FORMAL VERIFICATION OF TIMING CONSTRAINED FINITE-STATE SYSTEMS

by

Felice Balarin and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M92/8

27 January 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

FORMAL VERIFICATION OF TIMING CONSTRAINED FINITE-STATE SYSTEMS

by

Felice Balarin and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M92/8

27 January 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

1 Introduction

Recently, Dill [Dil89] and Alur and Dill [AD90] proposed a method for incorporating timing restriction into a model of communicating finite-state systems by introducing the notion of a timed automaton, containing fictitious time-measuring elements called clocks [AD90] or timers [Dil89]. The verification problem is shown to be equivalent to the speed-independent verification problem on an appropriate automaton. The fundamental problem with both approaches is state space explosion, i.e. state space growing exponentially in the number of timers (clocks).

Kurshan [Kur91] suggested to carry out the verification process on timed systems with COSPAN [HK88], a verification system for untimed processes, by relaxing the time constraints, verifying the relaxed system and if the verification is unsuccessful check whether the run that violates the property to be verified is infeasible under the timing constraints. If this is so, Kurshan removes the run and repeats the process. This strategy is appealing but heuristic in nature. There was no proof that the process would eventually converge to provably the correct answer.

In this paper, we introduce the notion of *pauses*, and construct an equivalent (non-pausing) automaton. In contrast to previous approaches, we build an equivalent automaton as a composition of the speed-independent (or unrestricted) automaton and many small automata. This enables us to build simple abstractions of the equivalent automaton that are still detailed enough to eliminate certain undesired behavior. This leads to a verification strategy similar to the heuristic proposed by Kurshan, where a verification process is started with the unrestricted automaton, which is then composed with simple automata imposing timing constraints, but only after the verification has failed, and imposing only those constraints which are violated in the failure report. In the worst case, all constraints must be used, so that the verification is performed with the automaton of full complexity. However, some preliminary experiments have shown that the worst case is not the common case and that this strategy can indeed lead to considerable time and space savings.

The rest of this paper is organized as follows. L-processes [Kur90] are shortly discussed in section 2. In section 3, we introduce the notion of timed L-process, and then we construct the equivalent (not timed) L-process in section 4. In section 5 two main steps of the proposed verification strategy are described: extracting timing violations from the failure report, and imposing that subset of timing constraints to the model of the system. Final remarks are provided in section 6.

2 L-processes

An L-process [Kur90] is an automaton defined over the alphabet $S(L)$, where L is a Boolean algebra with product $*$, sum $+$, complement $\bar{}$, multiplicative identity 1, and additive identity 0. A partial

FORMAL VERIFICATION OF TIMING CONSTRAINED FINITE-STATE SYSTEMS

by

Felice Balarin and Alberto L. Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M92/8

27 January 1992

1 Introduction

Recently, Dill [Dil89] and Alur and Dill [AD90] proposed a method for incorporating timing restriction into a model of communicating finite-state systems by introducing the notion of a timed automaton, containing fictitious time-measuring elements called clocks [AD90] or timers [Dil89]. The verification problem is shown to be equivalent to the speed-independent verification problem on an appropriate automaton. The fundamental problem with both approaches is state space explosion, i.e. state space growing exponentially in the number of timers (clocks).

Kurshan [Kur91] suggested to carry out the verification process on timed systems with COSPAN [HK88], a verification system for untimed processes, by relaxing the time constraints, verifying the relaxed system and if the verification is unsuccessful check whether the run that violates the property to be verified is infeasible under the timing constraints. If this is so, Kurshan removes the run and repeats the process. This strategy is appealing but heuristic in nature. There was no proof that the process would eventually converge to provably the correct answer.

In this paper, we introduce the notion of *pauses*, and construct an equivalent (non-pausing) automaton. In contrast to previous approaches, we build an equivalent automaton as a composition of the speed-independent (or unrestricted) automaton and many small automata. This enables us to build simple abstractions of the equivalent automaton that are still detailed enough to eliminate certain undesired behavior. This leads to a verification strategy similar to the heuristic proposed by Kurshan, where a verification process is started with the unrestricted automaton, which is then composed with simple automata imposing timing constraints, but only after the verification has failed, and imposing only those constraints which are violated in the failure report. In the worst case, all constraints must be used, so that the verification is performed with the automaton of full complexity. However, some preliminary experiments have shown that the worst case is not the common case and that this strategy can indeed lead to considerable time and space savings.

The rest of this paper is organized as follows. L-processes [Kur90] are shortly discussed in section 2. In section 3, we introduce the notion of timed L-process, and then we construct the equivalent (not timed) L-process in section 4. In section 5 two main steps of the proposed verification strategy are described: extracting timing violations from the failure report, and imposing that subset of timing constraints to the model of the system. Final remarks are provided in section 6.

2 L-processes

An L-process [Kur90] is an automaton defined over the alphabet $S(L)$, where L is a Boolean algebra with product $*$, sum $+$, complement $\bar{}$, multiplicative identity 1, and additive identity 0. A partial

Abstract

At high levels of design, the task of formal verification is to prove that the design satisfies some properties. Designs are usually presented in some form of a finite-state system. Verification of many properties can be done without regard to the speed of the components of the system. However, some of the properties can be verified only under certain timing constraints. It has been shown that timing constraints can be expressed in the framework of finite-state systems, at the expense of exponential explosion of number of states. We propose a new verification strategy for timing constrained finite-state systems. The strategy is to start the verification process on the unconstrained system, and then add incrementally timing constraints, but only after the verification process has failed and adding only those constraints that have been violated. The strategy can avoid the state space explosion problem for a class of systems. We use our strategy in the context of *timed L-process*, a new model that we propose for timing constrained systems.

1 Introduction

Recently, Dill [Dil89] and Alur and Dill [AD90] proposed a method for incorporating timing restriction into a model of communicating finite-state systems by introducing the notion of a timed automaton, containing fictitious time-measuring elements called clocks [AD90] or timers [Dil89]. The verification problem is shown to be equivalent to the speed-independent verification problem on an appropriate automaton. The fundamental problem with both approaches is state space explosion, i.e. state space growing exponentially in the number of timers (clocks).

Kurshan [Kur91] suggested to carry out the verification process on timed systems with COSPAN [HK88], a verification system for untimed processes, by relaxing the time constraints, verifying the relaxed system and if the verification is unsuccessful check whether the run that violates the property to be verified is infeasible under the timing constraints. If this is so, Kurshan removes the run and repeats the process. This strategy is appealing but heuristic in nature. There was no proof that the process would eventually converge to provably the correct answer.

In this paper, we introduce the notion of *pauses*, and construct an equivalent (non-pausing) automaton. In contrast to previous approaches, we build an equivalent automaton as a composition of the speed-independent (or unrestricted) automaton and many small automata. This enables us to build simple abstractions of the equivalent automaton that are still detailed enough to eliminate certain undesired behavior. This leads to a verification strategy similar to the heuristic proposed by Kurshan, where a verification process is started with the unrestricted automaton, which is then composed with simple automata imposing timing constraints, but only after the verification has failed, and imposing only those constraints which are violated in the failure report. In the worst case, all constraints must be used, so that the verification is performed with the automaton of full complexity. However, some preliminary experiments have shown that the worst case is not the common case and that this strategy can indeed lead to considerable time and space savings.

The rest of this paper is organized as follows. L-processes [Kur90] are shortly discussed in section 2. In section 3, we introduce the notion of timed L-process, and then we construct the equivalent (not timed) L-process in section 4. In section 5 two main steps of the proposed verification strategy are described: extracting timing violations from the failure report, and imposing that subset of timing constraints to the model of the system. Final remarks are provided in section 6.

2 L-processes

An L-process [Kur90] is an automaton defined over the alphabet $S(L)$, where L is a Boolean algebra with product $*$, sum $+$, complement $\bar{}$, multiplicative identity 1, and additive identity 0. A partial

order $\leq$ in L is defined by: $x \leq y$ if and only if $x * y = x$. *Atoms* are minimal elements with respect to this ordering. If L is finite (and we will consider only those), it is completely specified by a set of its atoms, denoted by $S(L)$.

Atoms of L corresponds to the input alphabet in classical automata-theoretic terms. Introducing algebraic structure on the alphabet enables us to describe easily various sets of letters. For example if x is a letter (an atom), $\bar{x}$ denotes a set containing all other letters. Boolean algebras are particularly convenient, when the alphabet contains different alphabets of separate automata. In this case, in automata-theoretic terms the alphabet is a Cartesian product of components. In Boolean algebra terms atoms are just products of atoms of independent subalgebras.

Let $\hat{L} = L \cdot L^{\perp}$ where L and $L^{\perp}$ are independent subalgebras of a Boolean algebra $\hat{L}$. Then, for any $a \in S(\hat{L})$ it must be true that $a = a_1 * a_2$ where $a_1 \in S(L)$ and $a_2 \in S(L^{\perp})$. We say that a_1 is a *projection* of a to L, and write $\Pi_L(a) = a_1$.

The L-process P is a 5-tuple $(L_P, M_P, I(P), R(P), Z(P))$, where L_P (output subalgebra of P) is a subalgebra of L, $V(P)$ (a state space of P) is some non-empty set, M_P (a transition matrix of P) maps $V(P) \times V(P)$ to L, $I(P)$ (initial states) is some non-empty subset of $V(P)$, $R(P)$ (recur edges) is some subset of $V(P) \times V(P)$, and $Z(P)$ (cycle sets) is some set of subsets of $V(P)$.

