On Checking Parallel Real-Time Systems for Linear Duration Properties

Zhao Jianhua* and Dang Van Hung**

The United Nations University
International Institute for Software Technology, P.O.Box 3058, Macau
email: {zjh, dvh}@iist.unu.edu

Abstract. The major problem of checking a parallel composition of real-time systems for a real-time property is the explosion of untimed states and time regions. To attack this problem, one can use bisimulation equivalence w.r.t. the property to be checked to minimise the system state space. In this paper, we define such equivalence for the integrated linear duration properties of real-time automaton networks with shared variables. To avoid exhaustive state space exploration, we define a compatibility relation, which is a one-direction simulation relation between configurations. Based on this technique, we develop an algorithm for checking a real-time automaton network with shared variables w.r.t. a linear duration property. Our algorithm can avoid exhaustive state space exploration significantly when applied to Fischer's mutual exclusion protocol.

1 Introduction

In the last few years, some verification tools have been developed for timed systems [10, 11, 3]. The verification engines of most of these tools are based on reachability analysis of timed automata following the pioneering work of Alur and Dill [2].

A series of techniques have been developed to attack the potential state explosions arising not only from the control-structure but also from the region space. Efficient data structures and algorithms have been sought to represent and manipulate timing constraints over clock variables [9] and to avoid exhaustive state space exploration.

In this paper, we show how to apply and to improve these techniques for a class of model-checking problems for real-time systems. Namely, we use linear duration properties (a kind of simple Duration Calculus [4] formulas) as system specifications and real-time automaton networks with shared variables as real-time system models. Checking linear duration properties is difficult because

* On leave from Department of Computer Science, Nanjing University, Nanjing 210093, P.R. China. email: seg@nju.edu.cn. Partly supported by National NSF.

** On leave from the Institute of Information Technology, Nghia Do, Tu Liem, Hanoi, Vietnam.

the durations of states are defined on the history of the systems. Actually, the problem has been solved for real-time automata (real-time automaton networks of single process) in [5] by using linear programming technique. In that paper, an algorithm was given for reducing the problem to a finite set of very simple linear programming problems. This technique has been generalised for verifying a subset of timed automata in [6], and for verifying a parallel composition of real-time automata in [12]. This technique has also been generalised for verifying a subset of hybrid automata in [7]. For timed automata, it has been shown in [14] that the problem can be solved but ones have to use the mixed integer and linear programming techniques, which are very complicated. In [1], an algorithm has been developed for checking duration-bounded reachability which asks whether there is a run of an automaton from a start state to a target state along which the accumulated duration of system states satisfies a constraint. In that paper, the coefficients corresponding to the state durations are restricted to non-negative integers only.

In this paper, we consider the problem for a network of real-time automata with shared variables. We show that a linear duration property is satisfied by a real-time automaton network iff it is satisfied by all the integer behaviours of the network. Then, we define a so-called compatibility relation between configurations and apply the technique in [13] for that relation to develop a model-checking algorithm for real-time automaton networks that can avoid exhaustive state space exploration in some cases. We apply our technique to Fischer's mutual exclusion protocol and find that this technique results in a significant space-reduction.

The paper is organised as follows. In Section 2 real-time automaton networks with shared variables are formally defined. Linear duration properties and their satisfaction by a network are described in Section 3. Our basic idea and the algorithm are given in Section 4. Section 5 is devoted to the verification of Fischer's mutual exclusion protocol using our algorithm. The last section is the conclusion of the paper.

2 Real-Time Automaton Networks with Shared Variables

In this section, we give a formal definition of real-time automaton networks with shared variables and their behaviours. Let *Nat* denote the set of nonnegative integers, and *Intv* denote the set of intervals on *Nat*.

