Automatic Verification of Real–Time Communicating Systems by Constŕaint–Solving *

Wang Yi, Paul Pettersson and Mats Daniels

Department of Computer Systems, Uppsala University, Box 325, S751 05, Uppsala, Sweden. Email: {yi,paupet,matsd}@docs.uu.se

In this paper, an algebra of timed processes with real–valued clocks is presented, which serves as a formal description language for real–time communicating systems. We show that requirements such as "a process will never reach an undesired state" can be verified by solving a simple class of constraint systems on the clock–variables. A complete method for reachability analysis associated with the language is developed, and implemented as an automatic verification tool based on constraint–solving techniques. Finally as examples, we study and verify the safety–properties of Fischer's mutual exclusion protocol and a railway crossing controller.

1. INTRODUCTION

Correct timing plays an important role in ensuring the correct operation of real–time systems. Since such systems are often embedded in safety–critical environments, it is important to formally verify that certain crucial requirements are always met by the systems. This creates a need for formalisms to describe the abstract behavior of timed systems (i.e. modeling) and to check logical properties of the abstract descriptions (i.e. verification). During the past few years, researchers have developed various formal techniques for modeling and verifying real–time systems, e.g. automaton based, [1–3, 14] and process algebra based [24, 7, 9, 15, 22, 17, 13, 20, 29]. One of the most successful approaches is *timed automata* due to Alur and Dill [3], which are the classical finite-state automata extended with variables modeling system clocks.

In this paper, we study real-time communicating systems. Such a system may consist of a number of components with their own or shared clocks. The components may communicate with each other, and the environment through channels according to the timing constraints on the values of the clocks. Naturally, we can use timed automata to describe the components. However, it is not obvious how to combine the component descriptions to achieve the whole system description. Originally, the parallel composition of timed automata is interpreted as logical conjunction, which is similar to the strong (multi–) synchronization operator from process algebras, defined by the rule:

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{a} Q'}{P\&Q \xrightarrow{a} P'\&Q'}$$

*This research is supported in part by the Swedish Board for Technical Development (NUTEK), project No. 93-3244.

D. Hogrefe et al. (eds.), *Formal Description Techniques VII*

Intuitively, it means that the whole system described by $P\&Q$ may make a move (i.e. doing a) only if the components described by P and Q can do the same. That is, all components of a concurrent system must synchronize on every action at every time point. Otherwise, the system will be deadlocked. This seems to be a strong restriction for practical application of timed automata, as real systems are often highly distributed and in many cases, a system component may only want to communicate with the environment or a particular component, without synchronizing with the others. Therefore, we introduce a CCS-like parallel composition operator for timed automata, to describe one-to-one communication and interleaving.

As the first contribution of this paper, we present an algebra of timed automata, which provides a number of algebraic operators including the parallel composition operator to model communication and concurrency. The operators can be used to construct complex automata (i.e. complex system descriptions) in terms of simpler ones (i.e. component descriptions). Thus, the algebra may serve as a *structural* description language for real-time communicating systems.

As the second contribution of the paper, we develop a verification tool based on constraint-solving techniques, for the type of systems described above. There have been a number of verification techniques developed for timed automata, e.g. [2, 1, 14]. However, most of the existing algorithms are based on the notion of region graphs, which always construct the whole reachability graph for a given automaton first and then check properties of the graph. Though there have been efficient algorithms to construct minimal reachability graphs such as [2], the problem of state-explosion is still an obstacle for automatic verification. In particular, when the systems to be analyzed include many components, it would be impossible to construct the whole reachability graph even in the untimed setting.

It has been pointed out in [12] and elsewhere that the practical goal of verification of real-time systems is to verify simple logical properties, which does not need the whole power of model-checking (e.g. for timed CTL). We shall only consider simple safety-properties, which can be verified without constructing the whole reachability graph of a timed system. For instance, a railway control system (see section 4) should guarantee that "at most one train can cross a critical point at the same time". This is a typical safety-property meaning that bad things can never happen. However, we can also verify properties requiring that a *good* thing will eventually happen within a certain time limit. For example, "a train should be able to pass a critical point (such as a bridge), within a bounded delay". We will show that such properties can be verified by solving a simple class of linear constraint systems.

The rest of the paper will be organized as follows: In section 2, we present an algebra of timed processes, in which a syntactical term describes a timed automaton; any timed automaton can be expressed in the algebra. In section 3, we study the reachability problems associated with the algebra. An algorithm is presented, and proved to be sound (i.e. it always provides the right answer) and complete (i.e. it always terminates). It is implemented as a tool, based on an existing constraint-solving program. In section 4, as examples, we study a variant of the railway crossing problem and Fischer's mutual exclusion protocol. Finally, in section 5 we give some concluding remarks.

2. AN ALGEBRA OF PROCESSES WITH CLOCKS

Process algebras provide a clean and general paradigm for compositional specification of communicating processes. We present an algebra of timed automata, serving as a structural description language for real-time communicating systems. The idea is to use algebraic operators to construct complex system descriptions in terms of simpler ones (or component descriptions). Following the tradition of process algebra, we shall call the algebraic terms *processes* instead of timed automata.

2.1. Syntax

Traditionally, a prefix expression $\alpha.P$ in process algebras like CCS describes a process which may perform an α-transition and then continue with P. But no timing information is given on when the transition may be taken.

Following Alur and Dill [3], we assume a set of clocks to specify timing constraints on transitions. Conceptually, the clocks may be considered as the system clocks of a concurrent system, owned or shared by processes in the system. The processes may test the clocks by comparing the clock values with integer constants and reset the clocks (i.e. assigning clock values to 0). Further, assume that all clocks proceed at the same rate and measure the amount of time that has been elapsed since they were reset or started.

We extend the action prefix $\alpha.P$ to the form $(g, \alpha, \phi).P$ where g is a predicate over the clock values and ϕ is a subset of clocks to be reset. Intuitively, $(g, \alpha, \phi).P$ describes a timed process which may perform an α-transition instantaneously when g is true of the current clock values and then continue with P with the clocks in ϕ being reset (and the other clocks will proceed with their old values).