Let $a = (a_1, a_2, \ldots) \in S(L)^{\omega}$ be an infinite sequence of atoms of L, and let $v = (v_1, v_2, \ldots)$ be an infinite sequence of elements of $V(P)$. We say that v is a *run* of a in P if and only if $a_k * M_P(v_k, v_{k+1}) \neq 0$ for all $k \geq 1$.

We say that $a \in L^{\omega}$ is in the language of P, and write $a \in \mathcal{L}(P)$, if and only if there exists a run of a in P, starting at some of the initial states, which neither crosses any of the recur edges infinitely often, nor remains forever in one of the cycle sets.

A projection is naturally extended to sequences of atoms and languages, i.e. a projection of a sequence is a sequence of projections of its elements, and a projection of the language is a set of projections of sequences in the language.

It can be verified automatically [HK88] whether the language of an L-process is contained in the language describing some properties. If this is not the case, there exists at least one loop of states reachable from the initial states, that does not contain any recur edges, is not fully covered by any cycle set and is not in the language of the task. Usually, one such a loop is included in the failure report produced by automatic tools.

3 Timed L-processes

First we introduce the notion of a *simple timed L-process.* Intuitively, we describe one pause by a pair of states $\{v_i^{\circledcirc}, v_i^{d}\}$, as shown in Figure 1. When a system enters a state $v_i^{\circledcirc}$, a pause begins. An

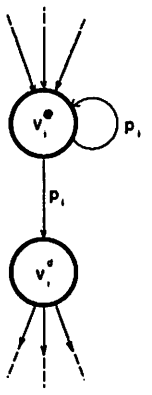

Figure 1: A pair of states representing one pause

atom p_i, uniquely associated with that state indicates that a pause is in progress. A pause finishes when a system exits a state v_i^d. An interval d_i contains the allowable durations of the pause.

Formally, a simple timed L-process T is a pair (P, d), where P is an L-process (called *unrestricted process of* T) and d is a set of pauses. A *pause* $i \in d$ is a quadruple $(v_i^@, v_i^d, p_i, d_i)$ satisfying the following:

- d_i is an integer bounded interval of positive real numbers (it can be open or close on the left or right, it can also be unbounded from above),
- $v_i^@$ and v_i^d are states of P and p_i is an atom of the output subalgebra of P, such that $M_P(v_i^@, v_i^@) = M_P(v_i^@, v_i^d) = p_i$, no other transitions are conditioned on p_i, $v_i^@$ has no other fanouts and v_i^d has no other fanins,
- $V^@$ (a set of all $v_i^@$), V^d (a set of all $v^d i$) and the set of initial states of P are mutually disjoint.

Figure 2 shows three examples of simple timed L-processes. The set of atoms of output subalgebra of the process T_1 (the output alphabet of T_1) is $\{a_1, p_1, b_1\}$. Similarly, output alphabets of T_2 and T_3 are $\{a_2, p_2, b_2\}$ and $\{a_3, p_3, b_3\}$, respectively. Each process contains one pause, with associated intervals $(2, \infty)$, $[1, 2)$ and $[1, 3]$, respectively.

For each pause i we define a set of feasible elapsed times in that pause, by: [1]

$$F_i = [0, \inf(d_i)] \cup d_i.$$

It is also useful to define a discretization function D_i for each pause i. The domain of D_i is F_i. We set $D_i(x) = x$ if x is an integer and $D_i(x) = \lfloor x \rfloor + \frac{1}{2}$ otherwise. The exception is made if F_i is not bounded from above when we set $D_i(x) = inf(d_i) + \frac{1}{2}$ whenever $x > \inf(d_i)$. We also define a discretization

[1] $\inf(d_i)$ and $\sup(d_i)$ denote infinum and supremum of d_i

function for a pair of pauses (i,j) such that $D_{ij}(x) = D_i(x)$ if $x \geq 0$ and $D_{ij}(x) = -D_j(-x)$ if $x < 0$. For the example in Figure 2 $F_2 = [0,2)$, the range of D_1 is $\{0, 0.5, 1, 1.5, 2, 2.5\}$ and the range of $D_{1,2}$ is $\{-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2, 2.5\}$.

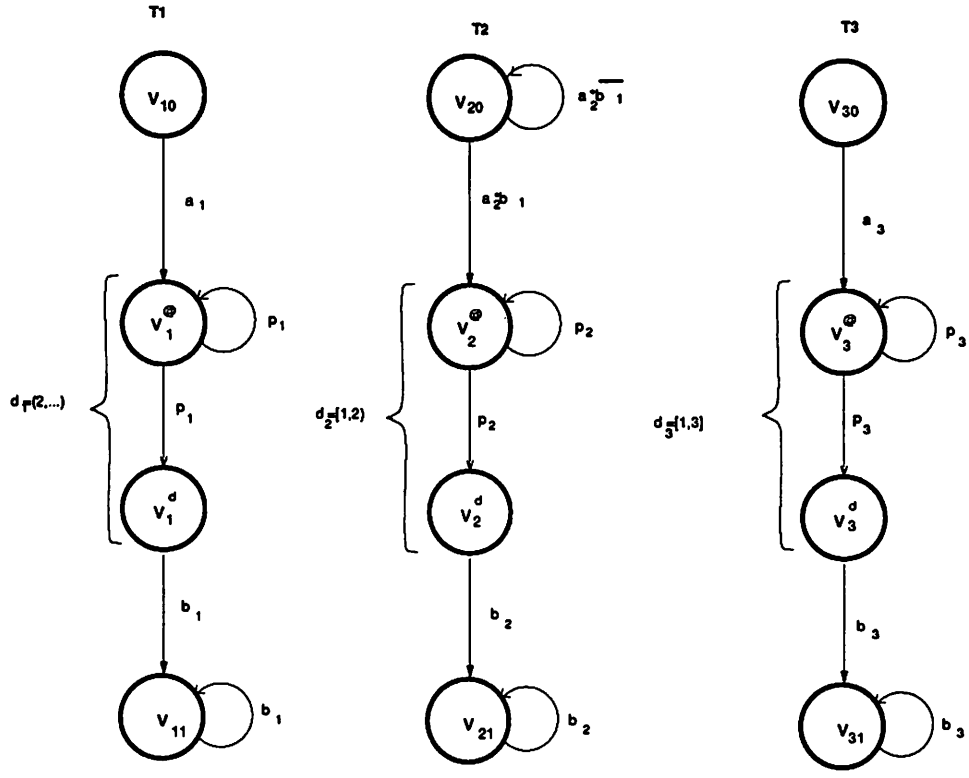

Figure 2: Examples of simple timed L-processes

Let $T = (P,d)$ and assume $a = (a_1, a_2, \ldots) \in \mathcal{L}(P)$. Also, let $v = (v_1, v_2, \ldots) \in V(P)^\omega$ be some accepting run of a in P. Then, for each $v_k \in V^d \cup V^@$ let $first(v_k)$ be the element of v where that pause has started, i.e. $first(v_k) = v_l \in V^@$ if and only if $v_l \neq v_{l-1}$ and $v_l = v_m$ for all $m = l+1, \ldots, k-1$.

Given a sequence a, we define a *timed sequence* (a,t) where t is a function assigning a real positive time t_k to every a_k in a. We can extend naturally a timing function to some run v of a by: $t(v_k) = t(a_k) = t_k$. For each v_k element of pause i we define the *elapsed time of pause i*: $\tau_k^i = t(v_k) - t(first(v_k))$. It is convenient to extend this definition to the whole v by setting $\tau_k^i = 0$, if v_k is not element of pause i.

We say that a t is a proper timing of v if all of the following consistency conditions hold:

1. $t_1 = 0$, $t_{k+1} \geq t_k$, for all $k \geq 1$ (time is non-decreasing),

2. if $v_k \notin V^d$ and $v_k \neq v_{k-1}$, then $t_k = t_{k-1}$ (all state changes outside a pause are instantaneous,

the time can advance only inside a pause, or in a self-loop),

3. if $v_k = v_i^d \in V^d$ then $\tau_k^i \in d_i$ (pause-finishing times are in d_i).

4. $\tau_k^i \in F_i$ (elapsed times are feasible, this condition is not redundant only if $v_k = v_i^{@}$ for all k greater then some k'),

For example, for the run $(v_{30}, v_3^{@}, v_3^{@}, v_3^d, v_{31}, \ldots)$ in T_3 from Figure 2, $(0, 0, 0.2, 2.8, 2.9, 2.9, \ldots)$ is a proper timing, while $(0, 0.2, \ldots)$ violates Condition 2, and $(0, 0, 0.2, 0.8, 0.9, 0.9, \ldots)$ violates Condition 3.

Now, we can define the language of a simple timed process. We say that a timed sequence (a, t) is in the language of a simple timed L-process $T = (P, d)$, and write $(a, t) \in \mathcal{L}(T)$, if and only if there exists v such that v is an accepting run of a in P, and t is a proper timing of v.

A *timed L-process* is a N-tuple of simple timed processes $(T_1, \ldots, T_N)$. We say that T_n, $1 \leq n \leq N$ are the components of T. A language of the timed process T is defined to be an intersection of languages of T_n's.

For some timed L-process, let V^d, $V^{@}$, d be unions of corresponding sets of its components. We say that P is the unrestricted process of T if P is a product of unrestricted processes of components of T.

