

SAT-Based Reachability Checking for Timed Automata with Discrete Data*

Andrzej Zbrzezny[†]

Institute of Mathematics and Computer Science, Jan Długosz University
Armii Krajowej 13/15, 42-200 Częstochowa, Poland
a.zbrzezny@ajd.czyst.pl

Agata Półrola

Faculty of Mathematics and Computer Science, University of Lodz
Banacha 22, 90-238 Lodz, Poland
polrola@math.uni.lodz.pl

Abstract. Reachability analysis for timed automata using SAT-based methods was considered in many papers, occurring to be a very efficient model checking technique. In this paper we show how to apply this method of verification to timed automata with discrete data, i.e., to standard timed automata augmented with integer variables. The theoretical description is supported by some preliminary experimental results.

1. Introduction

Verification of correctness of systems, programs and protocols is a topic of a big practical importance. There exist many methods of testing correctness. One of the most widely used is the *model checking* approach.

The first step towards applying model checking is to describe the system to be verified, using one of the appropriate formalisms. In the case of systems with time dependencies, the most popular description methods are Petri nets with time [12, 16] and timed automata [1]. In order to obtain a more compact

*Partly supported by the Ministry of Science and Higher Education under the grant no. 3 T11C 011 28

[†]Address for correspondence: Institute of Mathematics and Computer Science, Jan Długosz University, Armii Krajowej 13/15, 42-200 Częstochowa, Poland

description, the formalisms are augmented with additional data. In this paper we consider TADD - timed automata with discrete data (i.e., integer variables).

Model checking techniques can be applied to verification of various classes of properties. In this work we focus on testing reachability, i.e., checking whether the system can ever be in a state of certain (usually undesired) features. This, in principle, should require searching through the whole state space of this system (model). However, in most the cases such a search cannot be done due to the *state explosion*, i.e. the fact that the state spaces - in particular these of timed systems - are very large (even infinite). Many methods of dealing with this problem have been developed. In this work, SAT-based verification (i.e., Bounded Model Checking - BMC) is applied.

There exist many publications devoted to verification of timed automata - either standard ones, or extended in various ways [3, 5, 7, 8, 9, 10, 14, 17]. These augmented with integer variables are considered mainly in a series of papers related to the tool UPPAAL [6]. On the other hand, reachability checking is covered in many works [5, 8, 10, 15, 17]. In our paper we consider a subset of automata supported by UPPAAL, and the approach to reachability verification which follows some previous works dealing with SAT-based model checking for standard timed automata [17, 18, 19].

The rest of the paper is organised as follows: in Section 2 we provide preliminary information, among others the definitions of timed automata, their networks and concrete models for them. The next section introduces discretised models for the automata, whereas Section 4 discusses reachability verification. Sections 5 and 6 contain experimental results and final remarks.

2. Preliminaries

Let \mathbb{N} denote the set of naturals (including 0), \mathbb{Z} - the set of integers, \mathbb{Q} - the set of rational numbers, and \mathbb{R} (\mathbb{R}_+) - the set of (non-negative) reals.

2.1. Transition Systems

In our further considerations we need the notion of *transition systems*. So, in what follows by a transition system we mean a tuple $\mathcal{S} = (S, s^0, \Lambda, \rightarrow)$, where S is a set of states, $s^0 \in S$ is the initial state, Λ is a set of labels, and $\rightarrow \subseteq S \times \Lambda \times S$ is a (labelled) transition relation. The system starts at the initial state, and if $(s, \lambda, s') \in \rightarrow$ then it can change its state from s to s' on the label λ . We write $s \xrightarrow{\lambda} s'$ if $(s, \lambda, s') \in \rightarrow$, and $s \rightarrow s'$ if there exists $\lambda \in \Lambda$ s.t. $(s, \lambda, s') \in \rightarrow$. By a *k-path* in \mathcal{S} , for $k \in \mathbb{N}$, we mean a finite sequence $\pi = (s_0, \dots, s_k)$ of states of S s.t. $s_0 = s^0$ and $s_i \rightarrow s_{i+1}$ for any $0 \leq i < k$. We say that a state s is *reachable* in \mathcal{S} if there exists $k \in \mathbb{N}$ and a *k-path* $\pi = (s_0, \dots, s_k)$ such that $s_k = s$.

2.2. Variables

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

$$expr ::= c \mid v \mid v \otimes c \mid c \otimes v \mid v \otimes v,$$

where $c \in \mathbb{Z}$, $v \in IV$ and $\otimes \in \{+, -\}$.

The set of *boolean expressions* over IV , denoted $BoE(IV)$, is defined by

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

where $expr \in Expr(IV)$ and $\sim \in \{=, \neq, <, \leq, \geq, >\}$.

The set of *instructions* over IV , denoted $Ins(IV)$, is given by

$$\alpha ::= \epsilon \mid v := expr \mid \alpha\alpha,$$

where $v \in IV$, $expr \in Expr(IV)$, and ϵ denotes the empty sequence. Thus, an instruction over IV is either an *atomic instruction* over IV ($v := expr$), or a (possibly empty) *sequence* of atomic instructions. Moreover, by $Ins^\diamond(IV)$ we denote the set consisting of all these $\alpha \in Ins(IV)$ in which any $v \in IV$ appears on the left-hand side of “:=” (i.e. is assigned a new value) at most once.

2.2.1. Variables valuation

By a *variables valuation* we mean a total mapping $\mathbf{v} : IV \rightarrow \mathbb{Z}$. We extend this mapping to expressions of $Expr(IV)$ in the usual way. Moreover, we assume that the domain of values for each variable is finite.

The satisfaction relation (\models) for a boolean expression $\beta \in BoE(IV)$ and a valuation \mathbf{v} is defined as: $\mathbf{v} \models true$, $\mathbf{v} \models \beta_1 \wedge \beta_2$ iff $\mathbf{v} \models \beta_1$ and $\mathbf{v} \models \beta_2$, $\mathbf{v} \models \beta_1 \vee \beta_2$ iff $\mathbf{v} \models \beta_1$ or $\mathbf{v} \models \beta_2$, $\mathbf{v} \models \neg\beta$ iff $\mathbf{v} \not\models \beta$, and $\mathbf{v} \models expr_1 \sim expr_2$ iff $\mathbf{v}(expr_1) \sim \mathbf{v}(expr_2)$. Given a variables valuation \mathbf{v} and an instruction $\alpha \in Ins(IV)$, we denote by $\mathbf{v}(\alpha)$ a valuation \mathbf{v}' such that

- if $\alpha = \epsilon$ then $\mathbf{v}' = \mathbf{v}$,
- if $\alpha = (v := expr)$ then for all $v' \in IV$ it holds $\mathbf{v}'(v') = \mathbf{v}(expr)$ if $v' = v$, and $\mathbf{v}'(v') = \mathbf{v}(v')$ otherwise,
- if $\alpha = \alpha_1\alpha_2$ then $\mathbf{v}' = (\mathbf{v}(\alpha_1))(\alpha_2)$.

2.3. Clocks

Let $\mathcal{X} = \{x_1, \dots, x_{n_{\mathcal{X}}}\}$ be a finite set of real-valued variables, called *clocks*. The set of *clock constraints* over \mathcal{X} , denoted $\mathcal{C}(\mathcal{X})$, is defined by the grammar:

$$cc ::= true \mid x_i \sim c \mid x_i - x_j \sim c \mid cc \wedge cc,$$

where $x_i, x_j \in \mathcal{X}$, $c \in \mathbb{N}$, and $\sim \in \{\leq, <, =, >, \geq\}$. Let \mathcal{X}^+ denote the set $\mathcal{X} \cup \{x_0\}$, where $x_0 \notin \mathcal{X}$ is a fictitious clock representing the constant 0. An *assignment* over \mathcal{X} is a function $\mathbf{a} : \mathcal{X} \rightarrow \mathcal{X}^+$. $Asg(\mathcal{X})$ denotes the set of all the assignments over \mathcal{X} .