Enabling Conditions. We use C to denote the set of clocks, ranged over by x, y, z. An *enabling condition* g is a logical formula generated by the following syntax:

$$g ::= \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{A} \mid \mathsf{g} \wedge \mathsf{g}$$

where A is an atomic formula of the form: $x \prec n$ for $\prec \in \{\leq, \geq, <, >\}$ and n being a natural number. We could allow a more general form of formulas such as disjunction $g \vee f$. However, it will not give more expressive power to the description language we are going to develop. In fact, logical disjunction can be modeled by the behavioural choice operator.

The language is essentially CCS extended with the timed action prefix $(g, \alpha, \phi).P$. As in CCS, we assume a set $\Lambda = \Delta \cup \bar{\Delta}$[2] with $\bar{\bar{\alpha}} = \alpha$ for all $\alpha \in \Lambda$, ranged over by α, β representing external actions, and a distinct symbol τ representing internal actions. We use $\mathcal{A}ct$ to denote the set $\Lambda \cup \{\tau\}$ ranged over by a, b, c representing both internal and external actions. Further, assume a set of process variables ranged over by X, Y (and sequences of letters).

We shall see that the algebraic structure of a process expression P represents the control–structure of a process. This will be clear when we present the operational semantics. We adopt a two-phase syntax according to two types of control–structures: *regular* and *concurrent.*

[2] The action $\bar{\alpha}$ is called the co-action of α. In our example, we shall use $\alpha!$ instead of $\bar{\alpha}$ to denote an *output* event and $\alpha?$ instead of α to denote an *input* event.

Processes with Regular Control-Structure. We start with processes whose control-structure is regular in the sense that no concurrency is involved. The regular process expressions are generated by the following grammar:

$$E ::= \ \mathsf{nil} \ \Big| \ X \ \Big| \ (g,a,\phi).E \ \Big| \ E+F \ \Big| \ X \overset{def}{=} E$$

We shall restrict expressions to be *well-guarded* in the following sense:

Definition 2.1 *X is well-guarded in E if and only if every free occurrence of X in E is within a subexpression (a guard) of the prefix form $(g,a,\phi).F$ in E. E is well-guarded if and only if every free variable in E is well-guarded in E, and for every subexpression of the form $X \overset{def}{=} F$ in E, X is well-guarded in F.* □

Let $\mathcal{A}$ denote the set of closed and well-guarded expressions generated by the grammar above. We call $\mathcal{A}$ *regular timed processes*. Note that if we consider the prefix (g,a,ϕ) to be a single guard (or structured action), $\mathcal{A}$ corresponds precisely to the set of CCS regular processes.

Processes with Concurrent Control-Structure. We shall study concurrent processes in the form:

$$(P_1|...|P_n)\backslash L$$

where $P_i \in \mathcal{A}$ describing the components and $L \subseteq \Lambda$ representing the set of internal channels connecting the components.

We use $\mathcal{P}$ to denote the set of timed concurrent processes, ranged over by P, Q and R. For simplicity, we have ignored the relabelling operator. The results of this paper can easily be extended to more general types of processes modeled by the combination of parallel composition, restriction and relabelling.

2.2. Semantics

We interpret $\mathcal{P}$ in terms of clock assignments. A clock assignment $\rho : C \longrightarrow \mathcal{R}_{+0}$ is a function mapping each clock x to a non-negative real $\rho(x)$. We assume that a process is always started with an initial clock assignment.

Before going further, we need to define some notation. Assume that d is a non-negative real and ϕ is a set of clocks. We use $\rho + d$ to denote the clock assignment which maps each clock x to $\rho(x) + d$, and $\phi[\rho]$ to denote the clock assignment which maps x to 0 if $x \in \phi$ or $\rho(x)$ otherwise. Furthermore, given a predicate g over C, we write $g(\rho)$ to mean the truth value of g, relative to assignment ρ.

A global state (or a configuration) of a process is a pair (P,ρ) where $P \in \mathcal{P}$ stands for the current control-state and ρ denotes the current clock values. A process may make two types of transitions from state to state:

- Timed transition: $(P,\rho)\overset{d}{\leadsto}(P,\rho+d)$ following the rules given in definition 2.2.
- Action transition: $(P,\rho)\overset{a}{\leadsto}(P',\rho')$ following the rules given in definition 2.3.

The timed transition relation describes the pure passing of time; the action transition relation describes the instantanesous occurrence of actions and, possibly, the resetting of clocks.

Definition 2.2 *(time transition relation)*

$$\frac{d \leq M(E,\rho)}{(E,\rho)\overset{d}{\rightsquigarrow}(E,\rho+d)}$$

where $M(E,\rho)$ is the maximal delay of (E,ρ), defined inductively as follows:

$$\begin{array}{rcl}
M(\mathsf{nil},\rho) & = & \infty \\
M((g,a,\phi).E,\rho) & = & \sup\{t|g(\rho+t)\} \\
M(E+F,\rho) & = & \max\{M(E,\rho),M(F,\rho)\} \\
M(X,\rho) & = & M(E,\rho)\ \textit{if}\ X \overset{def}{=} E \\
M(E|F,\rho) & = & \min\{M(E,\rho),M(F,\rho)\} \\
M(E\backslash L,\rho) & = & M(E,\rho)
\end{array}$$

□

For example, assume $\rho(x) = 0.4$ and $g \equiv x < 1$. Then $\sup\{t|g(\rho+t)\} = 0.6$ and $M((g,a,\phi).E,\rho) = 0.6$.

Intuitively, $M(P,\rho)$ is the maximal time that (P,ρ) may stay in the same control-state i.e. P before it must switch to another control-state. This is also assumed by the *maximal progress assumption* adopted in timed process algebras [9, 29]. For example, the control-state of $X \overset{def}{=} (x < 1, \tau, \{\}).Q$ will become Q by doing the τ-action before the clock value of x proceeds to 1. However, when the value of x is larger than, or equal to 1, the control-state of $Y \overset{def}{=} (x < 1, \tau, \{\}).Q + (x \geq 1, a, \{x\}).R$ will remain the same, i.e. Y, but τ-action will be disabled and a-action will be enabled.