We define an untimed language of T by:

$$Untime(\mathcal{L}(T)) = \{a | \exists t, (a, t) \in \mathcal{L}(T)\}.$$

Generally, $Untime(\mathcal{L}(T)) \subseteq \mathcal{L}(P)$ holds. It is easy to check that for simple timed L-process the equality holds.

Figure 2 shows an example of a timed L-process. Processes T_1 and T_3 starts pausing immediately, while process T_2 starts pausing when process T_1 finishes. One of the properties we might be interested in is:

"Is the state (v_{11}, v_{21}, v_3^d) reachable?"

We can rephrase this question in terms of the language:

"Does b_2 appears before b_3 in any sequence in the language?"

It is easy to see that the answer is negative. However, if we remove timing restrictions, the answer is positive, i.e. the language of the unrestricted process P contains the sequence in which b_2 appears before b_3.

4 Equivalent $\hat{L}$-process

We will show that for every timed L-process T there exists a process $\hat{P}$, such that $\Pi_L(\mathcal{L}(\hat{P})) = Untime(\mathcal{L}(T))$. We will define such a process in some extension $\hat{L}$ of algebra L, as a product of the *unrestricted $\hat{L}$-process* U, satisfying $\Pi_L(\mathcal{L}(U)) = \mathcal{L}(P)$, and the *region $\hat{L}$-process* R, imposing timing constraints by tracking down the possible values of elapsed times of pauses. Both U and R are constructed by forming the product of several smaller processes.

In the rest of this section we assume that $T = (T_1, \ldots, T_N)$ is a timed L-process, P is its unrestricted L-process, $P_1, \ldots, P_N$ are unrestricted L-processes, and $d_1, \ldots, d_N$ are sets of pauses of its components. We also assume that L is a subalgebra of $\hat{L}$.

We construct the process U from the process P by adding to its output information required by the process R. The first piece of information we need to add is whether or not a transition in P can take some time, as required by consistency condition 2. Formally, we assume that there exists[2] $\mathbf{B}(t)$, a subalgebra of $\hat{L}$ independent of L, and for $n = 1, \ldots, N$ we define a $\hat{L}$-process U'_n with output subalgebra $L_{P_n} \cdot \mathbf{B}(t)$, state space, initial states, recur edges and cycle sets all the same as in P_n and transition matrix:

$$M_{U'_n}(v, w) = \begin{cases} M_{P_n}(v, w) * \bar{t} & \text{if } v \neq w \text{ and } w \notin V^d \\ M_{P_n}(v, w) & \text{otherwise} \end{cases} .$$

We have introduced a new symbol t and labeled with $\bar{t}$ all transitions in P which must not take any time. Later on, we will mark transitions in R that must take some time with t, making sure that they can not happen simultaneously.

Another piece of information required by R is when pauses are finishing. Therefore, we will add one "flag" for each pause in P signaling when that pause is finishing. Formally, we assume that $\mathbf{B}(f_i)$ for all $i \in \bigcup_{n=1}^{N} d_n$ are subalgebras of $\hat{L}$ mutually independent and independent of L and $\mathbf{B}(t)$, and for $n = 1, \ldots, N$ we define a $\hat{L}$-process U_n with output subalgebra $L_{U'_n} \cdot \prod_{i \in d_n} \mathbf{B}(f_i)$, state space, initial states, recur edges and cycle sets all the same as in P_n and transition matrix:

$$M_{U_n}(v, w) = M_{U'_n}(v, w) * \prod_{i \in d_n} \tilde{f}_i, \text{ where } \tilde{f}_i = \begin{cases} f_i & \text{if } v = v_i^d \\ \bar{f}_i & \text{otherwise} \end{cases} .$$

Finally, let: $U = \bigotimes_{n=1}^{N} U_n$. It is easy to check that U behaves exactly like P, i.e. that $\Pi_L(\mathcal{L}(U)) = \mathcal{L}(P)$. Processes U_n for the example in Figure 2 are shown in Figure 3.

Before we formally define the region process R, let us consider a simple example. Consider pauses 3 and 2 in the example from Figure 2. In Figure 4 we record on the τ^3-axis an elapsed time in pause 3, and on the τ^2-axis, an elapsed time in pause 2. For any sequence $a \in \mathcal{L}(U_3 \otimes U_2)$ and any timing

[2] $\mathbf{B}(x)$ is a subalgebra of $\hat{L}$ containing x, with fewer elements than any other subalgebra of $\hat{L}$ containing x.

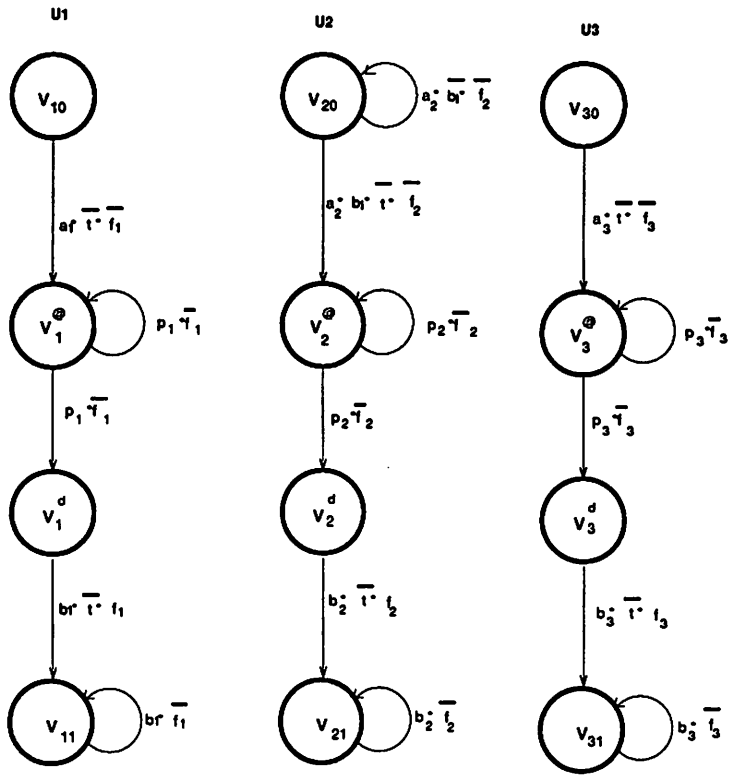

Figure 3: A process U for the example in Figure 2

t of a, we can construct a trajectory. The trajectory is constructed incrementally, adding a segment from (τ_k^3, τ_k^2) to $(\tau_{k+1}^3, \tau_{k+1}^2)$ according to a_k and t_k. The construction rules are as follows:

Rule 1 : we begin at point $(0,0)$ and stay there as long as neither T_3 nor T_2 are pausing, i.e. $a_k \leq \bar{p}_3 * \bar{p}_2$,

Rule 2(3) : if T_3 (T_2) is pausing and T_2 (T_3) is not, i.e. if $a_k \leq p_3 * \bar{p}_2$ ($a_k \leq \bar{p}_3 * p_2$), we move forward, along the τ^3 (τ^2) axis,

Rule 4 : if both T_3 and T_2 are pausing, i.e. if $a_k \leq p_3 * p_2$, we move forward, along a 45° line,

Rule 5(6) : if T_2 (T_3) is finishing pausing and T_3 (T_2) is not, i.e. if $a_k \leq f_2 * \bar{f}_3$ ($a_k \leq \bar{f}_2 * f_3$) we move to the point $(\tau_k^3, 0)$ ($(0, \tau_k^2)$ respectively),

Rule 7 : if both T_3 and T_2 are finishing pausing, i.e. if $a_k \leq f_2 * f_3$, we move to the point $(0,0)$.

The length of all forward movements is determined by $t_{k+1} - t_k$. One possible trajectory is shown in Figure 4. Each segment of the trajectory is labeled with the number of the rule that generated it. Note that for a proper timing of a rule 5 can only be applied in the region $F_3 \times d_2$. Similarly, rules 6 and 7 can only be applied in regions $d_3 \times F_2$ and $d_3 \times d_2$, respectively.

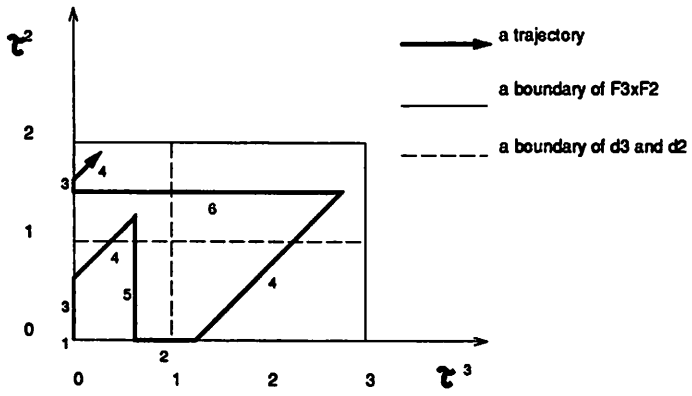

Figure 4: One trajectory in the feasible rectangle

We can partition the feasible values of (τ^3, τ^2) into regions r^x such that $(\tau^3, \tau^2) \in r^x$ if $D_{3,2}(\tau^3 - \tau^2) = x$. It is easy to see that such partition exist, and that it is unique and finite. A partition of the feasible rectangle in Figure 4 is shown in Figure 5. In this case, there are 10 regions: the region r^3 contains only a point $(3, 0)$, regions r^2, r^1, r^0 and r^{-1} contain segments of lines $\tau^3 - \tau^2 = 2$, $\tau^3 - \tau^2 = 1$, $\tau^3 - \tau^2 = 0$ and $\tau^3 - \tau^2 = -1$, respectively, and finally regions $r^{2.5}, r^{1.5}, r^{0.5}, r^{-0.5}$ and $r^{-1.5}$ contain strips $2 < \tau^3 - \tau^2 < 3$, $1 < \tau^3 - \tau^2 < 2$, $0 < \tau^3 - \tau^2 < 1$, $-1 < \tau^3 - \tau^2 < 0$ and $-2 < \tau^3 - \tau^2 < -1$, respectively.

