *For each location $q \in Q_i \cap Q^R_i$, such that none of the locations from Q'_i depends on q, there exists exactly one location $q_1 \in Q'_i$, such that for all transitions $t \in out(q) \subseteq T_i$, if $target(t) \in Q'_i$, then $target(t) = q_1$, otherwise $target(t) \overset{inv}{\Longrightarrow} q_1$.*

Proof:
Conversely, suppose that there is an invisible path π from q to q_1 and an invisible path π' from q to $q_2 \neq q_1$. Suppose that q is not post dominated by q_1. There are two cases. If the path π contains locations which are not post dominated by q_1, then let q_3 be such a location, which is nearest to q_1. But, in such case q_1 is control dependent on q_3 and $q_3 \in Q'_i$ according to Algorithm 1, which leads to a contradiction with the fact that q_3 is contained in an invisible path. If all locations contained in π except q are post dominated by q_1, then q_1 is control dependent on q, which in turn is a contradiction with the assumption.

Similarly, we show that q is post dominated by q_2. Since both q_1 and q_2 post dominate q, q_1 is reachable from q_2 and q_2 is reachable from q_1 which leads to a contradiction with the fact that the automaton has a reducible control flow. Similar arguments apply to other cases. $\square$

Proposition 3 *Let $q \in Q'_i \cap Q^R_i$, where $1 \leq i \leq n$, $s = (q_1, \ldots, q_n, v, \tau) \in S$, $s' = (q'_1, \ldots, q'_n, v', \tau') \in S'$ and $s \cong s'$.*

1. *If a transition $t \in out(q) \subseteq T_i$ is enabled at s, then each transition $t' \in T'_i$ corresponding to t is enabled at s'.*

2. *If a transition $t \in out(q) \subseteq T_i$ is fireable at s, then there exists a transition $t' \in T'_i$ corresponding to t, which is fireable at s'.*

3. *If a transition $t' \in out(q) \subseteq T'_i$ is enabled at s', then there exits a transition $t \in T_i$ corresponding to t', which is enabled at s.*

4. *If a transition $t' \in out(q) \subseteq T'_i$ is fireable at s', then there exits a transition $t \in T_i$ corresponding to t', which is fireable at s.*

Proof:
Let $t \in out(q) \subseteq T_i$ and $t' \in T'_i$ be a pair of corresponding transitions.

1. Since $enabled(t, s)$ it follows that $q = source(t) = q_i$ and $v \models guard(t)$. By Def. 10.1 we have $q_i = q'_i$. This gives $source(t') = source(t) = q_i = q'_i$. Besides $guard(t') = guard(t)|_{A^\varphi}$. There are two cases: $guard(t') = guard(t)$ or $guard(t') = true$. By Def. 10.2 $v(V') = v'(V')$. We have $v' \models guard(t')$ since $vars(guard(t')) \subseteq V'$. Therefore t' is enabled at s'.

2. Since $fireable(t, s)$ it follows that $enabled(t, s)$. As shown above each transition corresponding to t is enabled at s'. By Def. 10.3 $\tau = \tau'$. Next, by Def. 9 there exists a transition t' corresponding to t such that $delay(t) = delay(t')$. From this we have $\tau' = \tau \models delay(t) = delay(t')$. Therefore t' is fireable at s'.

3. Assume $enabled(t', s')$. Proposition 1 shows that there exists transition $t \in T_i$ corresponding to t'. We have two cases: there exists transition $t \in T_i$ corresponding to t' such that $guard(t) = guard(t')$ or for each transition $t \in T_i$ corresponding to t', $guard(t) \neq guard(t') = true$. In

the first case the same reasoning as above applies. In the other case none of the locations from $Q'_i \cap Q^R_i$ depends on q. Indeed, if such a location exists, then according to Algorithm 1 the guards of its outgoing transitions would be relevant and $guard(t')$ would be equal to $guard(t)$. Similarly, we have $|\Sigma(label(t)| = 1$. Next, by Def. 8 there is at least one transition $t'' \in out(q)$ enabled at s and $urgency(t'') = urgency(t) = urgency(t')$. Then, from $guard(t'') \neq guard(t') = true$ we conclude that there is no relevant operation in action of t''. Thus $action(t') = \varepsilon = action(t'')|_{A^\varphi}$. Since none of the locations from $Q'_i \cap Q^R_i$ depends on q, Proposition 2 holds, namely: if $target(t') \in Q'_i$, then $target(t'') = target(t')$, otherwise $target(t'') \stackrel{inv}{\Longrightarrow} target(t')$. Combining these we obtain that t'' correspond to t' end t'' is enabled at s.