Definition 2.3 *(action transition relation)*

$$\frac{g(\rho)}{((g,\alpha,\phi).E,\rho)\overset{a}{\rightsquigarrow}(E,\phi[\rho])} \qquad \frac{(E,\rho)\overset{a}{\rightsquigarrow}(E',\phi[\rho])}{(E+F,\rho)\overset{a}{\rightsquigarrow}(E',\phi[\rho])} \qquad \frac{(F,\rho)\overset{a}{\rightsquigarrow}(F',\phi[\rho])}{(E+F,\rho)\overset{a}{\rightsquigarrow}(F',\phi[\rho])}$$

$$\frac{(E,\rho)\overset{a}{\rightsquigarrow}(E',\phi[\rho])}{(X,\rho)\overset{a}{\rightsquigarrow}(E',\phi[\rho])}\ [X \overset{def}{=} E] \qquad \frac{(E,\rho)\overset{a}{\rightsquigarrow}(E',\phi[\rho])}{(E|F,\rho)\overset{a}{\rightsquigarrow}(E'|F,\phi[\rho])} \qquad \frac{(F,\rho)\overset{a}{\rightsquigarrow}(F',\phi[\rho])}{(E|F,\rho)\overset{a}{\rightsquigarrow}(E|F',\phi[\rho])}$$

$$\frac{(E,\rho)\overset{a}{\rightsquigarrow}(E',\phi[\rho]) \quad (F,\rho)\overset{\bar{a}}{\rightsquigarrow}(F',\varphi[\rho])}{(E|F,\rho)\overset{\tau}{\rightsquigarrow}(E'|F',(\phi\cup\varphi)[\rho])} \qquad \frac{(E,\rho)\overset{a}{\rightsquigarrow}(E',\phi[\rho])}{(E\backslash L,\rho)\overset{a}{\rightsquigarrow}(E'\backslash L,\phi[\rho])}\ [a,\bar{a} \notin L]$$

□

3. VERIFYING SAFETY-PROPERTIES OF PROCESSES

The language developed in the previous section can be used as a tool to construct the abstract model of an existing system or a system to be designed. In this section, we discuss how to verify properties of such systems in terms of their abstract models.

3.1. Verification by Reachability Analysis

It has been pointed out in [12] and elsewhere that the practical goal of verification of real–time systems, in particular safety–critical systems is to verify simple safety–properties. The type of properties is usually formalized as temporal logic formulas in the form $\Box\neg F$ read as "it is impossible that F will be true in the future". Here, F describes a certain undesired situation or logical property. For example, to verify a railway control system, the first question to ask would be: is it possible that two trains are crossing a certain critical point at the same time? For finite–state systems, this kind of properties can be verified simply by checking all reachable states whether they satisfy F or not, that is, by "reachability analysis". Unfortunately, the systems concerned here are infinite–state because the clock values range over the reals.

Definition 3.1 *(Simple Reachability Problem) Assume $P_0, P_f \in \mathcal{P}$ and ρ_0, ρ_f are clock assignments. We say that (P_f, ρ_f) is reachable from (P_0, ρ_0) iff there is a natural number n and a sequence of transitions starting from (P_0, ρ_0) and ending up with (P_f, ρ_f), i.e. $(P_0, \rho_0)\overset{\sigma_1}{\leadsto}(P_1, \rho_1)...(P_{n-1}, \rho_{n-1})\overset{\sigma_n}{\leadsto}(P_f, \rho_f)$ for $\sigma_i \in \mathcal{A}ct \cup \mathcal{R}_{+0}$.* □

More generally, we will consider the reachability problem for sets of clock assignments.

Definition 3.2 *(General Reachability Problem) Assume $P_0, P_f \in \mathcal{P}$ and D_0, D_f are sets of clock assignments. We say that (P_f, D_f) is reachable from (P_0, D_0) iff there exists $\rho_0 \in D_0$, and $\rho_f \in D_f$ such that (P_f, ρ_f) is reachable from (P_0, ρ_0).* □

We shall develop an algorithm based on constraint–solving techniques, for solving the General Reachability Problem.

3.2. Reachability Analysis by Constraint–Solving

Given a process to be analyzed, we assume that its clocks are ordered as a vector $< x_1, x_2, ..., x_n >$. Then a clock assignment can be considered as a vector of reals or a point in the n-dimensional space $\mathcal{R}^n_{+0}$. We shall use linear constraint systems to describe regions of points in such a space (as their solution sets), and solve the reachability problems by manipulating a simple class of linear constraint systems.

3.2.1. A Class of Linear Constraint Systems

By a *linear constraint system*, we simply mean a set of linear inequalities over a set of variables ranging over $\mathcal{R}_{+0}$ (in our case, the clock variables). A *solution* to such a system is an assignment that maps each variable to a value, which satisfies the set of inequalities. In general, a constraint system may have more than one solution. In the rest of the paper, we shall simply call a constraint system D a *region*, which means its solution set. We shall write (1) $D = \{\}$ to mean D is not satisfiable, (i.e. its solution set is empty), (2) $D \subseteq D'$ to mean that D implies D' (i.e. the solution set of D is included in the solution set of D'), and $D \wedge D'$ to mean the intersection of the solution sets of D and D'.

As we are only allowed to compare clock variables with natural numbers in enabling conditions, the class of constraint systems we need to deal with is restricted to a simple class which we call *time regions.* We shall use $\mathcal{D}$ to denote this class of constraint systems ranged over by D, D'.