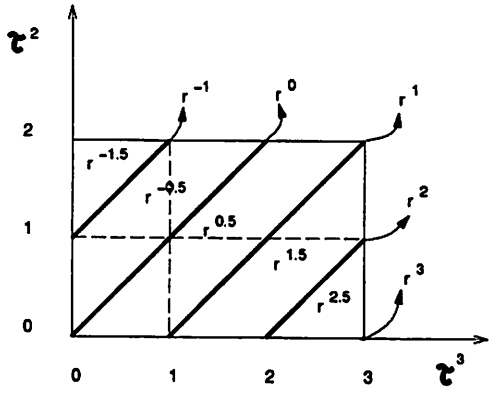

Figure 5: A partition of the feasible rectangle

We will construct the *region $\hat{L}$-process* for each pair of pauses by mapping the regions into states, and mapping the movement rules into transitions. The region process "observes" processes U_2 and U_3 and changes states according to the rules above. A transition between regions r^x and r^y exists if a segment of some proper trajectory connects two points in those regions. The transition is enabled if the conditions stated in the rule that generated the segment are met.

We don't have to construct such processes for all pairs of pauses. If two pauses belong to the

same simple timed process they can never be active simultaneously, so they have no interaction. Pairs of pauses not belonging to the same component process are called *interesting pairs of pauses*, and the set of such pauses for some timed L-process T is denoted by $IPP(T)$.

Generally, there will be more than one region process associated with the same pause. Obviously, all of these processes must agree on the value of elapsed time in that pause. To accomplish this we introduce one "multi valued variable" for each pause taking values in the range of associated discretization function and label each transition in every region process with the possible discretized value of the elapsed time in the next state. Formally, we assume that for each $i \in \bigcup_{n=1}^{N} d_n$ there exists L_i independent of any other L_j and all other subalgebras mentioned before, with the set of atoms: $S(L_i) = \{l_i^x | x \text{ in the range of } D_i\}$. For example, for pause 1 in Figure 2 $S(L_1) = \{l_1^0, l_1^{0.5}, l_1^1, l_1^{1.5}, l_1^2, l_1^{2.5}\}$.

The process $R_{3,2}$ resulting from a partition in Figure 5 is shown in Figure 6. All states are shown (with names $v_{3,2}^x$ abbreviated to x), but for simplicity only few transitions are shown. Enabling conditions for transitions shown in Figure 6 are:

$$\begin{aligned} M_{R_{3,2}}(v_{3,2}^{0.5}, v_{3,2}^{1.5}) &= t * p_3 * \bar{p}_2 * l_3^{1.5} * l_2^0 + & (1)\\ &\quad \bar{t} * \bar{f}_3 * f_2 * l_3^{1.5} * l_2^0, & (2)\\ M_{R_{3,2}}(v_{3,2}^{-1.5}, v_{3,2}^{-1.5}) &= t * p_3 * p_2 * l_3^{0.5} * l_2^{1.5} + & (3)\\ &\quad \bar{t} * \bar{f}_3 * \bar{f}_2 * (l_3^0 * l_2^{1.5} + l_3^{0.5} * l_2^{1.5}), & (4)\\ M_{R_{3,2}}(v_{3,2}^{-1}, v_{3,2}^{0}) &= \bar{t} * f_3 * f_2 * l_3^0 * l_2^0. & (5) \end{aligned}$$

We now formalize the notion of a region process. For each interesting pair of pauses $(i,j) \in IPP(T)$ we define a *pair region* $\hat{L}$-process R_{ij}. States of R_{ij} are regions described above, i.e.:

$$V(R_{ij}) = \{v_{ij}^x \subset F_i \times F_j | (\tau^1, \tau^2) \in v_{ij}^x \text{ if and only if } D_{ij}(\tau^1 - \tau^2) = x\}.$$

Formally, R_{ij} is an $\hat{L}$-process with output algebra $L_i \cdot L_j$, state space $V(R_{ij})$, unique initial state v_{ij}^0 (as required by Rule 1), no recur edges nor cycle sets.

We now define the transition matrix of R_{ij}. First we consider transitions that take time. By Rule 1, we can stay in the initial states as long as neither pause is active. Formally,

$$M_{R_{ij}}(v_{ij}^0, v_{ij}^0) = t * \bar{p}_i * \bar{p}_j * l_i^0 * l_j^0. \quad (6)$$

Possible discretized values of elapsed times are obviously zero in this case. By Rule 2, if i is active and j is not, we can move from some state v_{ij}^x, $x \geq 0$, to any state v_{ij}^y satisfying $y > x$ or $y \geq x$ if x is not an integer. Clearly, the discretized elapsed time of j in v_{ij}^y is 0 and of i is y. Formally, for

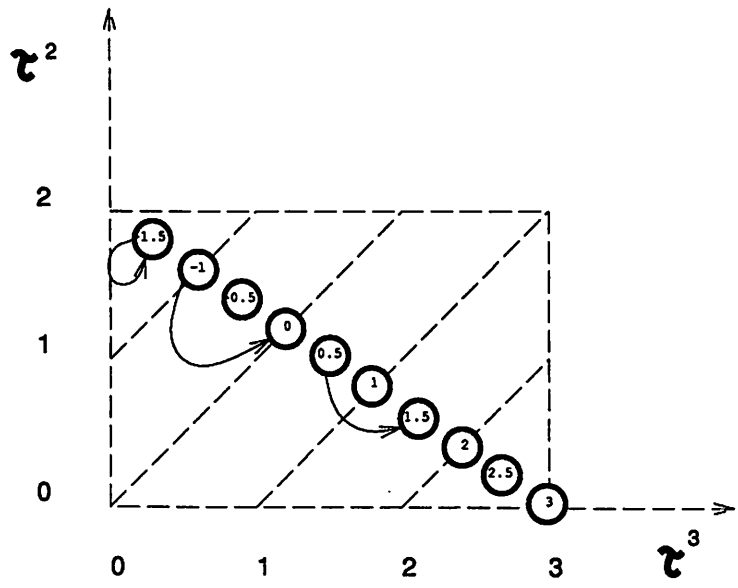

Figure 6: A part of the region process

any x and y satisfying $y > x \geq 0$ or $y \geq x > 0, x \notin Z$ we define a transition:

$$M_{R_{ij}}(v_{ij}^x, v_{ij}^y) = t * p_i * \bar{p}_j * l_i^y * l_j^0. \tag{7}$$

An example of such a transition is (1). Similarly, by Rule 3, for x and y satisfying $y < x \leq 0$ or $y \leq x < 0, x \notin Z$ we define:

$$M_{R_{ij}}(v_{ij}^x, v_{ij}^y) = t * \bar{p}_i * p_j * l_i^0 * l_j^{-y}. \tag{8}$$

Finally, by Rule 4, if both i and j are active, R_{ij} remains in the same region. However, if v_{ij}^x is an one-point region (like r^3 in Figure 5), Rule 4 can not be applied. Actually, no transition that takes time is possible from that state. Let $E_{ij}(x)$ be a set of pairs of discretized values of points in region v_{ij}^x, i.e. $E_{ij}(x) = \{(w,z) | w = D_i(\tau^1), z = D_j(\tau^2)$ for some $(\tau^1, \tau^2) \in v_{ij}^x\}$. The possible next-state values when both i and j are pausing and some time has passed, are all $(w,z) \in E_{ij}(x)$ with both $w > 0$ and $z > 0$. Now, we define formally:

$$M_{R_{ij}}(v_{ij}^x, v_{ij}^x) = t * p_i * p_j * \sum_{\substack{(w,z) \in E_{ij}(x) \\ w > 0, z > 0}} l_i^w * l_j^z, \tag{9}$$

and assume that the sum over an empty set is 0. That is the case if $x = \sup(d_i)$ or $x = -\sup(d_j)$. An example of such a transition is (3).

Next, we consider transitions that do not take time. It follows from all the rules that for 0 time

step R_{ij} remains in the same region, if neither i nor j are finishing. Formally:

$$M_{R_{ij}}(v_{ij}^x, v_{ij}^x) = \bar{t} * \bar{f}_i * \bar{f}_j * \sum_{(w,z)\in E_{ij}(x)} l_i^w * l_j^z. \tag{10}$$

An example of such a transition is (4). Rule 5 can be applied between v_{ij}^x and v_{ij}^y if for some point $(\tau^1, \tau^2) \in v_{ij}^x$ both $D_i(\tau^1) = y$ and $\tau^2 \in d_j$. In this case a discretized value of elapsed time in v_{ij}^y is y for i and 0 for j. Formally, if there exists $(\tau^1, \tau^2) \in v_{ij}^x$ such that $D_i(\tau^1) = y$ and $\tau^2 \in d_j$, then:

$$M_{R_{ij}}(v_{ij}^x, v_{ij}^y) = \bar{t} * \bar{f}_i * f_j * l_i^y * l_j^0. \tag{11}$$

An example of such a transition is (2). Similarly, by Rule 6, if there exists $(\tau^1, \tau^2) \in v_{ij}^x$ such that $\tau^1 \in d_i$ and $D_j(\tau^2) = -y$, then:

$$M_{R_{ij}}(v_{ij}^x, v_{ij}^y) = \bar{t} * f_i * \bar{f}_j * l_i^0 * l_j^{-y}. \tag{12}$$

Finally, by Rule 7, if there exists $(\tau^1, \tau^2) \in v_{ij}^x$ such that $\tau^1 \in d_i$ and $\tau^2 \in d_j$, then:

$$M_{R_{ij}}(v_{ij}^x, v_{ij}^0) = \bar{t} * f_i * f_j * l_i^0 * l_j^0. \tag{13}$$

An example of such a transition is (5).