Definition 1. *A real-time automaton [5] is a pair* $\langle \mathbf{A}, \Gamma \rangle$*, where*

1. $\mathbf{A} = \langle \mathcal{S}, s_0, \mathcal{E} \rangle$ *is a conventional automaton,*
2. Γ *is a mapping from* $\mathcal{E}$ *to* $Intv$*;* Γ *assigns to each transition* $e \in \mathcal{E}$ *a time interval* $[a_e, b_e]$ *or* $[a_e, \infty)$*, where* a_e, b_e *are integers and* $0 \leq a_e \leq b_e$*, which express the delay* a_e *and the deadline* b_e *or* ∞ *of the transition* e*.*

For a given state $s \in \mathcal{S}$, we use U_s to denote the value $\max\{b_e \mid (\overleftarrow{e} = s)\}$. $\Gamma(e)$ is denoted by $[a_e, b_e]$, or $[a_e, \infty)$ when $b_e = \infty$. The empty word ϵ is considered as a special transition of all the real-time automata in this paper.

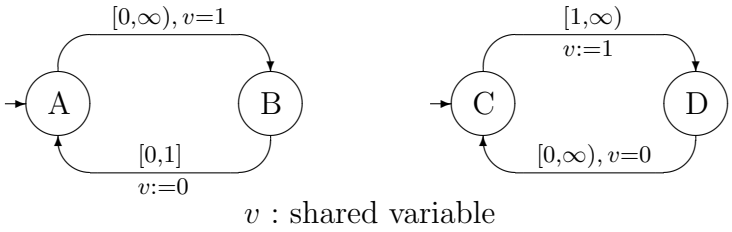

Fig. 1. A real-time automaton network with shared variables

Definition 2. *A real-time automaton network with shared variables (real-time automaton network for short) is a tuple* $\mathcal{N}=\langle \mathcal{P}, \mathcal{V}, \mathcal{G}, \mathcal{R}\rangle$*, where*

1. $\mathcal{P} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$ *is a finite set of real-time automata; for* $i = 1, 2, \ldots, n$ *let* $\mathcal{A}_i = \langle \mathbf{A}_i, \Gamma_i\rangle$ *and* $\mathbf{A}_i = \langle \mathcal{S}_i, q_0^i, \mathcal{E}_i\rangle$*,*
2. $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$ *is a finite set of shared variables,*
3. $\mathcal{G}$ *is a mapping from* $(\mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_n)$ *to* $VarGuard$ *(defined below)which assigns to each transition of the automata in* $\mathcal{P}$ *a boolean expression on the values of the shared variables in* $\mathcal{V}$*,*
4. $\mathcal{R} \in (\mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_n) \to (\mathcal{V} \to Nat)$*;* $\mathcal{R}$ *expresses the values of shared variables reset by the transitions; when a transition* e *takes place, each shared variable* v *in the domain of* $\mathcal{R}(e)$ *is assigned the value* $\mathcal{R}(e)(v)$*.*

$VarGuard$ ranged over by ψ is generated by the following grammar:

$$\psi \triangleq \ true \mid v = c \mid \psi_1 \wedge \psi_2 \,,$$

where v stands for a shared variable, and c stands for an integer constant. A guard $\psi \in VarGuard$ is evaluated to be true on a valuation V iff for each equation $v = c$ appeared in ψ, the value of v is c.

For a vector $\overline{x}$, let x_i denote the ith element of $\overline{x}$, and let $\overline{x}[x_i'/i]$ denote the vector obtained from $\overline{x}$ by replacing the ith element with x_i'.

Definition 3. *Let* $\mathcal{N} = \langle \mathcal{P}, \mathcal{V}, \mathcal{G}, \mathcal{R}\rangle$ *be a real-time automaton network. An untimed state* $\overline{s}$ *of* $\mathcal{N}$ *is an* n*-dimensional vector of local states (*s_i *is a state of* $\mathcal{A}_i$ *for* $1 \le i \le n$*). A configuration of* $\mathcal{N}$ *is a tuple* $\langle \overline{s}, \overline{t}, V\rangle$*, where* $\overline{s}$ *is an untimed state of* $\mathcal{N}$*,* $\overline{t}$ *is an* n*-dimensional vector of nonnegative reals. The value* t_i *expresses how long the automaton* $\mathcal{A}_i$ *has been staying at* s_i *and* V *is a valuation of the shared variables.*

The transition system of a real-time automaton network is described as follows.