4. The proof runs as before.

□

Proposition 4 *Let* $s = (q_1, \ldots, q_n, v, \tau) \in S$, $s' = (q'_1, \ldots, q'_n, v', \tau') \in S'$ *and* $s \cong s'$. $q_i \in Q_i \setminus Q^R_i$ *iff* $q'_i \in Q'_i \setminus Q^R_i$.

Proof:
Conversely, suppose that $q_i \in Q_i \setminus Q^R_i$ and $q'_i \in Q'_i \cap Q^R_i$. From this it follows that $q_i \neq q'_i$. Hence, by Def. 10.1(c) we have $q'_i \stackrel{inv}{\Longrightarrow} q_i$. But according to Algorithm 1 the set Q'_i contains the locations from $Q_i \setminus Q^R$, which have predecessors in Q^R_i. Thus, there is no invisible path from q'_i to q_i, which leads to a contradiction. Next, suppose that $q_i \in Q^R_i$ and $q'_i \in Q'_i \setminus Q^R_i$. This gives $q_i \neq q'_i$. Hence, $q_i \stackrel{inv}{\Longrightarrow} q'_i$ must hold to satisfy Def. 10.1(c), which again leads to a contradiction with the fact that the locations from $Q_i \setminus Q^R$, which have predecessors in Q^R_i are contained in Q'_i. □

Proposition 5 *Let* $t \in T_i$, $v'_1 = v_1(action(t))$ *and* $v'_2 = v_2(action(t)|_{A^\varphi})$. *If* $v_1(V') = v_2(V')$, *then* $v'_1(V') = v'_2(V')$.

Proof:
According to the definition all operations of action of t on variables from V' are included in the action reduced to relevant operations in the same order. Since $v_1(V') = v_2(V')$, it follows that after the execution of the same sequence of operations $v'_1(V') = v'_2(V')$. □

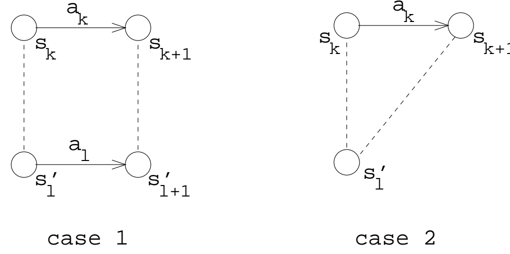
Proof of Lemma 1:
We show that the relation $\cong$ satisfies conditions of Def. 11. It is easy to check that $s_0 \cong s'_0$. We proceed to prove that $\mathcal{V}(s) = \mathcal{V}'(s')$. for $s \in S$, $s' \in S'$, where $s \cong s'$. By Def. 4 and 9 we have $vars(P^\varphi) \subseteq V'$. By Def.10.2 $v'(V') = v(V')$. Hence, $(e_1 \sim e_2) \in \mathcal{V}(s)$ iff $(e_1 \sim e_2) \in \mathcal{V}'(s')$ for $e_1, e_2 \in \Phi(V)$. It remains to show that $(\mathcal{T}A_i.q) \in \mathcal{V}(s)$ iff $(\mathcal{T}A'_i.q) \in \mathcal{V}'(s')$ for $q \in states(P^\varphi)$. By Def. 4 $states(P^\varphi) \subseteq Q'_i \cap Q^R_i$.

1. if $q_i \in Q'_i \cap Q^R_i$, then by Def. 10.1(a) $q'_i = q_i$.

2. if $q_i \in Q_i^R \setminus Q_i'$, then Def. 10.1(b) $q_i \stackrel{inv}{\Longrightarrow} q_i'$. Since $q_i \notin Q_i'$, we have $q \neq q_i$. Suppose $q_i' = q$. According to Def. 4 predecessors of locations from $locs(P^\varphi)$ are contained in the set Q_i'. Therefore there is no invisible path from q_i to $q_i' = q$, which leads to a contradiction.

3. if $q_i \in Q_i \setminus Q_i^R$, then $q_i \neq q$. Proposition 4 gives $q_i' \in Q_i' \setminus Q_i^R$, hence $q \neq q_i'$.