2.3.1. Clock valuation

By a *clock valuation* we mean a total mapping $\mathbf{c} : \mathcal{X} \rightarrow \mathbb{R}_+$. The satisfaction relation (\models) for a clock constraint $cc \in \mathcal{C}(\mathcal{X})$ and a clock valuation \mathbf{c} is defined as $\mathbf{c} \models true$, $\mathbf{c} \models (x_i \sim c)$ iff $\mathbf{c}(x_i) \sim c$, $\mathbf{c} \models (x_i - x_j \sim c)$ iff $\mathbf{c}(x_i) - \mathbf{c}(x_j) \sim c$, and $\mathbf{c} \models cc_1 \wedge cc_2$ iff $\mathbf{c} \models cc_1$ and $\mathbf{c} \models cc_2$. In what follows, the

set of all the clock valuations satisfying a clock constraint \mathbf{cc} is denoted by $\llbracket \mathbf{cc} \rrbracket$. Given a clock valuation \mathbf{c} and $\delta \in \mathbb{R}_+$, by $\mathbf{c} + \delta$ we denote a clock valuation \mathbf{c}' such that $\mathbf{c}'(x) = \mathbf{c}(x) + \delta$ for all $x \in \mathcal{X}$. Moreover, for a clock valuation \mathbf{c} and an assignment $\mathbf{a} \in \text{Asg}(\mathcal{X})$, by $\mathbf{c}(\mathbf{a})$ we denote a clock valuation \mathbf{c}' such that for all $x \in \mathcal{X}$ it holds $\mathbf{c}'(x) = \mathbf{c}(\mathbf{a}(x))$ if $\mathbf{a}(x) \in \mathcal{X}$, and $\mathbf{c}'(x) = 0$ otherwise (i.e., if $\mathbf{a}(x) = x_0$). Finally, by \mathbf{c}^0 we denote the *initial* clock valuation, i.e., the valuation such that $\mathbf{c}^0(x) = 0$ for all $x \in \mathcal{X}$.

2.4. Timed Automata with Discrete Data

In our paper we assume a definition of timed automata with discrete data which extend the standard timed automata of [1] in the following way:

Definition 2.1. A *timed automaton with discrete data* (TADD) is a tuple $\mathcal{A} = (\mathcal{L}, L, l^0, IV, \mathcal{X}, \mathcal{E}, \mathcal{I})$, where \mathcal{L} is a finite set of labels (actions), L is a finite set of locations, l^0 is the initial location, IV is a finite set of integer variables, \mathcal{X} is a finite set of clocks, $\mathcal{E} \subseteq L \times L \times \text{BoE}(IV) \times \mathcal{C}(\mathcal{X}) \times \text{Ins}^\diamond(IV) \times \text{Asg}(\mathcal{X}) \times L$ is a transition relation, and $\mathcal{I} : L \rightarrow \mathcal{C}(\mathcal{X})$ is an invariant function.

The invariant function assigns to each location a clock constraint expressing the condition under which \mathcal{A} can stay in this location. Each element $t = (l, \mathfrak{l}, \beta, \mathbf{cc}, \alpha, \mathbf{a}, l') \in \mathcal{E}$ denotes a transition from the location l to the location l' , where \mathfrak{l} is the label of the transition t , β and \mathbf{cc} define the enabling conditions for t , α is the instruction to be performed, and \mathbf{a} is the clock assignment. Moreover, for a transition $t = (l, \mathfrak{l}, \beta, \mathbf{cc}, \alpha, \mathbf{a}, l') \in \mathcal{E}$ we write $\text{source}(t)$, $\text{label}(t)$, $\text{vguard}(t)$, $\text{cguard}(t)$, $\text{instr}(t)$, $\text{asgn}(t)$ and $\text{target}(t)$ for l , \mathfrak{l} , β , \mathbf{cc} , α , \mathbf{a} and l' respectively.

Semantics of the above automata is given as follows:

Definition 2.2. *Semantics* of a TADD $\mathcal{A} = (\mathcal{L}, L, l^0, IV, \mathcal{X}, \mathcal{E}, \mathcal{I})$ for an initial variables valuation $\mathbf{v}^0 : IV \rightarrow \mathbb{Z}$ is a labelled transition system $\mathcal{S}(\mathcal{A}) = (Q, q^0, \mathcal{L}_S, \rightarrow)$, where:

- $Q = \{(l, \mathbf{v}, \mathbf{c}) \mid l \in L \wedge \mathbf{v} \in \mathbb{Z}^{|IV|} \wedge \mathbf{c} \in \mathbb{R}_+^{|\mathcal{X}|} \wedge \mathbf{c} \models \mathcal{I}(l)\}$ is the set of states,
- $q^0 = (l^0, \mathbf{v}^0, \mathbf{c}^0)$ is the initial state,
- $\mathcal{L}_S = \mathcal{L} \cup \mathbb{R}_+$ is the set of labels,
- $\rightarrow \subseteq Q \times \mathcal{L}_S \times Q$ is the smallest transition relation defined by the rules:
 - for $\mathfrak{l} \in \mathcal{L}$, $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\mathfrak{l}} (l', \mathbf{v}', \mathbf{c}')$ iff there exists a transition $t = (l, \mathfrak{l}, \beta, \mathbf{cc}, \alpha, \mathbf{a}, l') \in \mathcal{E}$ such that $\mathbf{v} \models \beta$, $\mathbf{c} \models \mathbf{cc}$, $\mathbf{v}' = \mathbf{v}(\alpha)$, $\mathbf{c} \models \mathcal{I}(l)$ and $\mathbf{c}' = \mathbf{c}(\mathbf{a}) \models \mathcal{I}(l')$ (*action transition*),
 - for $\delta \in \mathbb{R}_+$, $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\delta} (l, \mathbf{v}, \mathbf{c} + \delta)$ iff $\mathbf{c}, \mathbf{c} + \delta \models \mathcal{I}(l)$ (*time transition*).

A transition $t \in \mathcal{E}$ is *enabled* at a state $(l, \mathbf{v}, \mathbf{c})$ if $\mathbf{v} \models \text{vguard}(t)$, $\mathbf{c} \models \text{cguard}(t)$ and $\mathbf{c}(\text{asgn}(t)) \models \mathcal{I}(\text{target}(t))$. Intuitively, in the initial state all the variables are set to their initial values, and all the clocks are set to zero. Then, being in a state $q = (l, \mathbf{v}, \mathbf{c})$ the system can either execute an enabled transition t and move to the state $q' = (l', \mathbf{v}', \mathbf{c}')$ where $l' = \text{target}(t)$, the valuation of variables is changed according to $\text{instr}(t)$, and the clock valuation is changed according to $\text{asgn}(t)$, or move to the state $q' = (l, \mathbf{v}, \mathbf{c} + \delta)$ which results from passing some time $\delta \in \mathbb{R}_+$ such that $\mathbf{c} + \delta \models \mathcal{I}(l)$.

We say that a location l (a variables valuation \mathbf{v} , respectively) is *reachable* if some state (l, \cdot, \cdot) ($(\cdot, \mathbf{v}, \cdot)$, respectively) is reachable in $\mathcal{S}(\mathcal{A})$.

2.5. Networks of TADD

We assume that a system to be tested is described by a set (*network*) of timed automata with discrete data $\mathfrak{A} = \{\mathcal{A}_i \mid i = 1, \dots, n\}$ which run in parallel. The automata communicate with each other via shared (i.e., common for some automata) variables, and perform transitions with shared labels synchronously. We assume the scheme of *multisynchronisation*, which requires the transitions with a shared label to be executed synchronously by each automaton that contains this label in its set of labels. To obtain a clear semantics of variable updating it is necessary to fix the order of instructions in the case of synchronous execution of transitions. Thus, the transition whose instruction is to be taken first (called the *output transition*) is marked with the character $!$, whereas these which are to be taken later (the *input* ones) are marked with the character $?$. We assume additionally that the input transitions do not update shared variables, and that for each shared label l there is exactly one $\mathcal{A}_i \in \mathfrak{A}$ in which transitions labelled with l are output ones.