Definition 4. *Let* $\mathcal{N} = \langle \mathcal{P}, \mathcal{V}, \mathcal{G}, \mathcal{R}\rangle$ *be a real-time automaton network,* $\langle \overline{s}, \overline{t}, V\rangle$ *and* $\langle \overline{s}', \overline{t}', V'\rangle$ *be its configurations. We define:*

1. $\langle \overline{s}, \overline{t}, V\rangle \xrightarrow{e^i} \langle \overline{s}', \overline{t}', V'\rangle$ *(*e^i *is a transition of* $\mathcal{A}_i$*) iff*
   (a) $(s_i = \overleftarrow{e^i}) \wedge (\overline{s}' = \overline{s}[\overrightarrow{e^i}/i])$*, and*

(b) $t_i \in [a_{e^i}, b_{e^i}] \wedge \vec{t}' = \overline{t}[0/i]$, *and*
(c) $\mathcal{G}(e^i)(V) \wedge V' = V \dagger \mathcal{R}(e^i)$.

2. $\langle \overline{s}, \overline{t}, V \rangle \xrightarrow{\epsilon,d} \langle \overline{s}, \overline{t}', V \rangle \;\; (d \geq 0)$ *iff* $(t'_i = t_i + d) \wedge (t'_i \leq U_{s_i})$ *for* $1 \leq i \leq n$.

Given two configurations $\mathcal{C}_1$ *and* $\mathcal{C}_2$, *we write* $\mathcal{C}_1 \xrightarrow{e,d} \mathcal{C}_2$ *iff there exists a configuration* $\mathcal{C}'$ *such that* $\mathcal{C}_1 \xrightarrow{\epsilon,d} \mathcal{C}'$ *and* $\mathcal{C}' \xrightarrow{e} \mathcal{C}_2$. *For a configuration* $\mathcal{C}$, *for a transition* e *(can be* ϵ*) and for an integer* d, *there is at most one configuration* $\mathcal{C}'$ *satisfying* $\mathcal{C} \xrightarrow{e,d} \mathcal{C}'$. *We denote this configuration by* $Succ(\mathcal{C}, e, d)$ *in the case it exists.*

An execution α of the network $\mathcal{N}$ is defined as

$$\alpha = \mathcal{C}_0 \xrightarrow{e_1,d_1} \mathcal{C}_1 \xrightarrow{e_2,d_2} \ldots \xrightarrow{e_m,d_m} \mathcal{C}_m \xrightarrow{\epsilon,d_{m+1}} \mathcal{C}_{m+1} \,,$$

where $\mathcal{C}_0$ is the initial configuration of $\mathcal{N}$. We call the timed sequence $\tau = (e_1, t_1)\hat{}(e_2, t_2)\hat{} \ldots \hat{}(e_m, t_m)\hat{}(\epsilon, t_{m+1})$ the behaviour corresponding to the execution α and $\mathcal{C}_{m+1}$ the final configuration of α, where for each $1 \leq i \leq m+1$, $t_i = \sum_{j=1}^{i} d_j$. A configuration $\langle \overline{s}, \overline{t}, V \rangle$ is called an integer configuration iff for each i $(1 \leq i \leq n)$ t_i is an integer. The execution α is said to be an integer execution iff all $\mathcal{C}_i$'s $(0 \leq i \leq m+1)$ are integer configurations.

3 Linear Duration Properties

Let, from now on in this paper, $\mathcal{N}$ be a real-time automaton network as defined in Definition 2. The linear duration properties (LDPs) are simple Duration Calculus formulas used to express requirements of real-time systems. An LDP is a linear inequality of the state-durations of the form $\Psi \leq M$, where $\Psi = \sum_{i=1}^{k} c_i \int S_i$, and for each $i\,(1 \leq i \leq k)$, S_i is a predicate on untimed states of $\mathcal{N}$, c_i and M are real numbers (c_i is called the coefficient corresponding to S_i and $\int S_i$ the duration of S_i).