Definition 3.3 *(time region) Let $C = \{x_1 ... x_n\}$ be a set of clocks. A time region of C is a constraint system in the following form*[3]*:*

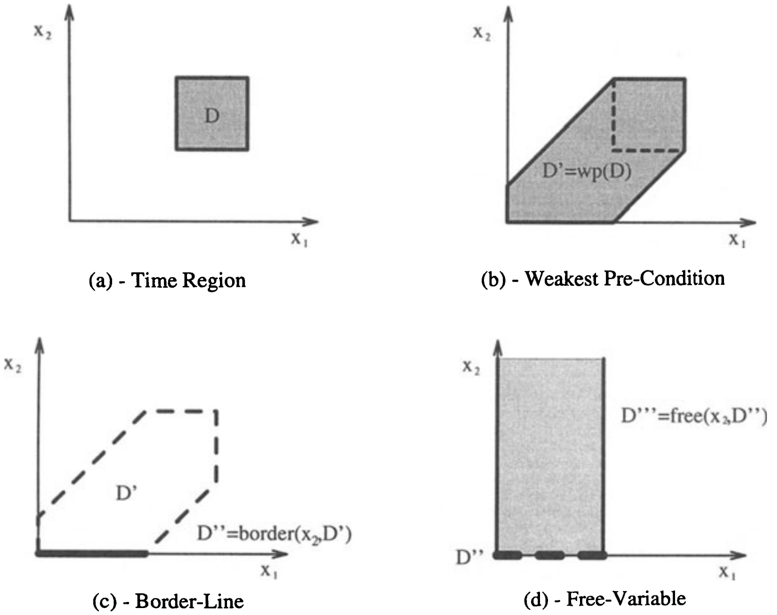
$$D = \{a_i \prec x_i | i \leq n\} \cup \{x_i \prec b_i | i \leq n\} \cup \{x_i - x_j \prec d_{ij} | i, j \leq n\}$$

where $\prec \in \{\leq, <\}$, and a_i, b_i, d_{ij} are natural numbers. In particular, b_i, d_{ij} may be ∞. □

Intuitively, a_i is the lower bound of x_i in D, b_i is the upper bound of x_i in D and d_{ij} is the maximal distance between x_i and x_j in D. We shall need a few operations on time regions in doing reachability analysis.

Definition 3.4 *Assume that D is a time region.*

1. *(Weakest Pre-Condition):* $\mathsf{wp}(D) = \{ \rho \mid \exists d \in \mathcal{R}_{+0} : \rho + d \in D\}$
2. *(Border-Line):* $\mathsf{border}(x, D) = \{ \rho \mid \rho \in D \text{ and } \rho(x) = 0 \}$
3. *(Free-Variable):* $\mathsf{free}(x, D) = \{ \rho[x := d] \mid \rho \in D \text{ and } d \in \mathcal{R}_{+0} \}$ □



(a) - Time Region

(b) - Weakest Pre-Condition

(c) - Border-Line

(d) - Free-Variable

Figure 1. Operations on Time Regions

[3] Note that the symbol $\cup$ here means the union of the sets of constraints. It does not mean the union of the solution sets, rather the intersection of the solution sets.

The three operations on time regions are illustrated in Fig 1 for the case of two clocks. The weakest pre–condition, $\mathsf{wp}(D)$ is the largest region of points that will eventually reach D after some delay, $\mathsf{border}(x_2, D')$ is the border–line of D' on x_1–coordinate (that is the region $D' \wedge \{x_2 = 0\}$) and $\mathsf{free}(x_2, D'')$ is the largest region that has the same projection as D'' on x_1–coordinate.

We will apply the operations $\mathsf{border}(x, D)$ and $\mathsf{free}(x, D)$ on sets of variables. Assume that ϕ is a set of clock variables and $\phi = \{x\} \cup \phi'$. We define $\mathsf{border}(\phi, D) = \mathsf{border}(x, (\mathsf{border}(\phi', D)))$, $\mathsf{free}(\phi, D) = \mathsf{free}(x, (\mathsf{free}(\phi', D)))$ and in particular, $\mathsf{border}(\{\}, D) = \mathsf{free}(\{\}, D) = D$. Furthermore, we define conjunction $D \wedge D'$ of two time regions D, D' in the standard way, that is, $\{\rho | \rho \in D \text{ and } \rho \in D'\}$.

It can be established that the class of constraint systems known as time regions is closed under the operations: "Conjunction", "Weakest Pre–Condition", "Border–line" and "Free-Variable".

Proposition 1 *Assume that C is a set of clocks, D and D' are time regions of C, and $\phi \subseteq C$. Then $D \wedge D', \mathsf{wp}(D), \mathsf{free}(\phi, D)$ and $\mathsf{border}(\phi, D)$ are also time regions of C.* □

We shall use these operations for backward reachability analysis. Similar operations such as strongest post–condition can be defined in order to do forward reachability analysis.

3.2.2. An Algorithm and Its Correctness

Having introduced the notion of time regions, in the following we will simply call (P, D) a region of states, and extend the transition relation $\rightsquigarrow$ to regions.

Definition 3.5 *Assume $a \in \mathcal{A}ct$ and a new symbol ε representing delays.*

1. *$(P, D)\overset{a}{\rightsquigarrow}(P', D')$ iff for all $\rho \in D$, $(P, \rho)\overset{a}{\rightsquigarrow}(P', \rho')$ for some $\rho' \in D'$, and vice versa for all $\rho' \in D'$, $(P, \rho)\overset{a}{\rightsquigarrow}(P', \rho')$ for some $\rho \in D$.*

2. *$(P, D)\overset{\varepsilon}{\rightsquigarrow}(P, D')$ iff for all $\rho \in D$, $(P, \rho)\overset{d}{\rightsquigarrow}(P', \rho')$ for some $d \in \mathcal{R}_{+0}$ and $\rho' \in D'$, and vice versa for all $\rho' \in D'$, $(P, \rho)\overset{d}{\rightsquigarrow}(P, \rho')$ for some $d \in \mathcal{R}_{+0}$ and $\rho \in D$.* □

Now, the general reachability problem can be reformalized as follows:

Proposition 2 *Given an initial region (P_0, D_0) and a final region (P_f, D_f), (P_f, D_f) is reachable from (P_0, D_0) iff there exists a finite number n, $\sigma_i \in \mathcal{A}ct \cup \{\varepsilon\}$ and $D_i \in \mathcal{D}$ for all $i \leq n$, such that $P_n \equiv P_f$, $D_n \wedge D_f \neq \{\}$ and $(P_0, D_0)\overset{\sigma_1}{\rightsquigarrow}(P_1, D_1) \ldots (P_{n-1}, D_{n-1})\overset{\sigma_n}{\rightsquigarrow}(P_n, D_n)$.* □

To achieve a decision algorithm for the problem, we shall take the approach of backward reachability analysis. Usually, to verify safety–properties, a backward analysis algorithm may terminate much faster than a forward analysis algorithm for the following reason: In case that a system does not contain an undesired state, the backward analysis needs not to check the whole reachable state–space of the system (but the forward analysis does), and the probability for a safety–critical system to contain a serious error is often very small.