Some of the pairs (v_{ij}^x, v_{ij}^y) may satisfy conditions for more than one of (6)–(13). In that case $M_{R_{ij}}(v_{ij}^x, v_{ij}^y)$ is the sum of all applicable expressions, as in (1)–(5).

Next, we define processes Q_i which track whether the elapsed time in pause i can be zero or not. For each pause i we define an $\hat{L}$-process Q_i with two states v_i^0 and v_i^1, the first one being a unique initial state, output subalgebra L_i, no recur edges nor cycle sets and transition matrix:

$$\begin{array}{llll} M_{Q_i}(v_i^0, v_i^0) = & (t * \bar{p}_i + \bar{t} * \bar{f}_i) * l_i^0 & M_{Q_i}(v_i^0, v_i^1) = t * p_i * \sum_{x\neq 0} l_i^x \\ M_{Q_i}(v_i^1, v_i^1) = & \bar{f}_i * \sum_{x\neq 0} l_i^x & M_{Q_i}(v_i^1, v_i^0) = f_i * l_i^0 \end{array}$$

Intuitively, Q_i is in v_i^0 if $\tau^i = 0$ and in v_i^1 if $\tau^i > 0$. A transition from v_i^0 to v_i^1 must absolutely take some time. A transition from v_i^1 to v_i^0 occurs if the pause i has finished. Note that f_i is not accepted in v_i^0.

Finally, let R be a product of all R_{ij}'s and all Q_i's and let $\hat{P} = U \otimes R$. It should be intuitively clear that any sequence that has a run in U and can be properly timed, has also a run in R. The opposite is also true, although not at all obvious: if a sequence has a run in $\hat{P}$, it can be properly timed. In the next section we described how abstractions of region processes can be used to eliminate from the language any sequence that can not be properly timed. These observations are formalized in the following theorem.

Theorem (equivalence theorem): $\Pi_L(\mathcal{L}(\hat{P})) = Untime(\mathcal{L}(T))$.

The proof is given in appendix.

5 Verification strategy

Verifying a task on $\hat{P}$ can run into difficulties, due to the large size of the state space that has to be searched. We propose a verification strategy to avoid this problem. We start a verification process with the unrestricted $\hat{L}$-process U. If the verification succeeds, we have verified the task. If the verification fails, there is at least one sequence in $S(\hat{L})^\omega$ which is in the language of the current abstraction of $\hat{P}$, but not in the language of the task. We analyze one run of such a sequence. If that run violates no timing constraints, we stop the verification unsuccessfully. However, if the run does violate some timing restrictions, we compose the current abstraction of $\hat{P}$ with some simple abstraction of process R, which is guaranteed to eliminate that run. We repeat this process until the verification is terminated, either successfully or unsuccessfully. This strategy can lead to significant savings in time and space, provided that the behavior of the system is not heavily dependent on the timing constraints. The verification strategy is outlined in Algorithm 1.

Algorithm 1: verification strategy

procedure verify_task()
  initialize $P_c = U$
  while not stop
    try to verify a task on P_c
    if success **then** stop, the task is verified
    find a timing violating loop G
    if such a loop does not exist **then** stop, the task is not verified
    P_c =eliminate_loop(G, P_c)
  end while
end procedure

5.1 Identifying timing violation

Assume that the error report from the verifier contains a loop and a path to that loop from the initial state. We can unfold the loop, thus forming a sequence of states. We form a graph with nodes being states in the sequence and the following set of edges:

- If some pause i begins at node k and is still active at node k' and $\sup(d_i)$ exists, we add an edge from k to k' and label it with $u_i = (-\infty, \sup(d_i)]$ if $\sup(d_i) \in d_i$, and $u_i = (-\infty, \sup(d_i))$ otherwise. We denote these edges with $f^{u_i}_{k,k'}$, and call them *forward pause edges.* These edges represent upper bounds on time distance between k and k', required by consistency condition 3.

- If some pause i begins at node k and is finishing at node k', we add an edge from k' to k and label it with $l_i = [\inf(d_i), \infty)$ if $\inf(d_i) \in d_i$, and $l_i = (\inf(d_i), \infty)$ otherwise. We denote these edges with $b^{l_i}_{k',k}$, and call them *backward pause edges.* These edges represent lower bounds on time distance between k and k',required by consistency condition 4.

- We add a edge from k to $k-1$ and label it with $l_0 = [0, \infty)$. We call these edges *backward non-pause edges.* They correspond to consistency condition 1.

- If $M_U(v_k, v_{k+1}) \leq \bar{t}$ we add an edge from k to $k+1$ and label it with $u_0 = (-\infty, 0]$. We call these edges *forward non-pause edges.* They correspond to consistency condition 2.

We say that a loop in the graph is overconstrained if every sum of numbers satisfying upper-bound constraints in forward edges is smaller than any sum of number satisfying lower-bound constraints in backward edges. An overconstrained loop indicates a timing violation. A loop that is not overconstrained can be properly timed. An overconstrained loop can be found by some modification of the standard constrained graph solver (e.g. [New87]). A modification is necessary to handle both included and excluded bounds correctly. Without loss of generality, we assume that the loop is minimal, in the sense that removing any edge enables proper timing of nodes.

Once a loop has been identified, we can collapse all non-pause edges, by merging their incident nodes. However, we mark nodes obtained by collapsing forward non-pause edges. Such a loop is an input to the algorithm which eliminates a timing constraint, described in the next subsection.

For example, for the timed L-process in Figure 2, a sequence of states:

$$v_1 = \begin{pmatrix} v_{10} \\ v_{20} \\ v_{30} \end{pmatrix}, v_2 = \begin{pmatrix} v_1^{@} \\ v_{20} \\ v_3^{@} \end{pmatrix}, v_3 = \begin{pmatrix} v_1^{d} \\ v_{20} \\ v_3^{@} \end{pmatrix}, v_4 = \begin{pmatrix} v_{11} \\ v_2^{@} \\ v_3^{@} \end{pmatrix}, v_5 = \begin{pmatrix} v_{11} \\ v_2^{d} \\ v_3^{@} \end{pmatrix}, \ldots \qquad (14)$$

is a run in U of a sequence that does not satisfy the property that b_2 can not appear before b_3. However, this sequence is not possible under the timing restrictions, because the time difference between v_2 and v_5 must be at most 3 (due to pause 3), but the time difference between v_2 and v_3 must be greater than 2 (due to pause 1), between v_3 and v_4 must be at least 0 and between v_4 and v_5 must be at least 1 (due to pause 2). The corresponding overconstrained loop (with non-pause edges collapsed) is show in Figure 7a. It has one forward edge $f^{u_3}_{1,3}$, where $u_3 = (-\infty, 3]$ (corresponding to the requirement that the distance between v_2 and v_5 is at most 3), and two backward edges: $b^{l_2}_{3,2}$, where $l_2 = [1, \infty)$ (corresponding to the requirement that the distance between u_4 and v_5 is at least 1), and $b^{l_1}_{2,1}$, where $l_1 = (2, \infty)$ (corresponding to the requirement that the distance v_2 and v_3 is greater than 2).

The constrained graph, as we defined it, is infinite. However, we can build it incrementally, adding one node at a time. While solving the constrained graph, we can easily compute possible

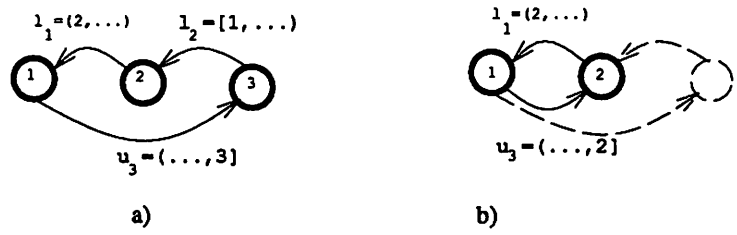

Figure 7: An overconstrained loop in two steps of the algorithm

discretized values of $\tau^i - \tau^j$, and we can establish whether τ^i can be 0. This, together with the state of U, determines the state of $\hat{P}$. Consequently, we can stop building the graph if we have repeated a state of $\hat{P}$. In this case, the run violates no constraints, so the verification process is unsuccessful.

5.2 Eliminating timing constraints

Given an overconstrained loop G, we want to build some abstraction of R which eliminates that run. The procedure is outlined in Algorithm 2.