Let $\alpha = \mathcal{C}_0 \xrightarrow{e_1,d_1} \mathcal{C}_1 \xrightarrow{e_2,d_2} \ldots \xrightarrow{e_m,d_m} \mathcal{C}_m \xrightarrow{\epsilon,d_{m+1}} \mathcal{C}_{m+1}$ be an execution of $\mathcal{N}$. The duration $\int S_i$ of system state S_i over the behaviour α is defined by

$$\int S_i(\alpha) = \sum_{u \in \beta_i} d_{u+1} \,,$$

where $\beta_i = \{u | (0 \leq u \leq m) \wedge (\overline{s}_u \text{ is the untimed state of } \mathcal{C}_u) \wedge (\overline{s}_u \Rightarrow S_i)\}$. Thus, given an LDP $\Psi \leq M$, the value of Ψ evaluated over α, denoted $\Psi(\alpha)$, is calculated as

$$\Psi(\alpha) = \sum_{i=1}^{k} c_i \int S_i(\alpha) = \sum_{i=1}^{k} c_i (\sum_{u \in \beta_i} d_{u+1}) = \sum_{j=0}^{m} C_{\overline{s}_j} d_{j+1} \,,$$

where $C_{\overline{s}_j} = \sum_{i \in \{i | \overline{s} \Rightarrow S_j\}} c_i$, and $\overline{s}_j$ $(0 \leq j \leq m)$ is the untimed state of $\mathcal{C}_j$.

Definition 5. *An LDP* $\Psi \leq M$ *is satisfied by an execution* α *iff* $\Psi(\alpha) \leq M$, *and is satisfied by* $\mathcal{N}$, *denoted* $\mathcal{N} \models \Psi \leq M$, *iff it is satisfied by all executions of* $\mathcal{N}$.

For example, the property that the time the first automaton in Figure 1 stays at the state A is longer than the time it stays at the state B can be expressed by the LDP $\int \mathbf{at}_B - \int \mathbf{at}_A \leq 0$ where $\mathbf{at}_A$ and $\mathbf{at}_B$ are predicates on untimed states such that $\langle A, C\rangle \Rightarrow \mathbf{at}_A$, $\langle A, D\rangle \Rightarrow \mathbf{at}_A$, $\langle B, C\rangle \Rightarrow \mathbf{at}_B$, $\langle B, D\rangle \Rightarrow \mathbf{at}_B$.

4 The Algorithm

4.1 Some Properties of Real-Time Automaton Networks and LDPs

Let, in this section, $\Psi \leq M$ be an LDP. The observation leading to our algorithm is formulated in the following lemma.

Lemma 1. *For any execution α of $\mathcal{N}$, there is an integer execution α' of $\mathcal{N}$ such that $\Psi(\alpha) \leq \Psi(\alpha')$.*

From Lemma 1, the network $\mathcal{N}$ satisfies the LDP $\Psi \leq M$ iff all the integer behaviours of $\mathcal{N}$ satisfy the LDP. In fact, we do not have to check all the integer executions. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be configurations for which each behaviour starting from $\mathcal{C}_1$ can be simulated by a behaviour starting from $\mathcal{C}_2$. Let α and α' be integer executions with $\mathcal{C}_1$ and $\mathcal{C}_2$ as their final configurations. If $\Psi(\alpha') \geq \Psi(\alpha)$ holds, We do not need to consider the right extensions of α. In order to formalise this idea we introduce the compatibility relation between configurations as follows.

Definition 6. *Let $\mathcal{C}_1 = \langle \overline{s}, \overline{t}, V\rangle$ and $\mathcal{C}_2 = \langle \overline{s}, \overline{t}', V\rangle$ be configurations of $\mathcal{N}$. $\mathcal{C}_1$ is said to be compatible with $\mathcal{C}_2$, denoted by $\mathcal{C}_1 \preceq \mathcal{C}_2$, iff for $1 \leq i \leq n$ the following holds,*

$$\begin{array}{l}(t_i = t'_i) \vee (t_i \geq RgBnd_{s_i} \wedge t'_i \geq RgBnd_{s_i}) \vee \\ (LowBnd_{s_i} \leq t_i < t'_i) \vee (t_i > t'_i \wedge \forall e \bullet (\overleftarrow{e} = s_i \Rightarrow b_e = \infty))\,,\end{array}$$

where $RgBnd_{s_i} = \max(\{a_e | \overleftarrow{e} = s_i\} \cup \{b_e + 1 | (\overleftarrow{e} = s_i) \wedge (b_e < \infty)\})$ and $LowBnd_{s_i} = \max(\{a_e | \overleftarrow{e} = s_i\})$.