The general principle of backward analysis is to start from the final and search back to the initial. If the initial is found, the algorithm terminates with answer "yes", otherwise "no". However, our backward analysis method can be easily adopted to forward analysis.

First, we need to study the control–structures more carefully. It has been said earlier that the algebraic structure of a term P describes the control–structure of a process. In fact, the set of subexpressions of P is a superset of the control–states of P, and the transitions among the control–states obey the rules in Fig. 2.

$\dfrac{}{(g,a,\phi).E \xrightarrow{g,a,\phi} E}$	$\dfrac{E \xrightarrow{g,a,\phi} E'}{E+F \xrightarrow{g,a,\phi} E'}$	$\dfrac{F \xrightarrow{g,a,\phi} F'}{E+F \xrightarrow{g,a,\phi} F'}$	$\dfrac{E \xrightarrow{g,a,\phi} E'}{X \xrightarrow{g,a,\phi} E'}\ [X \overset{def}{=} E]$
$\dfrac{E \xrightarrow{g,a,\phi} E'}{E \vert F \xrightarrow{g,a,\phi} E' \vert F}$	$\dfrac{F \xrightarrow{g,a,\phi} F'}{E \vert F \xrightarrow{g,a,\phi} E \vert F'}$	$\dfrac{E \xrightarrow{g,a,\phi} E' \quad F \xrightarrow{f,\bar{a},\varphi} F'}{E \vert F \xrightarrow{g \wedge f,\tau,\phi \cup \varphi} E' \vert F'}$	$\dfrac{E \xrightarrow{g,a,\phi} E' \quad [a,\bar{a} \notin L]}{E \backslash L \xrightarrow{g,a,\phi} E' \backslash L}$

Figure 2. Transition Rules for Control–States.

It should be obvious that each term P_0 describes a timed automaton, i.e. $< C_S, P_0, \longrightarrow >$ where C_S is all control–states reachable from P_0, $\longrightarrow$ is the least transition relation defined by the transitional rules. In particular, note that C_S is finite.

The reachability analysis algorithm is based on the following idea: Assume that we want to decide whether (P', D') may reach (P, D) in one step (i.e. without passing other control–states) or not. The first thing to check is whether it is possible for P' to switch to P directly. If this is not the case, that is, $P' \xrightarrow{g,a,\phi} P$ for no P', g, a, ϕ, we can conclude immediately that (P, D) is not reachable from (P', D') in one step. Now, assume $P' \xrightarrow{g,a,\phi} P$. To reach (P, D), there should be time regions D_0, D_1, D_2 such that $D' \cap D_0 \neq \{\}$ and $(P_0, D_0) \overset{\epsilon}{\leadsto} (P_0, D_1) \overset{a}{\leadsto} (P, D_2) \overset{\epsilon}{\leadsto} (P, D)$. Note that D' and D are given. We need to find D_0, D_1, D_2. Clearly, we may choose

$$
\begin{aligned}
D_2 &= \mathsf{border}(\phi, \mathsf{wp}(D)) \\
D_1 &= g \wedge \mathsf{free}(\phi, D_2) \\
D_0 &= \mathsf{wp}(D_1)
\end{aligned}
$$

That is, $D_0 = \mathsf{wp}(g \wedge [\mathsf{free}(\phi, \mathsf{border}(\phi, \mathsf{wp}(D)))])$. In fact, D_0 is the largest region of points that may (1) pass the guard g and (2) be reset by ϕ, and finally (3) reach D. In general, for any given g, ϕ and D, we define $\mathsf{image}(g, \phi, D) = \mathsf{wp}(g \wedge [\mathsf{free}(\phi, \mathsf{border}(\phi, \mathsf{wp}(D)))])$.

Now, we are ready to present the algorithm, shown in Fig. 3 for backward reachability analysis. We use two buffers for saving regions (of states): passed and waiting where passed stands for the set of regions that have been examined and waiting for the set of regions that are to be examined next.

```
Algorithm. (Input: $P_0, P_f \in \mathcal{P}$ and $D_0, D_f \in \mathcal{D}$ and Output: answer = 'yes' or 'no'.)

1. Initial: passed := {} and waiting := $\{(P_f, D_f)\}$.

2. Repeat
     for all $(P, D) \in$ waiting, do
         begin
         (1) If there is no $D'$ such that $D \subseteq D'$ and $(P, D') \in$ passed then
             begin
                 (a) passed := passed $\cup \{(P, D)\}$ and
                 (b) waiting := waiting $\cup \{(P', D')\}$ for all $P', g, a, \phi$
                     such that $P' \xrightarrow{g,a,\phi} P$, and $D' = \text{image}(g, \phi, D) \neq \{\}$.
             end;
         (2) waiting := waiting $- \{(P, D)\}$
         end
   until waiting = {} or $(P_0, D_0') \in$ waiting for some $D_0'$ such that $D_0 \wedge D_0' \neq \{\}$.

3. Termination: If waiting = {} then answer := 'no'; otherwise answer := 'yes'.
```

Figure 3. An Algorithm for Reachability Analysis.

The algorithm is started with passed = {} and waiting = $\{(P_f, D_f)\}$, and then repeatedly examines the regions in waiting. If a region (P, D) found in waiting is smaller than a region (P, D') (with the same control–state) in passed, then (P, D) does not need to be examined further. Otherwise, put all the regions that may reach (P, D) in one step into waiting to be examined later, and put (P, D) into passed. The algorithm will terminate when waiting is empty (i.e. nothing is left to be examined, and therefore fails to find the initial region) or a region (P_0, D_0') is found, which includes a part of the initial region (P_0, D_0) (i.e. $D_0 \wedge D_0' \neq \{\}$).

It is easy to prove the partial correctness (soundness) of the algorithm: given proper inputs, it always provides the right answer.

Theorem 1 *(Partial Correctness) For all initial regions (P_0, D_0) and final regions (P_f, D_f), if the algorithm terminates with* 'yes'*, then (P_f, D_f) is reachable from (P_0, D_0). Otherwise, (P_f, D_f) is not reachable from (P_0, D_0).* □

It is slightly more difficult to prove the total correctness (completeness) of the algorithm: given proper inputs, it always terminates with an answer.