From this we see that $q = q_i$ iff $q = q_i'$ which completes the proof of $\mathcal{V}(s) = \mathcal{V}'(s')$.
$(\Rightarrow)$
Let $s_k = (q_1^k, \ldots, q_n^k, v^k, \tau^k) \in S$, $s' = (q_1^{l\,\prime}, \ldots, q_n^{l\,\prime}, v^{l\prime}, \tau^{l\prime}) \in S'$. In this part of the proof we show that, if $s_k \cong s_l'$ and $s_k \xrightarrow{a_k} s_{k+1}$, then one of the following two cases holds (as in Item 2 of Def. 11):



case 1 there exists state s_{l+1}' and there exists transition with the label a_l', such that $s_l' \xrightarrow{a_l'} s_{l+1}'$ and $s_{k+1} \cong s_{l+1}'$ and if a_k is visible, then $a_l' = a_k$, otherwise a_l' is invisible,

case 2 $s_{k+1} \cong s_l'$ and a_k is invisible.

The proof falls into two parts.

1. $a_k \in \Sigma$. Let TA_i be one of automata performing a transition with the label a_k.

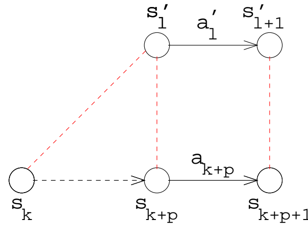
   case 1 In case of $q_i^k \in Q_i' \cap Q^R$ Proposition 3.2 shows that there exists a transition corresponding to t_k, which is fireable at s'. If $target(t_k) \in Q_i$, then $q_i^{k+1} = target(t_k) = target(t_l') = q_i^{l+1\prime}$. Otherwise, by Proposition 2 $q_i^{k+1} = target(t_k) \stackrel{inv}{\Longrightarrow} target(t_l') = q_i^{l+1\prime}$. It is clear that current locations of automata not performing any transition stay unchanged. This proves that Item 1 of the Def. 10 holds for s_{k+1} and s_{l+1}'. Satisfaction of Item 2 follows immediately from Proposition 5. It is easy to check that Item 3 is also satisfied. Combining these gives $s_{k+1} \cong s_{l+1}'$. Clearly, either $label(t_l') = label(t_k)$, which gives $a_l' = a_k$ or $\mathcal{V}(s_k) = \mathcal{V}(s_{k+1})$, which follows that a_k is invisible.

case 2 In case of $q_i^k \in Q_i^R \setminus Q_i'$ Proposition 2 gives $q_i^{k+1} \overset{inv}{\Longrightarrow} q_i^{l'}$. Otherwise, $q_i^k \in Q_i \setminus Q_i^R$ and $q_i^{k+1} \in Q_i \setminus Q_i^R$. By Def. 10.1(c) $q_i^{l'} \overset{inv}{\Longrightarrow} q_i^k$ or $q_i^k = q_i^{l'}$. Therefore $q_i^{l'} \overset{inv}{\Longrightarrow} {q_i}^{k+1}$. We see at once that Item 2 and 3 of Def. 10 are also satisfied. By the above $s_{k+1} \cong s_l'$. Since $\mathcal{V}(s_k) = \mathcal{V}(s_{k+1})$, it follows that a_k is invisible.

2. $a_k \in \mathbb{R}_+$, which represents time progress. In this case $q_i^{k+1} = q_i^k$ for all i, $v^{k+1} = v^k$ and $\tau^{k+1} = \tau^k + a_k$. Obviously, $q_i^{l+1'} = q_i^{l'}$ for all i 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 time progress of amount a_k is possible at the state s_l'. Recall that a timed transition of duration $a_k > 0$, can be performed at s_l' iff for all transitions $t' \in T_i'$, $\neg enabled(t', s_l')$ or $\neg urgency(t')$ and $\tau + a_k \models \mathcal{I}'(q_i')$ for all i. Let $t' \in T_i'$ for arbitrary i. Suppose that $enabled(t', s_l')$ and $urgency(t')$. There are three cases. If $q_i^k \in Q_i' \cap Q_i^R$, then by Def. 10 it follows that $q_i^{l'} = q_i^k$. Hence, from Proposition 3.3 there exists $t \in T$ corresponding to t' that is enabled at the state s_k. A contradiction with time progress. If $q_i^k \in Q_i^R \setminus Q_i'$, then it follows from Def. 8 that time cannot progress at the location q_i^k, which again leads to a contradiction. If $q_i^k \in Q_i \setminus Q_i^R$, then from Proposition 4 we see that $q_i^{l'} \in Q_i' \setminus Q_i^R$. But by Def. 9 we have that $q_i^{l'}$ is an ending location. A contradiction. This proves that for all transitions $t' \in T_i'$, $\neg enabled(t', s_l')$ or $\neg urgency(t')$. It follows easily from $\tau + a_k \models \mathcal{I}(q_i)$ that $\tau + a_k \models \mathcal{I}'(q_i')$ for all i.