Since for each local state s, $\forall e \bullet (\overleftarrow{e} = s \Rightarrow b_e = \infty)$ means $RgBnd_s = LowBnd_s$, the compatibility relation is transitive and reflexive.

Lemma 2. *Let $\mathcal{C}_1$, $\mathcal{C}_2$ be integer configurations of $\mathcal{N}$. $\mathcal{C}_1 \preceq \mathcal{C}_2$ implies that for arbitrary transition e (e can be ϵ), for arbitrary integer d and for arbitrary configuration $\mathcal{C}'_2$, if $\mathcal{C}_2 \xrightarrow{e,d} \mathcal{C}'_2$ then for some configuration $\mathcal{C}'_1$, it holds $\mathcal{C}_1 \xrightarrow{e,d} \mathcal{C}'_1 \wedge \mathcal{C}'_1 \preceq \mathcal{C}'_2$.*

This lemma means that given configurations $\mathcal{C}_1$ and $\mathcal{C}_2$, if $\mathcal{C}_1 \preceq \mathcal{C}_2$, then each behaviour starting from $\mathcal{C}_2$ can be simulated by a behaviour starting from $\mathcal{C}_1$.

Definition 7. *Integer configurations $\mathcal{C}_1$ and $\mathcal{C}_2$ are equivalent, denoted $\mathcal{C}_1 \equiv \mathcal{C}_2$, iff $(\mathcal{C}_1 \preceq \mathcal{C}_2) \wedge (\mathcal{C}_2 \preceq \mathcal{C}_1)$.*

It is obvious that $\equiv$ is an equivalence relation, and furthermore, of finite index. Indeed, let $\mathcal{R}_\mathcal{N}$ be $\{[\mathcal{C}] \,|\, (\mathcal{C}$ is a reachable integer configuration of $\mathcal{N})\}$, where $[\mathcal{C}]$ denote the equivalence class of $\equiv$ containing $\mathcal{C}$.

Lemma 3. *$\mathcal{R}_\mathcal{N}$ is a finite set.*

4.2 Weighted Directed Graphs

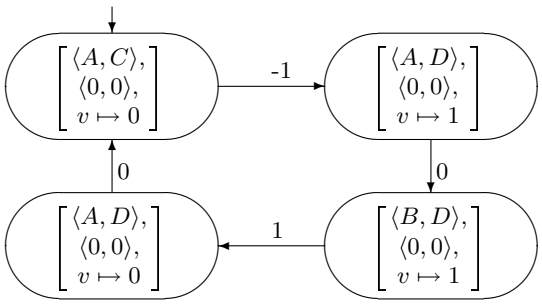

Fig. 2. A weighted directed graph

A *weighted directed graph* of the real-time automaton network $\mathcal{N}$ corresponding to the LDP $\Psi \leq M$ is defined as follows. The set of nodes is $\mathcal{R}_{\mathcal{N}}$. There is an arc a with weight D_a from a node n to a node n' iff there exist a transition e and configurations $\mathcal{C} \in n$ and $\mathcal{C}' \in n'$ for which $\mathcal{C} \xrightarrow{e,d} \mathcal{C}'$ for some d and $D_a = \max\{C_{\overline{s}}d \mid \exists \mathcal{C}_1, \mathcal{C}_1' \bullet (\mathcal{C}_1 \in n \wedge \mathcal{C}_1' \in n' \wedge \mathcal{C}_1 \xrightarrow{e,d} \mathcal{C}_1')\}$, where $\overline{s}$ is the untimed state of $\mathcal{C}$. From definition, D_a may be ∞, and if $D_a < \infty$, there is an integer d such that $C_{\overline{s}}d = D_a$. (Figure 2 shows such a graph for the network in Figure 1)

The initial node of the graph is the node containing the initial configuration of the network. From the definition of $\equiv$, there is an algorithm to enumerate all the successive nodes of any node n.