Theorem 2 *(Total Correctness) For all initial regions (P_0, D_0) and final regions (P_f, D_f), the algorithm always terminates with an answer which is either* 'yes' *or* 'no'*.* □

3.2.3. Implementation

In describing the reachability algorithm, we did not explain how to (1) perform the four operations (*wp, free, border,* $\wedge$) defined on time regions, (2) check the emptiness of a time region (or satisfiability of a constraint system), and (3) set–inclusion (i.e. $\subseteq$) between time regions. In fact, these functions are often provided by constraint solvers or straight forward to implement using primitive functions of a constraint solver.

We have implemented the algorithm as a tool, based on a constraint solver developed at the Swedish Institute of Computer Science called Prolog Constraint Solver (PCS) [21]. Several examples have been used to test the tool (see next section), which show that the implementation is fairly efficient.

4. EXAMPLES

In this section, we present examples which have been verified by our tool. In addition to clock variables, in describing the examples, we shall also use ordinary variables. These variables do not change their values automatically as the clock variables; they can only be assigned to values from finite domains, and therefore they will not cause infinite–stateness. Fortunately, the implementation of our tool is based on a general constraint solver which can handle logical constraints and assignments including ordinary variables in the same way as timing constraints and clock assignments.

4.1. Fischer's Mutual Exclusion Protocol

The protocol was proposed originally by Fischer and described by Lamport [18]. It is to guarantee mutual exclusion in a concurrent systems consisting of several processes using a shared variable (among the processes) and properly timing the processes in changing the shared variable. Each of the processes is assumed to have a local clock. The idea behind the protocol is that the timing constraints on the local clocks are set so that only one process can change the global variable to its own process number, then read the global variable later and if the shared variable is still equal to its own number, enter the critical section.

Assume a concurrent system with n processes $\mathsf{P}_1 ... \mathsf{P}_n$. We use x_i to model the local clock for each process P_i. The formal description of P_i is given in Fig. 4, and illustrated[4] in Fig 5.

This is a simplified version of the original protocol and has been studied by researchers, e.g. [4, 25], which permits only one process to enter the critical section and never exits it. Recovery actions from failure to enter the critical section are omitted. However, the protocol can be extended to an actual mutual exclusion algorithm.

The processes, P_i, may be in either of the four local states $\mathsf{A}_i, \mathsf{B}_i, \mathsf{C}_i, \mathsf{CS}_i$. Initially, all processes are in their A–state and the shared variable v is initially 0. A process, P_i, that tries to enter the critical section changes state from A_i to B_i if it sees v=0. In B_i, it will move to C_i before the clock x_i proceeds to **const**, and in doing so, reset the clock x_i (i.e. $\mathsf{x}_i := 0$) and assign v to its own process number (i.e. $\mathsf{v} := \mathsf{i}$). From C_i, it can move to the critical section CS_i if v is still equal to its process number (i.e. $\mathsf{v} = \mathsf{i}$) when the clock value of x_i is larger than **const**.

Intuitively, the protocol behaves as follows: The constraints on the shared variable v

[4] In figures, we adopt the convention that when a transition is not labelled with a timing constraint, it means implicitly that the constraint is tt, that is, the transition can be taken at any time. We shall also adopt the convention that when a transition is not labelled with an action, it means that the transition is an internal one, that is, labelled with τ.

$$\begin{array}{rcl}
P_i & \overset{def}{=} & A_i \\
A_i & \overset{def}{=} & (\{v = 0\}, \tau, \{x_i\}).B_i \\
B_i & \overset{def}{=} & (\{x_i < \mathbf{const}\}, \tau, \{v := i, x_i\}).C_i \\
C_i & \overset{def}{=} & (\{v = i, x_i > \mathbf{const}\}, \tau, \{\}).CS_i \\
CS_i & \overset{def}{=} & \mathsf{nil}
\end{array}$$

Figure 4. The Formal Description of Fischer's Protocol.

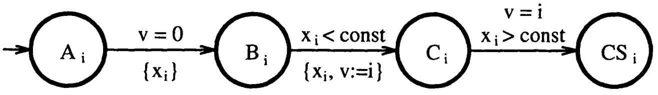


Figure 5. Fischer's Mutual Exclusion Protocol

ensure that a process must reach B–location before any process reach C–location; otherwise, it will never move from A-location to B-location. The timing constraints on the clocks ensure that all processes in C–location must wait until all processes in B–location reach C–location. The last process that reached C–location and set v to its own process number gets the right to enter its critical section. In fact, the protocol will guarantee mutual exclusion for any non–zero constant **const**.

We need to verify that the mutual exclusion property is satisfied, i.e. there will never be more than one process which may reach the critical section, CS_i. The requirement can be formalized as follows: The concurrent system, with an initial state where the control–state is $A_1|\ldots|A_n$ and arbitrary variable assignment, will never reach a state where the control–state is in the form

$$S_1|\ldots|CS_k|\ldots|CS_l|\ldots|S_n$$

for some $k, l \leq n$ and $S_i \in \{A_i, B_i, C_i, CS_i\}$.

We have used our tool and verified a system consisting of 10 processes and **const** $= 1$, which satisfies the property. We are in progress to extend the tool to treat the number of processes as a variable and verify that the property is satisfied by systems with arbitrary number of processes.

4.2. A Simple Railway Control System

We consider a railway control system to automatically control trains passing a critical point such as a bridge. The idea is to use a computer to guide trains from several tracks crossing a single bridge instead of building many bridges. Obviously, a safety–property of such a system is to avoid the situation where more than one train are crossing the bridge at the same time.

Assume that the whole system consists of n trains and a simple controller. We model the system by the following process:

$$(\mathsf{C}|\mathsf{Train_1}|\ldots|\mathsf{Train_n})\backslash\mathsf{A}$$

where $\mathsf{Train_i}$ describe the behavior of trains, C describes the behavior of the controller, and $\mathsf{A} = \{\mathsf{appr_i}, \mathsf{stop_i}, \mathsf{leave_i}, \mathsf{go_i}\}$ is the set of internal channel names (or signals) between the trains and the controller.