Let $\mathfrak{A} = \{\mathcal{A}_i = (\mathcal{L}_i, L_i, l_i^0, IV_i, \mathcal{X}_i, \mathcal{E}_i, \mathcal{I}_i) \mid i = 1, \dots, n\}$ be a set of TADD. Define $\mathcal{L}(l) = \{1 \leq i \leq n \mid l \in \mathcal{L}_i\}$, which is a set of indices identifying the automata of \mathfrak{A} containing the label l . In order to fix the ordering of instructions, we introduce a partial function $f_{\mathfrak{A}} : \bigcup_{i=1}^n \mathcal{E}_i \rightarrow \{!, ?\}$ which marks the transitions with the common labels with $!$ and $?$, satisfying the conditions: for each t with $|\mathcal{L}(label(t))| > 1$ there exists $i \in \mathcal{L}(label(t))$ such that for each $t' \in \mathcal{E}_i$ with $label(t') = label(t)$ $f_{\mathfrak{A}}(t') = !$ and for all $j \in \mathcal{L}(label(t)) \setminus \{i\}$ and all $t'' \in \mathcal{E}_j$ with $label(t'') = label(t)$ $f_{\mathfrak{A}}(t'') = ?$, whereas for each t with $|\mathcal{L}(label(t))| = 1$ $f_{\mathfrak{A}}(t)$ is undefined. Moreover, given a subset $J = \{j_1, \dots, j_m\}$ of $\{1, \dots, n\}$ and the instructions $\{\alpha_j \in Ins^\diamond(IV) \mid j \in J\}$, define $\bigsqcup_{j \in J} \alpha_j$ as a sequence $\alpha_{j_{k_1}} \dots \alpha_{j_{k_m}}$ with $j_{k_i} \in J$ and $j_{k_i} < j_{k_{i+1}}$ for each $i = 1, \dots, m-1$.

Definition 2.3. Let $\mathfrak{A} = \{\mathcal{A}_i = (\mathcal{L}_i, L_i, l_i^0, IV_i, \mathcal{X}_i, \mathcal{E}_i, \mathcal{I}_i) \mid i = 1, \dots, n\}$ be a set of timed automata with discrete data such that $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$ for all $i, j \in \{1, \dots, n\}$ with $i \neq j$, and let $f_{\mathfrak{A}} : \bigcup_{i=1}^n \mathcal{E}_i \rightarrow \{!, ?\}$ be a partial function which fixes the ordering of instructions. A *parallel composition (product)* of \mathfrak{A} , denoted $\mathcal{A}_1 \parallel \mathcal{A}_2 \parallel \dots \parallel \mathcal{A}_n$, is the timed automaton with discrete data $\mathcal{A} = (\mathcal{L}, L, l^0, IV, \mathcal{X}, \mathcal{E}, \mathcal{I})$, where:

- $\mathcal{L} = \bigcup_{i=1}^n \mathcal{L}_i$, $L = \prod_{i=1}^n L_i$, $l^0 = (l_1^0, \dots, l_n^0)$, $IV = \bigcup_{i=1}^n IV_i$, $\mathcal{X} = \bigcup_{i=1}^n \mathcal{X}_i$,
- the transition relation \mathcal{E} is defined in the following way:
 - $((l_1, \dots, l_n), l, \beta, \mathbf{cc}, \alpha, \mathbf{a}, (l'_1, \dots, l'_n)) \in \mathcal{E}$ iff
 - either there exists $j \in \mathcal{L}(l)$ and $t \in \mathcal{E}_j$ with $label(t) = l$ s.t. $f_{\mathfrak{A}}(t) = !$, and for all $i \in \mathcal{L}(l)$ $(l_i, l, \beta_i, \mathbf{cc}_i, \alpha_i, \mathbf{a}_i, l'_i) \in \mathcal{E}_i$, $\beta = \bigwedge_{i \in \mathcal{L}(l)} \beta_i$, $\mathbf{cc} = \bigwedge_{i \in \mathcal{L}(l)} \mathbf{cc}_i$, $\alpha = \alpha_j \bigsqcup_{i \in \mathcal{L}(l) \setminus \{j\}} \alpha_i$, $\mathbf{a} = \bigcup_{i \in \mathcal{L}(l)} \mathbf{a}_i$, and for all $i \in \{1, \dots, n\} \setminus \mathcal{L}(l)$ it holds $l'_i = l_i$,
 - or there exists $j \in \{1, \dots, n\}$ such that $\mathcal{L}(l) = \{j\}$, $(l_j, l, \beta, \mathbf{cc}, \alpha, \mathbf{a}, l'_j) \in \mathcal{E}_j$, and for all $i \in \{1, \dots, n\} \setminus \{j\}$ it holds $l'_i = l_i$,
- $\mathcal{I}((l_1, \dots, l_n)) = \bigwedge_{i=1}^n \mathcal{I}_i(l_i)$.

Notice that the assumption that the input transitions do not update shared variables ensures that the above definition is correct, since for each instruction α appearing in \mathcal{A} we have $\alpha \in Ins^\diamond(IV)$ (i.e., while performing any instruction in \mathcal{A} each variable is updated at most once). The assumption that the sets of clocks of the automata are disjoint is made for simplicity (without it, fixing the order of clock assignments in the case of synchronous execution of the transitions would be required).

2.6. Concrete Models for TADD

Let $\mathfrak{A} = \{\mathcal{A}_i \mid i = 1, \dots, n\}$ be a set of TADD, $\mathcal{A} = (\mathcal{L}, L, l^0, IV, \mathcal{X}, \mathcal{E}, \mathcal{I})$ be its parallel composition, and $\mathcal{S}(\mathcal{A}) = (Q, q^0, \mathcal{L}_S, \rightarrow)$ be the transition system (semantics) for \mathcal{A} . An *atomic proposition* is of the form $\mathcal{A}_i.l$ or $expr_1 \sim expr_2$, where $expr_1, expr_2 \in Expr(IV)$, \sim is a relational operator, and $l \in L_i$. The set of all the atomic propositions of the form $\mathcal{A}_i.l$ is denoted by PV_{loc} , the set of all the atomic propositions of the form $expr_1 \sim expr_2$ - by PV_{var} , we define also $PV := PV_{loc} \cup PV_{var}$. In order to reason about properties of a system represented by \mathfrak{A} , we introduce a *labelling function* $\mathcal{V} : Q \rightarrow 2^{PV}$. For $q = ((l_1, \dots, l_n), \mathbf{v}, \mathbf{c}) \in Q$, $\mathcal{V}(q)$ is given as follows: $expr_1 \sim expr_2 \in \mathcal{V}(q)$ iff $\mathbf{v} \models expr_1 \sim expr_2$, and $\mathcal{A}_i.l \in \mathcal{V}(q)$ iff $l_i = l$. A *concrete model* is a pair $M = (\mathcal{S}(\mathcal{A}), \mathcal{V})$.

3. Discretised Models for TADD

As it is easy to see from the above description, transition systems (and therefore concrete models) for TADD are usually infinite. Infinite number of their states follows from infinity of the time domain only, since we assumed the domain of values of the integer variables to be finite. Moreover, the set \mathcal{L}_S is infinite as well. However, in order to perform a verification in an efficient way, we need to replace the above model by a finite structure which preserves the properties of interest. For simplicity we provide a description for one automaton only, but the method can be easily applied also to networks of TADD.