For a configuration $\mathcal{C} = \langle \overline{s}, \overline{t}, V \rangle$, let $L_{\mathcal{C}}$ denote the value $\min(\{U_{s_i} - t_i | 1 \leq i \leq n\})$. $L_{\mathcal{C}}$ is the longest time the network can stay at $\overline{s}$ once it reaches the configuration $\mathcal{C}$. From the definition of the weighted graph, we have the following lemma.

Lemma 4. *Let* $p = [\mathcal{C}_0] \xrightarrow{a_1} [\mathcal{C}_1] \xrightarrow{a_2} \ldots \xrightarrow{a_m} [\mathcal{C}_m]$ *be an arbitrary path of the graph. Let* $\widehat{\Psi}(p)$ *denote the length of* p*, i.e.* $\widehat{\Psi}(p) = \sum_{i=1}^{m} D_{a_i}$*. Let* $\alpha = \mathcal{C}_0' \xrightarrow{e_1,d_1} \mathcal{C}_1' \xrightarrow{e_2,d_2} \ldots \xrightarrow{e_m,d_m} \mathcal{C}_m' \xrightarrow{\epsilon,d_{m+1}} \mathcal{C}_{m+1}'$ *be an execution of* $\mathcal{N}$ *satisfying that for* $0 \leq i \leq m$, $\mathcal{C}_i' \in [\mathcal{C}_i]$*. Then*

1. $\Psi(\alpha) \leq \widehat{\Psi}(p) + \max(0, C_{\overline{s}}L_{\mathcal{C}_m})$.
2. *If* $\widehat{\Psi}(p) + \max(0, C_{\overline{s}}L_{\mathcal{C}_m}) = \infty$*, then for any constant* M*, there is an execution* α' *satisfying* $\Psi(\alpha') > M$*. Else there is an execution* α'' *satisfying* $\Psi(\alpha'') = \widehat{\Psi}(p) + \max(0, C_{\overline{s}}L_{\mathcal{C}_m})$.

4.3 Algorithm

From Lemma 4, for checking whether the network $\mathcal{N}$ satisfies the LDP $\Psi \leq M$, we can check whether there is a path in the graph the length of which is greater

than M. We can use the depth-first method to generate and check the paths starting from the initial node of the graph. It performs back-tracking once it finds that one of the following two conditions holds (let $[\mathcal{C}]$ be the last node of the path) .

1. No new successive node can be generated.
2. A previously checked path p reaches $[\mathcal{C}']$ and $\mathcal{C}' \preceq \mathcal{C}$, the length of p is greater the current length.

```
G := {⟨[C^0], 0⟩}; p :=≪ [C^0] ≫;
while True do
        flg := NextNode;{append p with a new successive node}
        while flg = False do
        begin
              Delete the last element from p; {back-tracking}
              if (p is empty) return True;
              {No more paths can be generated and checked}
              flg := NextNode;
        end
        C := a configuration in the last node of p;
        if Ψ̂(p) + max(0, C_s̄ L_C) > M, where s̄ is the untimed state of C
              then return False;
        if there is a prefix p' of p such that
          C is compatible with the configuration in the last node of p'
          and Ψ̂(p) > Ψ̂(α')
              then return False;
        if ∃⟨[C'], l'⟩ ∈ G • (C' ⪯ C) ∧ (l' ≥ Ψ̂(p)) then
              delete the last element of p;{ back-tracking}
        else begin
              Add the tuple ⟨[C], Ψ̂(p)⟩ to G;
              Delete all the tuples ⟨[C'], l'⟩ from G satisfying Ψ̂(p) ≥ l' ∧ C ⪯ C'.
        end;{of else}
end;{of while}
```

Fig. 3. The Algorithm

The algorithm for checking whether the network $\mathcal{N}$ satisfies the LDP $\Psi \leq M$ is given in Figure 3. It uses auxiliary variables G and p. The variable G is used to record the nodes reached by previously checked paths together with the length of the longest paths reaching them. The variable p is used to record the currently generated path. In this algorithm $NextNode$ is a procedure used to generate a new successive node of the currently generated path in p. It returns $False$ if no such node can be generated. Otherwise, $NextNode$ generates a new node and appends it to p . Besides, $\mathcal{C}$ is a variable ranging over the set of configurations, flg is a boolean variable, l is a variable of real.