To describe timing constraints, we use x and $\mathsf{x_i}$ to model the local time of the controller and the trains respectively. The controller uses a list L for the trains waiting to cross the bridge. The formal descriptions of $\mathsf{Train_i}$'s and C are given in Fig. 6 and illustrated in Fig. 7.

$$\begin{array}{rcl}
Train_i & \stackrel{def}{=} & Safe_i \\
Safe_i & \stackrel{def}{=} & (\{\mathrm{tt}\}, appr_i!, \{x_i\}).Appr_i \\
Appr_i & \stackrel{def}{=} & (\{x_i \geq 0 \land x_i \leq 10\}, stop_i?, \{x_i\}).Slow_i \\
 & & +(\{x_i \geq 11 \land x_i \leq 20\}, \tau, \{x_i\}).Cross_i \\
Cross_i & \stackrel{def}{=} & (\{x_i \geq 3 \land x_i \leq 5\}, leave_i!, \{x_i\}).Safe_i \\
Slow_i & \stackrel{def}{=} & (\{x_i \geq 5 \land x_i \leq 7\}, \tau, \{x_i\}).Stop_i \\
Stop_i & \stackrel{def}{=} & (\{\mathrm{tt}\}, go_i?, \{x_i\}).Start_i \\
Start_i & \stackrel{def}{=} & (\{x_i \geq 7 \land x_i \leq 15\}, \tau, \{x_i\}).Cross_i \\
 & & \\
C & \stackrel{def}{=} & Occ_1 \\
Occ_1 & \stackrel{def}{=} & (\{\mathrm{tt}\}, leave_i?, \{L := L - i\}).Free + (\{\mathrm{tt}\}, appr_i?, \{n := i, x\}).Occ_2 \\
Occ_2 & \stackrel{def}{=} & (\{x < 10\}, stop_n!, \{L := L :: n\}).Occ_1 \\
Free & \stackrel{def}{=} & (\{L = empty\}, appr_i?, \{L := [i]\}).Occ_1 \\
 & & +(\{L \neq empty\}, go_i!, \{i := hd(L)\}).Occ_1
\end{array}$$

Figure 6. The Formal Description of the Railway Control System.

Intuitively, when a train, $\mathsf{Train_i}$, approaches the bridge it sends a signal to the controller within a certain distance. If the bridge is occupied the controller sends a stop signal $\mathsf{stop_i}$ within 10 time units to prevent the train from entering the bridge. Otherwise, if the approaching train does not receive a stop signal within 10 time units, it will start to cross the bridge within 20 time units (but it will take at least 11 time units for a train to enter the bridge). The crossing train is assumed to leave the bridge within 3 to 5 time units; a stopped train will slow down and eventually stop after some delay. When the bridge is

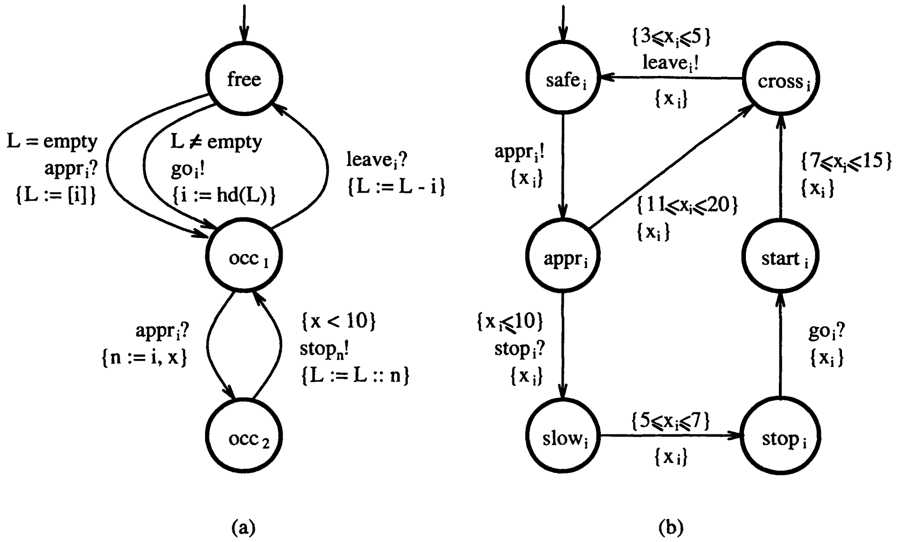


Figure 7. (a) – Controller, (b) – Train

free again and the controller signals (by sending go_i) the first train in the waiting list to cross.

Assume that the system is started with the following control-state:

$$(\mathsf{Free}|\mathsf{Safe}_1|\ldots|\mathsf{Safe}_n)\backslash\{\mathsf{appr}_i, \mathsf{stop}_i, \mathsf{leave}_i, \mathsf{go}_i\}$$

and all clocks are initialized to 0.

We need to guarantee that the system will never reach a control-state where two trains are in location Cross (the clocks may have any values). That is, a state in the form:

$$(\mathsf{S}_i|\mathsf{T}_1|\ldots|\mathsf{Cross}_k|\ldots|\mathsf{Cross}_l|\ldots|\mathsf{T}_n)\backslash\{\mathsf{appr}_i, \mathsf{stop}_i, \mathsf{leave}_i, \mathsf{go}_i\}$$

for some $\mathsf{k}, \mathsf{l} \leq \mathsf{n}$, $\mathsf{S}_i \in \{\mathsf{Free}, \mathsf{Occ}_1, \mathsf{Occ}_2\}$ and $\mathsf{T}_i \in \{\mathsf{Safe}_i, \mathsf{Appr}_i, \mathsf{Slow}_i, \mathsf{Stop}_i, \mathsf{Start}_i\}$.

We have verified a system consisting of 6 trains by our tool, which satisfies the safety–requirement. As in the previous example (Fisher's protocol), we can only check a system with a fixed number of trains. We hope to extend our system to deal with any number of trains.

5. CONCLUSION

The first contribution of this paper is an algebra of processes with clocks, which extends timed automata with algebraic operators. The algebra may serve as a formal description language for real–time communicating systems. In particular, a parallel composition operator is introduced for timed automata to model communication and concurrency, which can be used to construct complex system descriptions in terms of component descriptions.