Given a TADD $\mathcal{A} = (\mathcal{L}, L, l^0, IV, \mathcal{X}, \mathcal{E}, \mathcal{I})$, denote by $c_{max}(\mathcal{A})$ the largest constant appearing in the clock constraints occurring in all the enabling conditions and the values of the invariant function of \mathcal{A} . Moreover, for any $\delta \in \mathbb{R}_+$, let $frac(\delta)$ denote the fractional part of δ , and let $\lfloor \delta \rfloor$ denote its integral part. Denote by $\tilde{\mathcal{S}}(\mathcal{A})$ a transition system for \mathcal{A} which differs from $\mathcal{S}(\mathcal{A})$ in the set of labels only, i.e., $\tilde{\mathcal{S}}(\mathcal{A}) = (Q, q^0, \tilde{\mathcal{L}}_S, \rightarrow)$, with $\tilde{\mathcal{L}}_S = \mathcal{L} \cup [0, c_{max}(\mathcal{A}) + 1]$. It is easy to prove the following lemma:

Lemma 3.1. The following conditions hold:

- a) any location l of a TADD \mathcal{A} is reachable in $\mathcal{S}(\mathcal{A})$ iff it is reachable in $\tilde{\mathcal{S}}(\mathcal{A})$;
- b) any valuation \mathbf{v} of integer variables of a TADD \mathcal{A} is reachable in $\mathcal{S}(\mathcal{A})$ iff it is reachable in $\tilde{\mathcal{S}}(\mathcal{A})$.

Proof: The “ \Leftarrow ” parts of both (a) and (b) are obvious. The “ \Rightarrow ” parts can be proven by an induction on the length of the path on which the location l or the valuation \mathbf{v} is reachable in $\mathcal{S}(\mathcal{A})$. In order to prove (a) it is sufficient to notice that for any time step of a label $\delta > c_{max}(\mathcal{A}) + 1$ one can replace δ by $\delta' \leq c_{max}(\mathcal{A}) + 1$, obtaining a time transition in $\tilde{\mathcal{S}}(\mathcal{A})$. More precisely, if $frac(\delta) = 0$, then we can put $\delta' = c_{max}(\mathcal{A}) + 1$, and $\delta' = c_{max}(\mathcal{A}) + frac(\delta)$ otherwise. To prove (b), one should notice also that replacing δ by δ' does not influence the values of integer variables. \square

The transition systems $\mathcal{S}(\mathcal{A})$ and $\tilde{\mathcal{S}}(\mathcal{A})$ for a given automaton \mathcal{A} are of an infinite (and uncountable) number of labels and of an infinite (and uncountable) number of states. As it has been stated previously, the above fact does not follow from the presence of integer variables, since the number of their possible values is finite. Therefore, in order to obtain a finite structure appropriate for reachability verification, we can follow the solutions for standard timed automata. Thus, we apply the result of Alur and Dill [2] which states that the reachability problem for the propositions of PV_{loc} and $\mathcal{S}(\mathcal{A})$ (and therefore for PV_{loc} and $\tilde{\mathcal{S}}(\mathcal{A})$) can be reduced to the reachability problem for a transition system of finitely many

states and finitely many labels. This can be done by defining an equivalence relation (called *region equivalence*) which equates two states of the same location if they agree on the integral parts and on the ordering of the fractional parts of the values of all the clocks. However, following [19], we apply a weaker variant of the region equivalence, defined below.

The equivalence relation \simeq , called *weak region equivalence*, is defined over the set of all the clock valuations for \mathcal{X} in the following way:

Definition 3.1. For two clock valuations $\mathbf{c}, \mathbf{c}' \in \mathbb{R}_+^{|\mathcal{X}|}$, $\mathbf{c} \simeq \mathbf{c}'$ iff for all $x, x' \in \mathcal{X}$ the following conditions are met:

- a) $\lfloor \mathbf{c}(x) \rfloor = \lfloor \mathbf{c}'(x) \rfloor$,
- b) $\text{frac}(\mathbf{c}(x)) = 0$ iff $\text{frac}(\mathbf{c}'(x)) = 0$,
- c) $\text{frac}(\mathbf{c}(x)) < \text{frac}(\mathbf{c}(x'))$ iff $\text{frac}(\mathbf{c}'(x)) < \text{frac}(\mathbf{c}'(x'))$.

It has been shown in [19] that the following two lemmas hold:

Lemma 3.2. Let \mathcal{X} be a set of clocks, and $\mathbf{c}, \mathbf{c}' \in \mathbb{R}_+^{|\mathcal{X}|}$ be clock valuations s.t. $\mathbf{c} \simeq \mathbf{c}'$. Then, for any clock constraint $\text{cc} \in \mathcal{C}(\mathcal{X})$ it holds $\mathbf{c} \in \llbracket \text{cc} \rrbracket \iff \mathbf{c}' \in \llbracket \text{cc} \rrbracket$.

Lemma 3.3. Let \mathcal{X} be a set of clocks, and $\mathbf{c}, \mathbf{c}' \in \mathbb{R}_+^{|\mathcal{X}|}$ be clock valuations s.t. for any clock constraint $\text{cc} \in \mathcal{C}(\mathcal{X})$, $\mathbf{c} \in \llbracket \text{cc} \rrbracket \iff \mathbf{c}' \in \llbracket \text{cc} \rrbracket$. Then, it holds $\mathbf{c} \simeq \mathbf{c}'$.

The next lemma follows from the definitions of the involved notions in a straightforward way:

Lemma 3.4. Let \mathcal{X} be a set of clocks, and $\mathbf{c}, \mathbf{c}' \in \mathbb{R}_+^{|\mathcal{X}|}$ be clock valuations such that $\mathbf{c} \simeq \mathbf{c}'$. Then, for any $\mathbf{a} \in \text{Asg}(\mathcal{X})$ it holds $\mathbf{c}(\mathbf{a}) \simeq \mathbf{c}'(\mathbf{a})$.

3.1. Discretisation

In order to obtain a structure appropriate for reachability verification using BMC, we apply the discretisation defined in [19]:

Let $\mathcal{A} = (\mathcal{L}, L, l^0, IV, \mathcal{X}, \mathcal{E}, \mathcal{I})$ be a timed automaton with discrete data. For every $m \in \mathbb{N}$ we define $D_m = \{d \in \mathbb{Q} \mid (\exists k \in \mathbb{N}) d \cdot 2^m = k\}$ and $E_m = \{e \in \mathbb{Q} \mid (\exists k \in \mathbb{N}) e \cdot 2^m = k \wedge e \leq c_{\max}(\mathcal{A}) + 1\}$. Next, we choose $D = \bigcup_{m=0}^{\infty} D_m$ as the set of discretised clock values, and $E = \bigcup_{m=1}^{\infty} E_m$ as the set of labels. Then, we obtain the following lemmas [19]:

Lemma 3.5. For every $\mathbf{c} \in \mathbb{R}_+^{|\mathcal{X}|}$ there exists $\mathbf{c}' \in D^{|\mathcal{X}|}$ such that $\mathbf{c} \simeq \mathbf{c}'$.

Lemma 3.6. Let $\mathbf{c} \in \mathbb{R}_+^{|\mathcal{X}|}$ be a clock valuation, $\delta \in [0, c_{\max}(\mathcal{A}) + 1]$, and $m \in \mathbb{N}$. Then, for each $\mathbf{c}' \in D_m^{|\mathcal{X}|}$ such that $\mathbf{c} \simeq \mathbf{c}'$ there exists $\delta' \in E_{m+1}$ such that $\mathbf{c} + \delta \simeq \mathbf{c}' + \delta'$. Moreover, $\mathbf{c}' + \delta' \in D_{m+1}^{|\mathcal{X}|}$.