Algorithm 2: eliminating a timing violation

procedure eliminate_loop(G, P_c) /* G - an overconstrained loop, P_c - a current abstraction of $\hat{P}$ */

for each edge $b^{l_j}_{k,m}$, m marked **do** $P_c = P_c \otimes Q_i$

for each pair of edges $f^{u_i}_{k,m}, b^{l_j}_{m,n}$, m marked **do** $P_c = P_c \otimes R^{u_0,l_j}_{ij} \otimes Q_i$

while there exist a pair $f^{u_i}_{k,m}, b^{l_j}_{m,n}$

$P_c = P_c \otimes R^{u_i,l_j}_{ij}$

remove from G edges $f^{u_i}_{k,m}$ and $b^{l_j}_{m,n}$

if $k < n$ **then** add to G edge $f^{NewU(u_i,l_j)}_{k,n}$

if $k > n$ **then** add to G edge $b^{NewL(u_i,l_j)}_{k,n}$

end while

return P_c

end procedure

We start with an overconstrained loop G, and consider a pair of edges $f^{u_i}_{k,m}, b^{l_j}_{m,n}$. For a moment, assume that m is not marked. Then, we can divide a feasible rectangle of i and j into two regions. A region where both constraints u_i and l_j could be simultaneously satisfied is called a "good" region. In other words, the good region contains all regions intersecting $u_i \times l_j$. A "bad" region contains all other regions. If u_i and l_j are as originally defined, the finish of pause j is not accepted in the bad region.

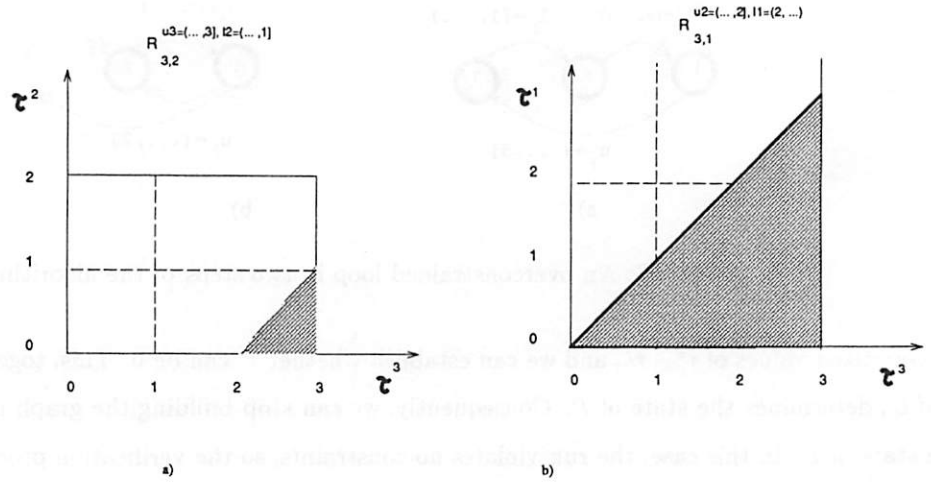

Figure 8: Two abstracted pair region processes

An *abstracted pair region process* $R_{ij}^{u_i,l_j}$. tracks whether the elapsed times of a pair of processes are in a good or bad region. Formally, it is a $\hat{L}$-process with output algebra $L_i \cdot L_j$, two states $V_{ij}^g = \{v_{ij}^x \in V(R_{ij}) | v_{ij}^x \text{ intersects } (u_i \times l_j)\}$ and $V_{ij}^b = V(R_{ij}) - V_{ij}^g$, a unique initial state being the one containing v_{ij}^0, no recur edges nor cycle sets, and transition matrix defined by:

$$M_{R_{ij}^{u_i,l_j}}(V,W) = \sum_{v \in V} \sum_{w \in W} M_{R_{ij}}(v,w).$$

The state V_{ij}^g corresponds to a good region, while the state V_{ij}^b corresponds to a bad region. We compose $R_{ij}^{u_i,l_j}$ with the current abstraction of $\hat{P}$, to distinguish between the two regions of elapsed times.

States of the abstracted pair region process for the overconstrained loop in Figure 7a are shown in the Figure 8a. The bad region is shaded. It contains states $v_{3,2}^3$ and $v_{3,2}^{2.5}$ of $R_{3,2}$. This process accepts the finish of pause 2 only in the good region.

Node marking complicates the procedure slightly. For example, if in the original G, we have edges $f_{k,m}^{u_i}, b_{m,n}^{l_j}$ and m is marked, then pause i finishes before pause j, but all transitions between these two events are subsets of $\bar{t}$, so no time can pass in between. In this case a process $R_{ij}^{u_i,l_j}$ is not enough, because even if it is in a bad state, when i finishes it will move into a good state. We need to combine $R_{ij}^{u_i,l_j}$ with $R_{ij}^{u_0,l_j}$ and Q_i. While $R_{ij}^{u_i,l_j}$ is in the bad state, $R_{ij}^{u_0,l_j}$ must also be in the bad state. When i finishes, $R_{ij}^{u_0,l_j}$ will be in the bad state, and must remain there as long as subsequent transitions are subsets of $\bar{t}$. Note that $R_{ij}^{u_0,l_j}$ will not accept the finish of j if the elapsed time of i is 0. But, Q_i accepts only 0 elapsed time as long as subsequent transitions are subsets of $\bar{t}$.

In some cases we also need processes Q_i to ensure that the elapsed time can not be greater than 0, before some transition intersect t.

Next, we propagate this partition backwards. For example, since the process in Figure 8a accepts the finish of pause 2 only in the good region, we request that the elapsed time of pause 3 to be larger or equal to 2 when pause 2 starts. Therefore, we remove the original edges and add an edge with the new constraint on pause 3, as shown in Figure 7b. Generally, for edges $f^{u_i}_{k,m}$ and $b^{l_j}_{m,n}$, if $k < n$ and the elapsed time of i is in the region $NewU(u_i, l_j)$ when j starts, where:

$$NewU(u_i, l_j) = \begin{cases} (-\infty, \sup(u_i) - \inf(l_j)] & \text{if } \sup(u_i) \in u_i, \inf(l_j) \in l_j \\ (-\infty, \sup(u_i) - \inf(l_j)) & \text{otherwise} \end{cases},$$

then the elapsed time of i and j are, and will remain in the good region. But, if the elapsed time of i is outside the $NewU(u_i, l_j)$, they will remain in the bad region. In this case we remove $f^{u_i}_{k,m}$ and $b^{l_j}_{m,n}$ from G and add an edge describing a new constraint on i.

Similarly, if $k > n$ and the elapsed time of j is outside the region $NewL(u_i, l_j)$ when i starts, where:

$$NewL(u_i, l_j) = \begin{cases} [\inf(l_j) - \sup(u_i), \infty) & \text{if } \sup(u_i) \in u_i, \inf(l_j) \in l_j \\ (\inf(l_j) - \sup(u_i), \infty) & \text{otherwise} \end{cases},$$

then the elapsed time of i and j will remain in the bad region. In this case we remove $f^{u_i}_{k,m}$ and $b^{l_j}_{m,n}$ from G and add an edge describing a new constraint on j.

We repeat this process until we process all constraints. For the example in Figure 7 this means just adding the process whose states are shown in Figure 8b. One can check now that a sequence that has a run (14) does not have a run in the product of processes in Figure 8. Initially, the process in Figure 8a (process A for short) is in the good state, while the process in Figure 8b (process B for short) is in the bad state. Process B remains in the bad state when pauses 1 and 3 start simultaneously, while process A can either stay in the good state or move to the bad state. The process B accepts the finish of pause 1 only if the elapsed time of pause 3 is larger than 2, but as long as pause 2 is not active, the elapsed time of pause 3 larger than 2 leads the process A into the bad state. Since A does not accept the finish of pause 2 in the bad region, we have eliminated a sequence that has a run 14 from the language. Thus, we have verified the property using the abstraction of R that has only 4 states, in contrast to the full process R that has 960 states ((12 of $R_{3,1}$)×(10 of $R_{3,2}$)×(2 of Q_1) ×(2 of Q_2)×(2 of Q_3)).

Generally, after we have finished, at each state of the original error run, at least one of the abstracted region pair processes is in the bad state. Eventually, one of the processes will not accept the finish of some pause. In this way, we have removed the error run from the language of the current abstraction of $\hat{P}$.

It is easy to see that $\mathcal{L}(R_{ij}) \subseteq \mathcal{L}(R^{u_i,l_j}_{ij})$. Therefore, by composing some abstraction of $\hat{P}$ with processes Q_i and abstracted pair region processes, we are iteratively approaching the language of $\hat{P}$.

Since there are only finitely many abstracted pair region processes, the iteration will converge in a finite number of steps.

This approach also offers some insight into the nature of the somewhat restricting assumption of integer bounded pauses. If we allow arbitrary real numbers to be the bounds for pauses duration, we can still use the procedure outlined in Algorithm 2 to iteratively approach the language of some timed L-process. However, in that case we are not guaranteed to converge in a finite number of steps.

6 Conclusions

To model timing behavior of finite-state systems, we have proposed timed L-processes. We believe that timed L-processes offer two major advantages over previous approaches. First, an equivalent L-process is defined as a composition of an unrestricted L-process and many smaller processes. We provide a transition matrix for each of these processes. In this way, the automatic generation of the equivalent process is simpler than in [Dil89] where there is one big region automaton and the computation of the next state function includes non-trivial matrix manipulation, and in [AD90] where the equivalent automaton is defined as a single automaton with very large state space.