Let p be the currently generated path and $[\mathcal{C}]$ be its last node. Let p' be a prefix of p and the last node of p' is also $[\mathcal{C}]$. If $\widehat{\Psi}(p') < \widehat{\Psi}(p)$, we can repeat the segment of p from the end of p' to the end of p for making the Ψ value unlimitedly large. So the LDP is violated in this case. If $\widehat{\Psi}(p') \geq \widehat{\Psi}(p)$, the tuple $\langle [\mathcal{C}], \widehat{\Psi}(p') \rangle$ must have been stored in G, so the algorithm performs back-tracking. Therefore, the algorithm generates only paths with at most one circle, there are only finite number of such paths. The algorithm terminates.

The body of the algorithm is written in a Pascal-like language, where $[\mathcal{C}^0]$ denotes the initial node of the graph.

5 Fischer's Mutual Exclusion Protocol: A Case Study

We have implemented the algorithm presented in the previous section in the C language and tried the program with Fischer's mutual exclusion protocol. To our knowledge, Fischer's protocol is well studied and the size of the example can be scaled up by increasing the number of processes in the protocol.

The protocol is to guarantee mutual exclusion in a concurrent system consisting of a number of processes, using clock constraints and a shared variable. The protocol can be modelled as a real-time automaton network with a shared variable v. The ith process has its own identifier i and can be modelled as a real-time automaton shown in Figure 4.

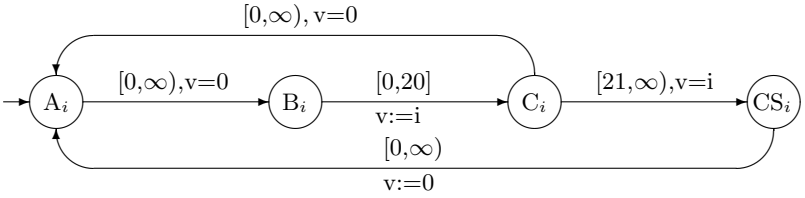

Fig. 4. The real-time automaton of the ith process

We need to verify formally that at any moment at most one process can be in its critical section. This requirement can be formalised as the following linear duration property:

$$\int Error \leq 0$$

where $Error$ is a predicate on the untimed states, and $\overline{s} \Rightarrow Error$ iff there exist j and i such that $j \neq i$ and $s_i = CS_i \wedge s_j = CS_j$.

Running on a SparcStation 5 with 48MB of primary memory, our program has verified the mutual exclusion property of Fischer's protocol for the cases $n = 2, 3, \ldots, 11$. The space requirements for these cases are listed in Table 1, these requirements are represented in term of the number of nodes generated and stored by the algorithm. It can be seen that for $k \leq 10$ the space requirement

n =	2	3	4	5	6	7	8	9	10	11
nodes	18	65	220	727	2378	7737	25080	81035	260998	837949

Table 1. Space requirements on checking Fischer's protocol using our algorithm

n =	2	3	4	5
space	16	237	3528	59715

Table 2. Space requirements on checking Fischer's protocol using UPPAAL

for the case $n = k + 1$ is approximately 4 times of the one for the case $n = k$. The run time requirement for the case $n = 11$ is about 15 minutes. The well-known verification tool UPPAAL can only check the Fischer's protocol with up to 8 processes by state space exploration [8]. The performance statistics data in Table 2 [9] shows that adding a new process makes the space requirement of UPPAAL rise to more than 12 times of the original one. The similar results have been reported for the tool HyTech.