Next, we define a discretised transition system for \mathcal{A} as $\mathcal{D}(\mathcal{A}) = (Q_D, q^0, \mathcal{L} \cup E, \rightarrow_D)$, where $Q_D = \{(l, \mathbf{v}, \mathbf{c}) \mid l \in L \wedge \mathbf{v} \in \mathbb{Z}^{|IV|} \wedge \mathbf{c} \in D^{|\mathcal{X}|} \wedge \mathbf{c} \in \mathcal{I}(l)\}$, and the transition relation \rightarrow_D consists of the following two types of transitions:

- for $\iota \in \mathcal{L}$, $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\iota}_D (l', \mathbf{v}', \mathbf{c}')$ iff $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\iota} (l', \mathbf{v}', \mathbf{c}')$ in $\tilde{\mathcal{S}}(\mathcal{A})$ (action transition),
- for $\delta \in E$, $(l, \mathbf{v}, \mathbf{c}) \xrightarrow{\delta}_D (l, \mathbf{v}, \mathbf{c} + \delta)$ iff $\mathbf{c} + \delta \models \mathcal{I}(l)$ (time transition).

We can prove the following theorem:

Theorem 3.1. Let $\mathcal{A} = (\mathcal{L}, L, l^0, IV, \mathcal{X}, \mathcal{E}, \mathcal{I})$ be a TADD, $l \in L$ be a location, $\mathbf{v} \in \mathbb{Z}^{|IV|}$ be a valuation of integer variables, and $\mathbf{c} \in \mathbb{R}_+^{|\mathcal{X}|}$ be a valuation of clocks. If the state $(l, \mathbf{v}, \mathbf{c})$ is reachable in $\tilde{\mathcal{S}}(\mathcal{A})$, then there exists a clock valuation $\mathbf{c}' \in D^{|\mathcal{X}|}$ such that $\mathbf{c}' \simeq \mathbf{c}$ and the state $(l, \mathbf{v}, \mathbf{c}')$ is reachable in $\mathcal{D}(\mathcal{A})$.

Proof: Assume that the state $(l, \mathbf{v}, \mathbf{c})$ is reachable in $\tilde{\mathcal{S}}(\mathcal{A})$. Thus, there exists a k -path $\pi = ((l_0, \mathbf{v}_0, \mathbf{c}_0), \dots, (l_k, \mathbf{v}_k, \mathbf{c}_k))$ with $(l_0, \mathbf{v}_0, \mathbf{c}_0) = (l^0, \mathbf{v}^0, \mathbf{c}^0)$ in $\tilde{\mathcal{S}}(\mathcal{A})$ such that $(l_k, \mathbf{v}_k, \mathbf{c}_k) = (l, \mathbf{v}, \mathbf{c})$. We apply the induction on the length of π .

The base case is obvious, since $\mathbf{c}^0 \in \mathbb{R}_+^{|\mathcal{X}|}$ and $\mathbf{c}^0 \in D^{|\mathcal{X}|}$, and therefore $(l^0, \mathbf{v}^0, \mathbf{c}^0)$ is reachable both in $\tilde{\mathcal{S}}(\mathcal{A})$ and in $\mathcal{D}(\mathcal{A})$. For the induction step assume that there exists a state $(l_{k-1}, \mathbf{v}_{k-1}, \mathbf{c}')$ such that $\mathbf{c}' \in D^{|\mathcal{X}|}$, $\mathbf{c}' \simeq \mathbf{c}_{k-1}$ and $(l_{k-1}, \mathbf{v}_{k-1}, \mathbf{c}')$ is reachable in $\mathcal{D}(\mathcal{A})$. Next, consider the following cases:

- a) The last transition of π is a time transition of $\tilde{\mathcal{S}}(\mathcal{A})$, i.e., it is labelled with $\delta \in [0, c_{max}(\mathcal{A}) + 1]$. Thus, we have $l_k = l_{k-1}$, $\mathbf{v}_k = \mathbf{v}_{k-1}$ and $\mathbf{c}_k = \mathbf{c}_{k-1} + \delta$. From Lemma 3.6 there exists $\delta' \in E$ such that $\mathbf{c}' + \delta' \simeq \mathbf{c}_{k-1} + \delta$. This means that the state $(l_k, \mathbf{v}_k, \mathbf{c}' + \delta')$ is reachable in $\mathcal{D}(\mathcal{A})$, and $\mathbf{c}' + \delta' \simeq \mathbf{c}_k$.
- b) The last transition of π is an action transition in $\tilde{\mathcal{S}}(\mathcal{A})$, i.e., it is labelled with some $\iota \in \mathcal{L}$ in $\tilde{\mathcal{S}}(\mathcal{A})$. Therefore, $\mathbf{v}_k = \mathbf{v}_{k-1}(\alpha)$, $\mathbf{c}_k = \mathbf{c}_{k-1}(\mathbf{a})$, and $(l_{k-1}, \mathbf{v}_{k-1}, \mathbf{c}_{k-1}) \xrightarrow{\iota} (l_k, \mathbf{v}_k, \mathbf{c}_k)$ for a transition $(l_{k-1}, \iota, \beta, \mathbf{cc}, \alpha, \mathbf{a}, l_k) \in \mathcal{E}$. From $\mathbf{c}' \simeq \mathbf{c}_{k-1}$ and Lemma 3.2 we have that $(l_{k-1}, \mathbf{v}_{k-1}, \mathbf{c}')$ $\xrightarrow{\iota}_D (l_k, \mathbf{v}_k, \mathbf{c}'(\mathbf{a}))$ is an action transition in $\mathcal{D}(\mathcal{A})$. This means that the state $(l_k, \mathbf{v}_k, \mathbf{c}'(\mathbf{a}))$ is reachable in $\mathcal{D}(\mathcal{A})$, and from Lemma 3.4 we have $\mathbf{c}'(\mathbf{a}) \simeq \mathbf{c}_k$.

□

It is clear that every k -path in $\mathcal{D}(\mathcal{A})$ is also a k -path in $\tilde{\mathcal{S}}(\mathcal{A})$. Thus, each reachable state of $\mathcal{D}(\mathcal{A})$ is also a reachable state of $\tilde{\mathcal{S}}(\mathcal{A})$. From this and from Lemma 3.5 we obtain the following results:

Corollary 3.1. Any location of \mathcal{A} and any valuation of integer variables is reachable in $\tilde{\mathcal{S}}(\mathcal{A})$ iff it is reachable in $\mathcal{D}(\mathcal{A})$.

Corollary 3.2. Given a set of propositional variables $PV = PV_{loc} \cup PV_{var}$ and a concrete model $M = (\mathcal{S}(\mathcal{A}), \mathcal{V})$ with $\mathcal{V} : Q \rightarrow 2^{PV}$. For any $\wp \in PV$, there exists a reachable state q of M satisfying $\wp \in \mathcal{V}(q)$ iff there exists a reachable state q' of a discretised model $M_D = (\mathcal{D}(\mathcal{A}), \mathcal{V}|_{Q_D})$ satisfying $\wp \in \mathcal{V}|_{Q_D}(q')$.

The last corollary means that discretised transition systems and models can be applied to testing reachability.

The transition system $\mathcal{D}(\mathcal{A})$ is of a countable set of states and of a countable set of labels. In order to check reachability in an efficient way, we again apply the solutions shown in [19], i.e, test reachability on *normalised special k -paths*. Given $m \in \mathbb{N}$, denote by $h(m)$ the value $\lfloor \frac{m+1}{2} \rfloor$. A normalised special k -path is a k -path $\pi = (q_0, \dots, q_k)$ s.t. for each even i ($0 \leq i < k$) the transition $q_i \rightarrow_D q_{i+1}$ is a time transition labelled with $\delta \in E_{h(i)+1}$, for each odd i ($0 < i < k$) the transition $q_i \rightarrow_D q_{i+1}$ is an action transition, and all the clock values of q_i ($i = 0, \dots, k$) belong to $D_{h(i)}$. Analogously as in [19] we can obtain the following result, useful for efficient encoding in SAT-based verification (see the next section):

Lemma 3.7. Any location l and any valuation of integer variables \mathbf{v} is reachable in $\mathcal{D}(\mathcal{A})$ iff it is reachable on some normalised k -path of $\mathcal{D}(\mathcal{A})$.