More importantly, we propose a verification strategy to deal with the state space explosion problem. Basically, we propose a "trial and error" strategy, starting with the unrestricted process, and using at each step only the minimum subset of timing constraints necessary to eliminate the reported error. In the simple example discussed in this paper a reduction of two order of magnitudes in the number of the states has been achieved. Besides time and space saving, this strategy could have positive impact on the design. Indeed, to perform the required task, a design does not have to meet all timing constraints, but only those used in the verification. Relaxing of constraints could be used to optimize the design.

Appendix: Proof of the equivalence theorem

Throughout this appendix we assume that $T = (T_1, \ldots, T_N)$ is a timed L-process, P is its unrestricted L-process, $P_1, \ldots, P_N$ are unrestricted L-processes of components of T and $d_1, \ldots, d_N$ are sets of pauses of its components. Furthermore, let $d = \bigcup_{n=1}^{N} d_n$ and let $\hat{P}$ and all of its component processes be as defined in section 4, except that for simplicity, we assume that $IPP(T)$ contains all pairs of pauses, not only interesting ones. This assumption does not change the complexity of the proposed verification strategy, because by following the procedure in section 5, only abstractions of interesting region pair processes will be generated.

We have not completely defined the algebra $\hat{L}$, but only assumed that it has some independent subalgebras. It is convenient (although not necessary) to assume that these subalgebras completely specify $\hat{L}$, i.e. that the following definition holds.

Definition: $\hat{L} = L \cdot \mathbf{B}(t) \cdot \prod_{i \in d}(\mathbf{B}(f_i) \cdot L_i)$, where L, $\mathbf{B}(t)$, $\mathbf{B}(f_i)$ and L_i for all $i \in d$, are all independent subalgebras of $\hat{L}$.

The following two lemmas are direct consequences of the definitions of processes U_n (Lemma 1) and P_n (Lemma 2). In fact, they can be regarded as alternative definitions of transition matrices of those processes. Therefore, we state them here without a proof.

Lemma 1: For all $n = 1, \ldots, N$ and all $(v, w) \in V(U_n) \times V(U_n)$, $M_{U_n}(v, w) = M_{P_n}(v, w) * M^t_{U_n}(v, w) * M^f_{U_n}(v, w)$, where $M_{P_n}(v, w) \in L$, $M^t_{U_n}(v, w) \in \mathbf{B}(t)$, $M^f_{U_n}(v, w) \in \prod_{i \in d_n} \mathbf{B}(f_i)$ and

$$M^t_{U_n} = \begin{cases} \bar{t} & \text{if } v \in V^d \\ 1 & \text{otherwise} \end{cases}, \tag{15}$$

$$M^f_{U_n} = \prod_{i \in d_n} \tilde{f}_i, \text{ where } \tilde{f}_i = \begin{cases} f_i & \text{if } v = v^d_i \\ \bar{f}_i & \text{otherwise} \end{cases}. \tag{16}$$

Lemma 2: For all $(i, j) \in IPP(T)$, all $(v, w) \in V(R_{ij}) \times V(R_{ij})$, all x in the range of D_i and all y in the range of D_j:

a) $M_{R_{ij}}(v, w) \geq t * \bar{p}_i * \bar{p}_j * l^x_i * l^y_j$ if and only if $v = w = v^0_{ij}$ and $x = y = 0$,

b) $M_{R_{ij}}(v, w) \geq t * \bar{p}_i * p_j * l^x_i * l^y_j$ if and only if $x = 0$ and there exists $(0, \tau^2) \in v$ and $\delta > 0$ such that $(0, \tau^2 + \delta) \in w$ and $y = D_j(\tau^2 + \delta)$,

c) $M_{R_{ij}}(v, w) \geq t * p_i * \bar{p}_j * l^x_i * l^y_j$ if and only if $y = 0$ and there exists $(\tau^1, 0) \in v$ and $\delta > 0$ such that $(\tau^1 + \delta, 0) \in w$ and $x = D_i(\tau^1 + \delta)$,

d) $M_{R_{ij}}(v, w) \geq t * p_i * p_j * l^x_i * l^y_j$ if and only if $v = w$ and there exists $(\tau^1, \tau^2) \in v$ such that $x = D_i(\tau^1) > 0$ and $y = D_j(\tau^2) > 0$,

e) $M_{R_{ij}}(v, w) \geq \bar{t} * f_i * f_j * l^x_i * l^y_j$ if and only if $w = v^0_{ij}$, $x = y = 0$ and there exists $(\tau^1, \tau^2) \in v$ such that $\tau^1 \in d_i$ and $\tau^2 \in d_j$,

f) $M_{R_{ij}}(v, w) \geq \bar{t} * f_i * \bar{f}_j * l^x_i * l^y_j$ if and only if $x = 0$ and there exists $(\tau^1, \tau^2) \in v$ such that $\tau^1 \in d_i$, $(0, \tau^2) \in w$ and $y = D_j(\tau^2)$,

g) $M_{R_{ij}}(v, w) \geq \bar{t} * \bar{f}_i * f_j * l^x_i * l^y_j$ if and only if $y = 0$ and there exists $(\tau^1, \tau^2) \in v$ such that $\tau^2 \in d_j$, $(\tau^1, 0) \in w$ and $x = D_i(\tau^1)$,

h) $M_{R_{ij}}(v, w) \geq \bar{t} * \bar{f}_i * \bar{f}_j * l^x_i * l^y_j$ if and only if $v = w$ and there exists $(\tau^1, \tau^2) \in v$ such that $x = D_i(\tau^1)$ and $y = D_j(\tau^2)$.

Intuitively, Lemma 2 states that there exist a transition between two regions (under certain conditions), if and only if there exists a segment of some proper trajectory connecting two points in those regions (under those conditions). We can now prove the first half of the equivalence theorem, i.e. $Untime(\mathcal{L}(T)) \subseteq \Pi_L(\mathcal{L}(\hat{P}))$. First, for every $(a,t) \in \mathcal{L}(T)$ we define $\hat{a} \in S(\hat{L})^\omega$ such that $a = \Pi_L(\hat{a})$, and then we prove that $\hat{a} \in \mathcal{L}(\hat{P})$.

Definition: Assume $(a,t) \in \mathcal{L}(T)$ and assume $v_n = ((v_n)_1, (v_n)_2, \ldots, (v_n)_k, \ldots)$, for $n = 1, \ldots, N$, are accepting runs of a in P_n, such that t is the proper timing of all of these runs. Then, let $\hat{a} \in S(\hat{L})^\omega$ be such that for all $k \geq 1$:

$$\hat{a}_k = a_k * a_k^t * (\prod_{n=1}^{N} M_{U_n}^f) * \prod_{i \in d} l_i^{D_i(\tau_{k+1}^i)},$$

where $M_{U_n}^f$ is as in (16) , and:

$$a_k^t = \begin{cases} t & \text{if } t_{k+1} > t_k \\ \bar{t} & \text{if } t_{k+1} = t_k \end{cases}.$$

Lemma 3: For every $n = 1, \ldots, N$, $\hat{a} \in \mathcal{L}(U_n)$.

Proof: We show that v_n from the definition of $\hat{a}$ is an accepting run of $\hat{a}$ in U_n. Since initial states, recur edges and cycle sets are the same in P_n and U_n, it is enough to show that v_n is a run of $\hat{a}$ in U_n,i.e. that $\hat{a}_k * M_{U_n}((v_n)_k, (v_n)_{k+1}) \neq 0$. By Lemma 1, we have:

$$\hat{a}_k * M_{U_n}((v_n)_k, (v_n)_{k+1}) =$$

$$\underbrace{a_k * M_{P_n}((v_n)_k, (v_n)_{k+1})}_{\in L} * \underbrace{a_k^t * M_{U_n}^t((v_n)_k, (v_n)_{k+1})}_{\in \mathbf{B}(t)} * \underbrace{\prod_{n=1}^{N} M_{U_n}^f}_{\in \prod_{i \in d} \mathbf{B}(f_i)} * \underbrace{\prod_{i \in d} l_i^{D_i(\tau_{k+1}^i)}}_{\in \prod_{i \in d} L_i}.$$

To show that the left hand side is different from 0, it is enough to show that each of the underbraced parts of the right hand side is different from 0, since they belong to independent subalgebras of $\hat{L}$. Last two parts (elements of $\prod_{i \in d} \mathbf{B}(f_i)$ and $\prod_{i \in d} L_i$) are themselves products of atoms od independent subalgebras of $\hat{L}$, hence different from 0. The first part (element of L) is different from 0 by the assumption that v_n is a run of a in P_n. Finally, the second part (element of $\mathbf{B}(t)$) is certainly different from 0 when $M_{U_n}^t = 1$. Since by consistency condition 2, t_{k+1} must be equal to t_k whenever one of the processes P_n changes states outside a pause, a_k^t must be equal to $\bar{t}$ whenever $M_{U_n}^t = \bar{t}$, hence their product is different from 0.□

Lemma 4: For all $(i,j) \in IPP(T)$, $\hat{a} \in \mathcal{L}(R_{ij})$.

Proof: We will show that $r = (r_1, r_2, \ldots, r_k, \ldots)$ where r_k is such that $(\tau_k^i, \tau_k^j) \in r_k$, is an initialized run of $\hat{a}$ in R_{ij}. Since R_{ij} has no recur edges nor cycle sets, r is also an accepting run of $\hat{a}$ in R_{ij}. First, we check that r_1 is an initial state of R_{ij}. This is true because by consistency condition 1, $\tau_1^i = 0$ for all $i \in d$, so $r_1 = v_{ij}^0 \in I(R_{ij})$.