The second contribution of this paper is a reachability analysis algorithm for the description language, based on constraint–solving techniques. The algorithm is proved to be sound (i.e. always provides the right answer) and complete (i.e. always terminates with an answer). It has been implemented as an automatic verification tool, for verifying safety–properties of real–time communicating systems, based on an existing constraint–solver. Several examples have been used to test the tool. In particular, we have studied and verified Fisher's mutual exclusion protocol and a railway controller using our tool.

There have been many proposals for verifying timed systems e.g. [2, 24, 1, 7, 14, 17]. However, most of them are intended to construct the whole reachability graph of a system or to obtain more efficient model–checking algorithms with respect to a real–time temporal logic, or to check equivalences between abstract specifications. We believe in that the goal of verifying real–time systems, in particular safety–critical systems is to check simple logical properties, which can be done without constructing the whole reachability graph or the full power of model–checking. We are of the opinion that our approach is simpler as it is based directly on constraint–solving techniques and can be fairly efficient in verifying systems consisting of many components as it avoids to explore the whole state–space.

We are in progress to extend our tool to deal with more general types of variables such as lists, in addition to clock variables. In particular, we will treat the number of components in a concurrent system as a parameter (i.e. an ordinary variable) in order to verify systems with many similar components such as the trains in the railway controller and the processes in Fisher's protocol in a more efficient way.

Acknowledgements: We would like to thank Parosh Abdula and Bengt Jonsson for discussions, and the anonymous referees for comments and suggestions on the paper. We would also like to thank Bengt Asker and Ulf Olsson from the steering group of the Swedish Board for Technical Development (NUTEK) for supervising the project.

REFERENCES

1. Rajeev Alur, Costas Courcoubetis, and David Dill. Model–checking for real–time systems. In *Proceedings of the Fifth IEEE Symposium on Logic in Computer Science*, 1990.
2. R. Alur, C Courcoubetis, N. Halbwachs, D. Dill, H. Wong–Toi. Minimization of Timed Transition Systems. CONCUR92, LNCS 630, 1992.
3. Rajeev Alur and David Dill. Automata for modelling real–time systems. In *Automata, Languages and Programming: Proceedings of the 17th ICALP*, LNCS 443. Springer-Verlag, 1990.
4. Martin Abadi and Leslie Lamport. An Old-Fashioned Recipe for Real Time. *Lecture Notes in Computer Science*, volume 600, Springer-Verlag, 1993.
5. J.C.M. Baeten and J.A. Bergstra. Real time process algebra. Technical Report P8916, University of Amsterdam, 1989.
6. B. Berhomieu and M. Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. In *IEEE trans. on Software Engineering*, pages 259–273, Vol 17, March 1991.
7. Karlis Cerans, Jens Chr. Godskesen and Kim G. Larsen. Time Modal Specification – Theory and Tools. *Proceedings of the 5th Int. Conf. on Computer Aided Verification*, 1993.
8. M. Diaz. Modeling and analysis of communication and cooperation protocols using Petri net

based models. *Computer Networks*, Dec. 1982.

9. Jim Davis and Steve Schneider. An introduction to timed CSP. Technical Report PRG–75, Oxford University Computing Laboratory, 1989.
10. Willem Jan Fokkink and Steven Klusener. Real time algebra with prefixed integration. Technical report, CWI, Amsterdam, 1991.
11. Jens Chr. Godskesen and Kim G. Larsen. Real-time calculi and expansion theorems. In *Twelfth Conference on the FST and TCS*, Lecture Notes in Computer Science. Springer-Verlag, December 1992.
12. Nicolas Halbwachs. Delay Analysis in Synchronous Programs. In the *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.
13. Uno Holmer, Kim Larsen, and Yi Wang. Deciding properties of regular timed processes. In Kim G. Larsen and Arne Skou, editors, *Proceedings of the Third Workshop on Computer Aided Verification,*, volume 575 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.
14. T. Henzinger, X. Nicollin, J. Sifakis, and J. Yovine. Symbolic Model Checking for Real–Time Systems. Proceedings of the 7th IEEE Symposium on Logic in Computer Science, 1992.
15. Matthew Hennessy and Tim Regan. A process algebra for timed systems. Technical Report 5/91, University of Sussex, 1991.
16. P.C. Kanellakis and S.A. Smolka, CCS Expressions, finite state processes, and three problems of equivalence. Information and Control Vol 86, 1990.
17. Kim G. Larsen and Wang Yi, Time Abstracted Bisimulation: Implicit Specification and Decidability. In the proceedings of MFPS93 , New Oleans, USA, 1993. Lecture Notes in Computer Science No. 802, 1994.
18. Leslie Lamport. A fast mutual exclusion algorithm. *ACM Trans. on Computer System*, pages 1–11, volume 5(1), February 1987.
19. Robin Milner. *Communication and Concurrency.* Series in Computer Science. Prentice–Hall International, 1989.
20. Faron Moller and Chris Tofts. A temporal calculus of communicating systems. In *CONCUR'90*, volume 458 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.
21. Martin Nilsson. Piecewise Linear Constraints and Entailment Technical report, Swedish Institute of Computer Science, August 1993.
22. X. Nicollin, J.-L. Richier, Joseph Sifakis, and J. Yovine. ATP: an algebra for timed processes. In *Proceedings of the IFIP TC 2 Working Conference on Programming Concepts and Methods*, Sea of Gallilee, Israel, April 1990.
23. Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. From ATP to timed graphs and hybrid systems. In *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.
24. Kārlis Čerāns. Decidability of bisimulation equivalences for processes with parallel timers. In the Proceedings of CAV'92, 1992.
25. N. Shankar. Verification of Real–Time Systems Using PVS. *Proceedings of the 5th Int. Conf. on Computer Aided Verification*, 1993.
26. Frits Vaandrager and Nancy Lynch. Action Transducers and Timed Automata. In *CONCUR '92*, volume 630 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.
27. Yi Wang. Real–time behaviour of asynchronous agents. In *CONCUR '90*, volume 458 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.
28. Yi Wang. *A Calculus of Real Time Systems.* PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 1991.
29. Yi Wang. CCS + time = an interleaving model for real time systems. In *ICALP '91, LNCS 510.* Springer-Verlag, 1991.