4. Testing Reachability

In order to apply Bounded Model Checking to testing reachability of a state satisfying certain (usually undesired) property, we unfold the transition relation of a given automaton up to some depth k , and encode this unfolding as a propositional formula. Then, the property to be tested is encoded as a propositional formula as well, and satisfiability of the conjunction of these two formulas is checked using a SAT-solver. If the conjunction is satisfiable, one can conclude that a counterexample (a path to an undesirable state) was found. Otherwise, the value of k is incremented. The above process can be terminated when the value of k is equal to the diameter of the system, i.e., to the maximal length of a shortest path between its two arbitrary states.

All the clock values on a normalised k -path are bounded by a number K depending on k and $c_{max}(\mathcal{A})$ [19]. Since the number of locations and values of integer variables is finite, we can represent (encode) each state on a normalised k -path by a bit vector of the length r_k depending on the number of the locations of \mathcal{A} , the number K , the number of the integer variables and clocks, and the maximal possible absolute value of the integer variables. In order to symbolically represent normalised k -paths in $\mathcal{D}(\mathcal{A})$ for a given TADD \mathcal{A} , we use vectors of propositional variables, called *state variables*. Denote by SV a set of state variables, containing the symbols *true* and *false*. Each state of a normalised k -path can be symbolically represented as a valuation of a vector $\bar{\mathbf{w}} = (w_1, \dots, w_{r_k})$ of state variables. If we define a valuation (interpretation) of state variables as $V : SV \rightarrow \{0, 1\}$, then its extension for vectors of r_k state variables $\mathbf{V} : SV^{r_k} \rightarrow \{0, 1\}^{r_k}$ can be given by $\mathbf{V}(w_1, \dots, w_{r_k}) = (V(w_1), \dots, V(w_{r_k}))$. In what follows, we identify for simplicity the states of $\mathcal{D}(\mathcal{A})$ and their symbolic representations. Then, we define the following propositional formulas ($\bar{\mathbf{w}} = (w_1, \dots, w_{r_k})$ and $\bar{\mathbf{w}}' = (w'_1, \dots, w'_{r_k})$ are vectors of state variables):

- $I(\bar{\mathbf{w}})$ - a formula s.t. for every interpretation $V : SV \rightarrow \{0, 1\}$ of state variables, V satisfies $I(\bar{\mathbf{w}})$ iff $\mathbf{V}(\bar{\mathbf{w}})$ is the initial state of $\mathcal{D}(\mathcal{A})$,
- $TT(\bar{\mathbf{w}}, \bar{\mathbf{w}}')$ - a formula s.t. for every interpretation $V : SV \rightarrow \{0, 1\}$ of state variables, V satisfies $TT(\bar{\mathbf{w}}, \bar{\mathbf{w}}')$ iff $\mathbf{V}(\bar{\mathbf{w}}) \xrightarrow{\delta}_D \mathbf{V}(\bar{\mathbf{w}}')$ in $\mathcal{D}(\mathcal{A})$, for some $\delta \in E$,
- $AT(\bar{\mathbf{w}}, \bar{\mathbf{w}}')$ - a formula s.t. for every interpretation $V : SV \rightarrow \{0, 1\}$ of state variables, V satisfies $AT(\bar{\mathbf{w}}, \bar{\mathbf{w}}')$ iff $\mathbf{V}(\bar{\mathbf{w}}) \xrightarrow{l}_D \mathbf{V}(\bar{\mathbf{w}}')$ in $\mathcal{D}(\mathcal{A})$, for some $l \in \mathcal{L}$.

Then, we define the formula $path_k(\bar{\mathbf{w}}_0, \dots, \bar{\mathbf{w}}_k)$ which symbolically encodes all the normalised k -paths starting at the initial state of $\mathcal{D}(\mathcal{A})$:

$$path_k(\bar{\mathbf{w}}_0, \dots, \bar{\mathbf{w}}_k) ::= I(\bar{\mathbf{w}}_0) \wedge \bigwedge_{i=0}^{k-1} T(\bar{\mathbf{w}}_i, \bar{\mathbf{w}}_{i+1}),$$

where $\bar{\mathbf{w}}_i$ are vectors of state variables, $T(\bar{\mathbf{w}}_i, \bar{\mathbf{w}}_{i+1}) = TT(\bar{\mathbf{w}}_i, \bar{\mathbf{w}}_{i+1})$ if $i \bmod 2 = 0$, and $T(\bar{\mathbf{w}}_i, \bar{\mathbf{w}}_{i+1}) = AT(\bar{\mathbf{w}}_i, \bar{\mathbf{w}}_{i+1})$ if $i \bmod 2 = 1$. Given the above formula and a propositional formula $udp(\bar{\mathbf{w}})$ which encodes the set of states satisfying some undesirable property whose reachability is to be checked, we try to establish whether a state that satisfies $udp(\bar{\mathbf{w}})$ can be reached on a k -path starting at the initial state. This is done by checking satisfiability of the formula

$$reach_k ::= path_k(\bar{\mathbf{w}}_0, \dots, \bar{\mathbf{w}}_k) \wedge \bigvee_{i=0}^k udp(\bar{\mathbf{w}}_i).$$

The unfolding of the transition relation (incrementing the value of k) is terminated if either the above formula is satisfiable (which means that a state we are searching for is reachable), or it is impossible for a given SAT-solver to check the satisfiability of the formula (and so in our case no answer can be found).

Besides searching for errors, the BMC technique can be also applied to proving correctness of a system (i.e., showing that the undesired property does not hold for it). Instead of the simplest but inefficient solution, i.e., incrementing the value of k up to the diameter of the system, the method of [18] can be applied. It consists in using a SAT-solver to searching for a minimal possible k such that if the property holds at none of the k -paths then it is unreachable at all, and then, when such a k is found, checking (as described above) whether the property really does not hold on the k -paths. A detailed explanation of the method can be found in [18] (the paper deals with standard timed automata, but the idea of the method is not influenced by the presence of integer variables).

4.1. Implementation Details: Encoding Action Transitions

One of the important elements of the implementation is to generate propositional formulas $TT(\bar{\mathbf{w}}_1, \bar{\mathbf{w}}_2)$ and $AT(\bar{\mathbf{w}}_1, \bar{\mathbf{w}}_2)$ which encode time- and action transitions of a (network of) TADD. Since the unfolding of $TT(\bar{\mathbf{w}}_1, \bar{\mathbf{w}}_2)$ does not differ from that for standard timed automata presented in previous works, we present the details of encoding action transitions of a network of TADD $\mathfrak{A} = \{\mathcal{A}_i = (\mathcal{L}_i, L_i, l_i^0, IV_i, \mathcal{X}_i, \mathcal{E}_i, \mathcal{I}_i) \mid i = 1, \dots, n\}$. The encoding method assumes that the transition relation of a TADD can contain several elements of the same label.