6 Conclusion

In this paper, we have presented an algorithm to check the validity of a real-time automaton network with respect to a linear duration property, which is a linear inequality of integrated state-durations. To attack the state space explosion caused by time-constraints, a compatibility relation $\preceq$ over configurations is introduced. We use this technique to reduce the space-complexity of the algorithm. It has been shown that in some cases like the Fischer's mutual exclusion protocol, this technique can reduce the space-complexity significantly.

Although the model we used in this paper is simple, we believe that this idea can be applied to other models such as timed automata and networks of timed automata.

References

1. R. Alur, C. Courcoubetis, and T.A. Henzinger. Computing accumulated delays in real-time systems. In *Proceedings of the Fifth Conference on Computer-Aided Verification*, LNCS 693, pages 181–193. Springer-Verlag, 1993.
2. R. Alur and D. Dill. Automata for Modelling Real-Time Systems. In *Proc. of ICALP'90*, LNCS 443, 1990.
3. C.Daws, A.Olivero, S.Tripakis, and S.Yovine. The tool Kronos. In *Hybrid Systems III, Verification and Control*, number 1066 in Lecture Notes in Computer Science. Springer-Verlag, 1996.
4. Zhou Chaochen, C.A.R. Hoare, and A.P.Ravn. A Calculus of Durations. In *Information Processing Letter 40,5*, pages 269–276, 1991.
5. Zhou Chaochen, Zhang Jingzhong, Yang Lu, and Li Xiaoshan. Linear Duration Invariants. Research Report 11, UNU/IIST, P.O.Box 3058, Macau, July 1993.

Published in: Formal Techniques in Real-Time and Fault-Tolerant systems, LNCS 863, 1994.

6. Li Xuan Dong and Dang Van Hung. Checking Linear Duration invariants by Linear Programming. Research Report 70, UNU/IIST, P.O.Box 3058, Macau, May 1996. Published in Joxan Jaffar and Roland H. C. yap (Eds.), *Concurrency and Palalellism, Programming, Networking, and Securiry* LNCS 1179, Springer-Verlag, Dec 1996, pp. 321–332.
7. Li Xuan Dong, Dang Van Hung, and Zheng Tao. Checking Hybrid Automata for Linear Duration Invariants. Research Report 109, UNU/IIST, P.O.Box 3058, Macau, June 1997. Acepted to present at ASIAN'97, Asian Computing Science Conference, Kathmandu, Nepal, December 9–12, 1997, to apear in LNCS.
8. Kere J. Kristoffersen, Francois Larroussinie, Kim G.Larsen, Paul Pettersson, and Wang Yi. A Compositional Proof of a Real-Time Mutual Exclusion Protocol. In *Proceedings of TAPSOFT'97, the 7th International Joint Conference on the Theory and Practice of Software Development*, pages 14–18, April 1997.
9. Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction. December 1997. Accepted for presentation at the 18th IEEE Real-Time Systems Symposium. San Francisco, California, USA.
10. Kim G Larsen and Paul Pettersson Wang Yi. UPPAAL: Status & Developments. In Orna Grumberg, editor, *Proceedings of the 9th International Conference on Computer-Aided Verification. Haifa, Israel,*, LNCS 1254, pages 456–459. Springer-Verlag, June 1997.
11. T.A.Henzinger, P.-H. Ho, and H. Wong-Toi. A Users Guide to HyTech. Technical report, Department of Computer Science, Cornell University, 1995.
12. Pham Hong Thai and Dang Van Hung. Checking a regular class of Duration Calculus Models for Linear Duration Invariants. Technical Report 118, UNU/IIST, P.O.Box 3058, Macau, July 1997. Presented at the 5th Vietnamese Conference of Mathematics, Hanoi, 17-20 Sep, 1997.
13. Mihalis Yannakakis and David Lee. An Efficient Algorithm for Minimizing Real-Time Transition Systems. *Formal Methods in System Design*, 11(2):113–136, August 1997.
14. Y.Kesten, A.Pnueli, J.Sifakis, and S.Yovine. Integration Graphs: A Class of Decidable Hybrid Systems. In *Hybrid System*, number 736 in LNCS, pages 179–208, 1993.