($\Leftarrow$)

In the last part of the proof we show that if $s_k \cong s_l'$ and $s_l' \xrightarrow{a_l'} s_{l+1}'$ in M', then there exists a path $s_k \xrightarrow{a_k} s_{k+1} \xrightarrow{a_{k+1}} \dots \xrightarrow{a_{k+p-1}} s_{k+p} \xrightarrow{a_{k+p}} s_{k+p+1}$ in M, such that $s_{k+p+1} \cong s_{l+1}'$ and $s_{k+j} \cong s_l'$ for $0 \le j \le p$ and the transitions with labels a_{k+j} are invisible for $0 \le j < p$. Furthermore, if a transition with the label a_l' is visible, then $a_{k+p} = a_l'$, otherwise transition with the label a_{k+p} is invisible.



Again, there are two cases:

1. $a_l' \in \Sigma'$ is an action transition. Let $\mathcal{T}A_i$ be the one of automata performing the transition with the label a_l' and t_l' be this transition. We give only the main ideas for this part of the proof.

Execution of the transition t'_l in $\mathcal{S}'$ corresponds the execution of a sequence of transitions $t_k, \ldots, t_{k+p}$ in $\mathcal{S}$ for $p \geq 0$, such that the transition t_{k+p} corresponds to the transition t'_l and if $p > 0$, then $target(t_{p+k-1}) = q_i^{k+p} = q_i^{l'}$. Intuitively, if $q_i^k = q_i^{l'}$, which means that the current location of $\mathcal{T}A_i$ is relevant, then $\mathcal{T}A_i$ executes the transition corresponding to t'_l. Otherwise, if $q_i^k \overset{inv}{\Longrightarrow} q_i^{l'}$, then $\mathcal{T}A_i$ executes the sequences of transitions along an invisible path from q_i^k to $q_i^{k+p} = q_i^{l'}$ and the transition corresponding to t'_l.

It is straightforward that t_{k+j} is fireable at s_{k+j} for $j = 0, \ldots, p$. The proof of $s_{k+j} \cong s'_l$ for $0 \leq j \leq p$ can be handled in much the same way as the proof in the previous part.

Since $s_{k+j} \cong s'_l$ and $s_{k+j+1} \cong s'_l$ for $0 \leq j < p$, we obtain $\mathcal{V}(s_{k+j}) = \mathcal{V}'(s'_l) = \mathcal{V}(s_{k+j+1})$. Therefore a_{k+j} are invisible.

Next, since $fireable(t'_l, s'_l)$ from Proposition 3.4 there exists a transition t_p going out of location $q_i^{k+p} \in Q'_i$, corresponding to t'_l, which is fireable at s_{k+p}. Again, we leave it to the reader to verify that $s_{k+p+1} \cong s_{l+1}$.

2. $a'_l \in \mathbb{R}_+$ is a timed transition. The timed transition a'_l in $\mathcal{S}'$ corresponds to a sequence of transitions $t_1^1, \ldots, t_{p_1}^1, t_1^2, \ldots, t_{p_2}^2, \ldots, t_1^n, \ldots, t_{p_n}^n, t$ in $\mathcal{S}$, where t is the timed transition of duration a'_l and for all i, if $q_i^k \in Q_i^R \setminus Q'_i$, then $p_i > 0$, otherwise $p_i = 0$. Each subsequence $t_1^i, \ldots t_{p_i}^i$ is built in the same way as the sequence $t_k, \ldots t_{k+p-1}$ in Item 1 above (each automaton which is not in a relevant location goes to the nearest such a location). The construction is possible by assumption of progressiveness of the set of automata. We skip further details. The proof for $s_{k+p+1} \cong s'_{l+1}$ is similar to the previous cases.

Thus, the conditions of Def. 11 hold and $\cong$ is a visible bisimulation.

Pracę zgłosił Wojciech Penczek

Adresy autorów:

Agata Janowska, Paweł Janowski
Uniwersytet Warszawski
02-097 Warszawa, Banacha 2,
email: {janowska, janowski}@mimuw.edu.pl

D.2.4, F.3.1

Printed as manuscript
Na prawach rękopisu

Nakład 100 egz. Papier kserograficzny kl. III. Oddano do druku w lutym 2006r.
Wydawnictwo IPI PAN

ISSN:0138-0648