A formula which encodes the action part of the transition relation is a disjunction (over the set $\mathcal{L} = \bigcup_{i=1}^n \mathcal{L}_i$ of all the actions of \mathfrak{A}) of formulas which encode all the transitions labelled with the given action:

$$AT(\bar{\mathbf{w}}_1, \bar{\mathbf{w}}_2) ::= \bigvee_{l \in \mathcal{L}} B_l(\bar{\mathbf{w}}_1, \bar{\mathbf{w}}_2).$$

In order to encode the formula $B_l(\bar{\mathbf{w}}_1, \bar{\mathbf{w}}_2)$ we define some auxiliary sets and functions. Let for $j \in \{1, \dots, n\}$ and $l \in \mathcal{L}$, $Tr_{(j,l)}$ denote the set of all the transitions of \mathcal{A}_j which are labelled with the action l , and let $\mathcal{L}(l)$ be defined as in Sec. 2.5. Now, for each $j \in \mathcal{L}(l)$ let $g_{(j,l)}$ be an arbitrary $1 - 1$ function $g_{(j,l)} : Tr_{(j,l)} \longrightarrow \{0, \dots, |Tr_{(j,l)}| - 1\}$. Then, we define a function $h_{(j,l)}$ on the set $Tr_{(j,l)}$. Namely,

for each $t \in Tr_{(j,l)}$ let $h_{(j,l)}(t) = \langle j, l, source(t), target(t), g_{(j,l)}(t) \rangle$. Eventually, for a given action l , we define a function R_l on the set $\mathcal{L}(l)$. So, for each $j \in \mathcal{L}(l)$, let $R_l(j) = \prod_{t \in Tr_{(j,l)}} \{h_{(j,l)}(t)\}$. In other words, for each $j \in \mathcal{L}(l)$, $R_l(j)$ is a finite sequence (of the length $|Tr_{(j,l)}|$) of tuples of the form $\langle j, l, source(t), target(t), g_{(j,l)}(t) \rangle$, where $t \in Tr_{(j,l)}$. The function R_l is used to generate the set PE_l of all the possible executions of the action l . Now we define the formula $B_l(\bar{w}_1, \bar{w}_2)$ as a disjunction (over the set PE_l) of formulas which encode all the possible executions of the action l , i.e., as

$$B_l(\bar{w}_1, \bar{w}_2) ::= \bigvee_{r \in PE_l} C_r(\bar{w}_1, \bar{w}_2).$$

For a given execution r of the action l the formula $C_r(\bar{w}_1, \bar{w}_2)$ which encodes the execution r is a conjunction of the following formulas: the formula which encodes a concrete execution of the action l in all the components containing l , the formula which encodes staying all the components not containing l in their current locations, and the formula which encodes executing the instruction (i.e. a sequence of assignments) determined by the concrete execution of the action l . A formula which encodes a concrete execution of the action l in a given component (i.e. a concrete local action transition t) is a conjunction of the following formulas: the formula which encodes the change of a location in a given component by the transition t , the formula which encodes the guard of the transition t , the formula which encodes the invariant of the target location of the transition t , and the formula which encodes the enabling condition (on the integer variables) of the transition t .

5. Experimental Results

The algorithm presented in this paper has been implemented (as a part of the tool Verics [11]) in the programming language C++, and preliminary experiments were performed. The computer used was equipped with the processor Intel Pentium D (3000 MHz), 2 GB of main memory, and the system Linux.

The examples we considered in our tests are standard benchmarks used in the literature. The first one was (a modification of) the well-known Alternating Bit Protocol (ABP) [4] that provides a reliable communication over an unreliable network, using one-bit sequence numbers of messages and an acknowledgement to determine whether the message must be retransmitted. The system modelling it consists of three automata: Sender, Receiver and Faulty Buffer, which run in parallel. Since BMC is designed to search for errors, we modified the original automata to simulate an incorrect design, i.e., to have the property “at the beginning and always when Sender receives an acknowledgement, values of Sender’s bit and Receiver’s bits are equal” violated. The system (see Fig. 1) contains nine integer variables (s_data , b_data , s_bit , b_bit , s_ack , b_ack , r_data , r_tbit and r_bit), which we assume to be zero when it starts, and two clocks x_1 and x_2 . The initial locations are coloured. Enabling conditions of the actions and invariants of the locations are given in brackets. For simplicity, the pictures contain only the synchronisation labels of the transitions ($send_data$, rec_data , $send_ack$ and rec_ack); the local actions are shown as unlabelled. Moreover, the names of the propositions of PV_{loc} , labelling the locations, are shortened to contain no automata names (e.g., s_init instead of $Sender.s_init$). We have tested reachability of a state satisfying the property $\varphi ::= Sender.s_init \wedge s_bit \neq r_bit$. To do this, a path of length 14 was required. The boolean formula whose satisfiability was tested to this aim consisted of 22439 variables and 60791 clauses; our implementation needed 1.26 seconds and 6.20 MB of memory to produce it. Its satisfaction was tested by a SAT-solver MiniSat v.1.14 [13]; to check it 0.14 seconds

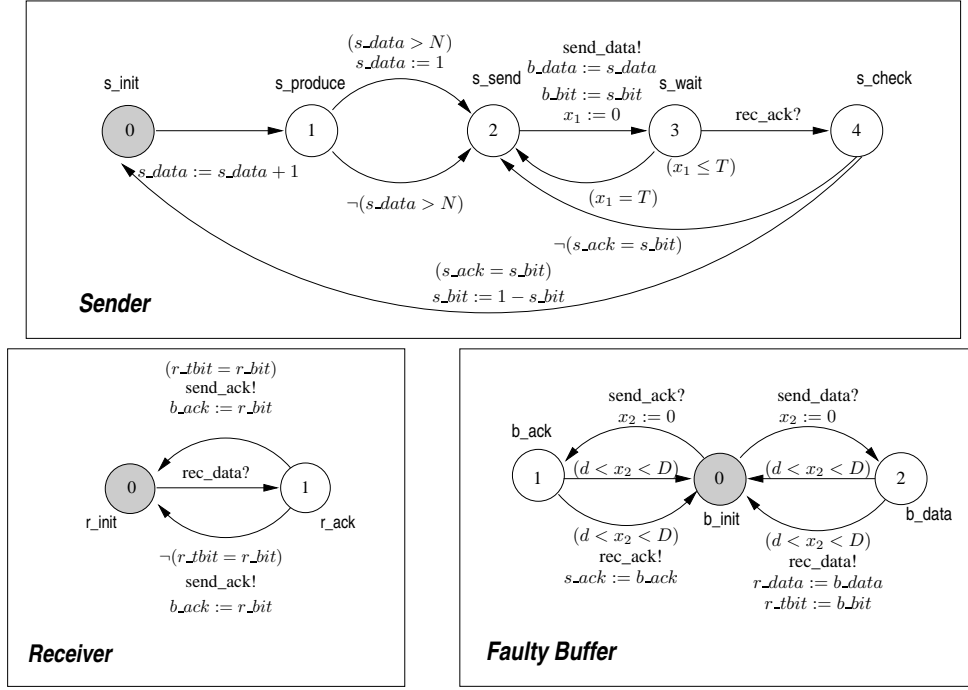


Figure 1. Automata for the Alternating Bit Protocol

and 6.75 MB of memory was used. An error trace (witness) for φ , generated by our implementation, is presented in Table 1 (the ordering of the variables in the table corresponds to that in the description above).

length of the path	locations			integer variables									clocks		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	451
2	1	0	0	1	0	0	0	0	0	0	0	0	0	1	451
3	1	0	0	1	0	0	0	0	0	0	0	0	0	2	393
4	2	0	0	1	0	0	0	0	0	0	0	0	0	2	393
5	2	0	0	1	0	0	0	0	0	0	0	0	0	7	393
6	3	2	0	1	1	0	0	0	0	0	0	0	0	0	0
7	3	2	0	1	1	0	0	0	0	0	0	0	0	1	256
8	3	0	1	1	1	0	0	0	0	0	1	0	0	1	256
9	3	0	1	1	1	0	0	0	0	0	1	0	0	5	256
10	3	1	0	1	1	0	0	0	0	0	1	0	0	5	256
11	3	1	0	1	1	0	0	0	0	0	1	0	0	7	238
12	4	0	0	1	1	0	0	0	0	0	1	0	0	7	238
13	4	0	0	1	1	0	0	0	0	0	1	0	0	9	239
14	0	0	0	1	1	1	0	0	0	0	1	0	0	9	239

Table 1. A witness for the property φ

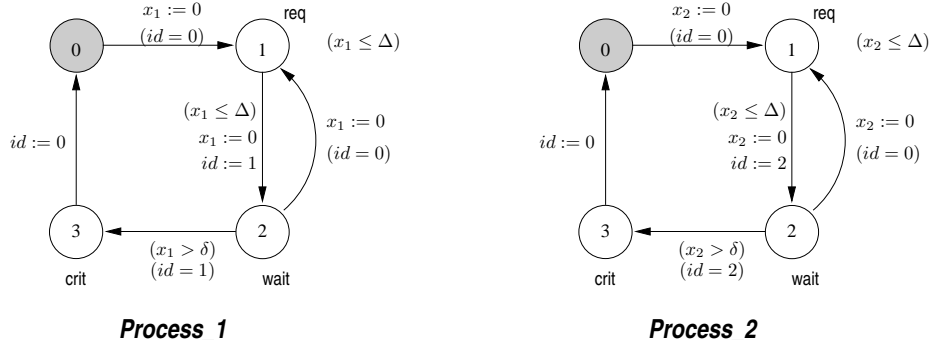


Figure 2. Automata for the mutual exclusion protocol (two processes)

The next example we considered was the standard Fischer’s mutual exclusion protocol (Mutex). The system consists of n processes able to enter a critical section. To coordinate their accesses, a shared variable id is used. The automata for Mutex with $n = 2$ are presented in Fig. 2 (the notations used in the figure follow the patterns introduced for ABP). The behaviour of the system depends on the values of the parameters Δ and δ ; the dependence $\delta < \Delta$ makes the mutual exclusion violated. In this case we tested the three properties: the negation of the standard mutual exclusion property, i.e., $\varphi_1 ::= \bigvee_{i,j \in \{1, \dots, n\}, i \neq j} Process_i.crit \wedge Process_j.crit$ (“at least two processes are in their critical sections at the same time”), and the properties $\varphi_2 ::= \bigwedge_{i \in \{1, \dots, n\}} Process_i.crit$ (“all the processes are in the critical sections at the same time”) and $\varphi_3 ::= Process_1.crit \wedge Process_n.crit$ (“the first and the last process are in their critical sections at the same time”). The results for the above tests, for the values of parameters $\Delta = 2$ and $\delta = 1$, are presented in Table 2. Moreover, we checked correctness of the

property	n	k	variables	clauses	BMC sec	BMC MB	MiniSAT sec	MiniSAT MB
φ_1	2	12	8969	24262	0.2	4.1	0.0	4.7
	10	12	40145	112622	1.1	9.1	1.5	9.9
	20	12	82385	232882	2.7	15.8	5.9	19.9
	50	12	231497	660838	10.1	40.0	18.1	53.0
	100	12	555353	1600106	33.8	44.4	97.8	133.3
	150	12	973777	2823078	72.7	161.9	870.8	450.2
	200	12	1486877	4330078	129.9	247.1	3464.3	990.7
φ_2	2	12	8969	24262	0.2	4.1	0.0	4.7
	3	26	48014	134434	1.2	10.2	1.1	10.4
	4	40	141516	402382	3.9	25.3	99.7	39.4
	5	54	342364	983829	10.0	57.9	1402.0	386.7
	6	68	673315	1946906	20.4	111.9	10304.5	1751.1
φ_3	2	12	8969	24262	0.2	4.0	0.0	4.7
	100	12	426581	1213790	28.4	71.1	2.8	60.6
	500	12	3313565	9616342	627.0	543.1	51.0	440.0
	1000	12	9613697	28193738	2644.6	1585.1	244.8	1348.1

Table 2. Results for the mutual exclusion protocol (n processes)

property	UPPAAL - DFS			UPPAAL - BFS			BMC		
	n	sec	MB	n	sec	MB	n	sec	MB
φ_1	1200	1321	1980	50	1118	1021	200	130 + 3464	247 + 991
φ_2	9	10256	375	10	528	207	6	20 + 10305	112 + 1751
φ_3	17	174	1120	50	738	1743	1000	2645 + 245	1585 + 1348

Table 3. A comparison with UPPAAL

protocol for $\Delta = 1$ and $\delta = 2$. In this case, we were able to verify the mutual exclusion property for the network consisting of 9 processes.

Table 3 provides a comparison of the above results for Mutex with the corresponding ones obtained using the tool UPPAAL run in the BFS and DFS mode (both with the options `-A -S2 -Z`). The values for BMC denote the time (or memory, resp.) needed to generate the set of clauses plus the time (memory, resp.) used to solve its satisfiability via MiniSat. The results allow to assume that although UPPAAL is more effective in the cases when a counterexample can be easily found using the DFS strategy, in the opposite case, i.e. when not many counterexamples exist (and therefore UPPAAL's BFS search gives better results than DFS), SAT-based verification occurs to be far more efficient. Therefore, the two strategies and tools seem to be complementary for practical applications.

6. Conclusions and Further Work

In the paper we have shown how to apply SAT-based reachability verification to the case of timed automata with discrete data. In the future, we are going to extend the method to TADD augmented with urgent transitions and more involved expressions on integer variables. As the next step, we are going to apply bounded model checking to verification of most complex properties of TADD, i.e. the properties expressible by formulas of the logics ACTL and TACTL.

References

- [1] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proc. of the 17th Int. Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.
- [2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [3] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In *Proc. of the 22nd Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE'02)*, volume 2529 of *LNCS*, pages 243–259. Springer-Verlag, 2002.
- [4] 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.
- [5] G. Behrmann, K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Efficient timed reachability analysis using Clock Difference Diagrams. In *Proc. of the 11th Int. Conf. on Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 341–353. Springer-Verlag, 1999.

- [6] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi, and C. Weise. New generation of UPPAAL. In *Proc. of the Int. Workshop on Software Tools for Technology Transfer*, 1998.
- [7] J. Bengtsson and W. Yi. On clock difference constraints and termination in reachability analysis in timed automata. In *Proc. of the 5th Int. Conf. on Formal Methods and Software Engineering (ICFEM'03)*, volume 2885 of *LNCS*, pages 491–503. Springer-Verlag, 2003.
- [8] D. Beyer. Improvements in BDD-based reachability analysis of timed automata. In *Proc. of the Int. Symp. Formal Methods Europe (FME'01)*, volume 2021 of *LNCS*, pages 318–343. Springer-Verlag, 2002.
- [9] M. Bozga, O. Maler, and S. Tripakis. Efficient verification of timed automata using dense and discrete time semantics. In *Proc. of the Conf. on Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *LNCS*, pages 125–141. Springer-Verlag, 1999.
- [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] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Pótroła, 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.
- [12] P. Merlin and D. J. Farber. Recoverability of communication protocols – implication of a theoretical study. *IEEE Trans. on Communications*, 24(9):1036–1043, 1976.
- [13] MiniSat. <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat>, 2006.
- [14] W. Penczek, B. Woźna, and A. Zbrzezny. Towards bounded model checking for the universal fragment of TCTL. In *Proc. of the 7th Int. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02)*, volume 2469 of *LNCS*, pages 265–288. Springer-Verlag, 2002.
- [15] A. Pótroła, W. Penczek, and M. Szreter. Towards efficient partition refinement for checking reachability in timed automata. In *Proc. of the 1st Int. Workshop on Formal Analysis and Modeling of Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 2–17. Springer-Verlag, 2004.
- [16] C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. Technical Report MAC-TR-120, Massachusetts Institute of Technology, February 1974.
- [17] B. Woźna, A. Zbrzezny, and W. Penczek. Checking reachability properties for timed automata via SAT. *Fundamenta Informaticae*, 55(2):223–241, 2003.
- [18] A. Zbrzezny. Improvements in SAT-based reachability analysis for timed automata. *Fundamenta Informaticae*, 60(1-4):417–434, 2004.
- [19] A. Zbrzezny. SAT-based reachability checking for timed automata with diagonal constraints. *Fundamenta Informaticae*, 67(1-3):303–322, 2005.