Next, we show that:

$$\hat{a}_k \leq M_{R,ij}(r_k, r_{k+1}) \tag{17}$$

Generally, (17) follows from Lemma 2. More specifically, let $\delta = t_{k+1} - t_k$ (where t_k is a proper timing of a_k, from the definition of $\hat{a}$). Then:

- if $a_k \leq t * \bar{p}_i * \bar{p}_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then by the definition of elapsed times, $\tau_k^i = \tau_k^j = \tau_{k+1}^i = \tau_{k+1}^j = 0$, so $r_k = r_{k+1} = v_{ij}^0$ and (17) holds by Lemma 2a,
- if $a_k \leq t * \bar{p}_i * p_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then $\delta > 0$, $\tau_k^i = \tau_{k+1}^i = 0$ and $\tau_{k+1}^j = \tau_k^j + \delta$, so (17) holds by Lemma 2b,
- if $a_k \leq t * p_i * \bar{p}_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then $\delta > 0$, $\tau_{k+1}^j = \tau_k^j = 0$ and $\tau_{k+1}^i = \tau_k^i + \delta$, so (17) holds by Lemma 2c,
- if $a_k \leq t * p_i * p_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then $\delta > 0$, $\tau_{k+1}^i = \tau_k^i + \delta > 0$, $\tau_{k+1}^j = \tau_k^j + \delta > 0$, so (17) holds by Lemma 2d,
- if $a_k \leq \bar{t} * f_i * f_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then $\tau_{k+1}^i = \tau_{k+1}^j = 0$, by consistency condition 3, $\tau_k^i \in d_i$ and $\tau_k^j \in d_j$, so (17) holds by Lemma 2e,
- if $a_k \leq \bar{t} * f_i * \bar{f}_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then $\tau_{k+1}^i = 0$, $\tau_{k+1}^j = \tau_k^j$ and $\tau_k^i \in d_i$ by consistency condition 3, so (17) holds by Lemma 2f,
- if $a_k \leq \bar{t} * \bar{f}_i * f_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then $\tau_{k+1}^j = 0$, $\tau_{k+1}^i = \tau_k^i$ and $\tau_k^j \in d_j$ by consistency condition 3, so (17) holds by Lemma 2g,
- if $a_k \leq \bar{t} * \bar{f}_i * \bar{f}_j * l_i^{D_i(\tau_{k+1}^i)} * l_j^{D_j(\tau_{k+1}^j)}$, then $\tau_{k+1}^i = \tau_k^i$ and $\tau_{k+1}^j = \tau_k^j$ so (17) holds by Lemma 2h.□

Lemma 5: For all $i \in d$, $\hat{a} \in \mathcal{L}(Q_i)$.

Proof: We will show that $q = (q_1, q_2, \ldots, q_k, \ldots)$, where:

$$q_k = \begin{cases} v_i^0 & \text{if } \tau_k^i = 0 \\ v_i^1 & \text{if } \tau_k^i > 0 \end{cases},$$

is an initialized run of $\hat{a}$ in Q_i. Since Q_i has no recur edges nor cycle sets, q is also an accepting run of $\hat{a}$ in Q_i. Initially, $\tau_k^i = 0$, so $q_1 = v_i^0$ is indeed an initial state of Q_i. Next, consider a case $\tau_{k+1}^i = \tau_k^i = 0$. It is possible only if $t_{k+1} > t_k$ and pause i is inactive (i.e. $a_k \leq t * \bar{p}_i$), or $t_{k+1} = t_k$ (i.e. $a_k \leq \bar{t}$). In either case $a_k \leq \bar{f}_i$ must also hold, because of the consistency condition 3 and the fact that $0 \notin d_i$ for any i. So, we have:

$$a_k \leq (t * \bar{p}_i + \bar{t} * \bar{f}_i) * l_i^0 = M_{Q_i}(v_i^0, v_i^0) = M_{Q_i}(q_k, q_{k+1}).$$

The case $\tau_{k+1}^i > \tau_k^i = 0$ is possible only if $t_{k+1} > t_k$ and pause i is active. So,

$$a_k \leq t * p_i * l_i^{D_i(\tau_{k+1}^i)} \leq M_{Q_i}(v_i^0, v_i^1) = M_{Q_i}(q_k, q_{k+1}).$$

The case $\tau_{k+1}^i \geq \tau_k^i > 0$ is possible as long as pause i does not finish, so in this case:

$$a_k \leq \bar{f}_i * l_i^{D_i(\tau_{k+1}^i)} \leq M_{Q_i}(v_i^1, v_i^1) = M_{Q_i}(q_k, q_{k+1}).$$

Finally, $\tau_k^i > \tau_{k+1}^i = 0$ is possible only if pause i finishes, in which case:

$$a_k \leq f_i * l_i^{D_i(\tau_{k+1}^i)} = M_{Q_i}(v_i^1, v_i^0) = M_{Q_i}(q_k, q_{k+1}).\square$$

Lemma 6: $Untime(\mathcal{L}(T)) \subseteq \Pi_L(\mathcal{L}(\hat{P}))$.

Proof: It is easy to check that $\Pi_L(\hat{a}) = a$, and by Lemmas 3–5 $\hat{a} \in \mathcal{L}(\hat{P})$. So for every $(a,t) \in \mathcal{L}(T)$, hence also for every $a \in Untime(\mathcal{L}(T))$ there exists $\hat{a} \in S(\hat{L})^\omega$, such that $\Pi_L(\hat{a}) = a$ and $\hat{a} \in \mathcal{L}(\hat{P})$.$\square$

We now turn to the proof of the second part of the equivalence theorem, i.e. we need to show that $Untime(\mathcal{L}(T)) \supseteq \Pi_L(\mathcal{L}(\hat{P}))$. In other words, we need to show that every sequence in the language of $\hat{P}$ can be properly timed. In the rest of this appendix we assume that $\hat{a} = (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k, \ldots) \in S(\hat{L})^\omega$ is in the language of $\hat{P}$ and that $((r_{ij})_1, (r_{ij})_2, \ldots, (r_{ij})_k, \ldots)$ is an accepting run of $\hat{a}$ in R_{ij}. We also assume that $\hat{a}$ is partitioned into subsequences according to the following rules:

- $\hat{a}_1$ is in the epilogue of subsequence 1. We say that $\hat{a}_1$ begins subsequence 1 and write $beg(1) = 1$. We also say that $\hat{a}_1$ begins the epilogue of subsequence 1 and write $epi(1) = 1$.

- If $\hat{a}_k$ is in the epilogue of subsequence m and $\hat{a}_k < \bar{t}$, then $\hat{a}_{k+1}$ is also in the epilogue of subsequence m.

- If $\hat{a}_k$ is in the epilogue of subsequence m, $\hat{a}_k < t$ and $\hat{a}_k < \bar{l}_i^0$ for all pauses i, then $\hat{a}_{k+1}$ is in the prologue of subsequence $m+1$. In this case, we say that $\hat{a}_{k+1}$ begins subsequence $m+1$ and write $beg(m+1) = k+1$. We also say that $\hat{a}_k$ ends subsequence m and write $end(m) = k$.

- If $\hat{a}_k$ is in the epilogue of subsequence m, $\hat{a}_k < t$ and $\hat{a}_k < l_i^0$ for some pause i, then $\hat{a}_{k+1}$ is in the epilogue of subsequence $m+1$. In this case, we say that $\hat{a}_{k+1}$ begins subsequence $m+1$ and its epilogue and write $beg(m+1) = epi(m+1) = k+1$. We also say that $\hat{a}_k$ ends subsequence m and write $end(m) = k$.

- If $\hat{a}_k$ is in the prologue of subsequence m, and $\hat{a}_k < \bar{l}_i^0$ for all pauses i, then $\hat{a}_{k+1}$ is also in the prologue of subsequence m.

- If $\hat{a}_k$ is in the prologue of subsequence m, and $\hat{a}_k < l_i^0$ for some pause i, then $\hat{a}_{k+1}$ is in the epilogue of subsequence m. In this case, we say that $\hat{a}_{k+1}$ begins the epilogue of subsequence m and write $epi(m) = k + 1$.

In summary, subsequences have a prologue, which may be empty and an epilogue, which always has at least one element. All pauses are active in all but last elements of the prologue, and all but last elements of the epilogue are contained in $\bar{t}$. If the prologue is not empty, at least one of the pauses finishes at the end of the prologue. The prologue is empty if at least one pause is inactive at the end of previous sequence, so $\hat{a}_{epi(m)-1} < l_i^0$ always holds for some pause i. If $end(m)$ exists, then $\hat{a}_{end(m)} < t$ holds. A sequence has infinitely many finite subsequences, or the last subsequence has infinite prologue or epilogue. We will show that there exist a proper timing of any $\hat{a} \in \mathcal{L}(\hat{P})$, such that all elements of the subsequence are assigned the same time. Furthermore, for any two pauses i and j elapsed times at step k will be in the region $(r_{ij})_k$. In the proof, we use the following property of discretization functions.

Lemma 7: Let $\tau_1, \tau_2, \sigma_1, \sigma_2$ be such that:

$$D_{ij}(\tau_1 - \tau_2) = D_{ij}(\sigma_1 - \sigma_2). \tag{18}$$

Then there exists δ such that

$$D_i(\sigma_1 + \delta) = D_i(\tau_1) \text{ and } D_j(\sigma_2 + \delta) = D_j(\tau_2). \tag{19}$$

Proof First, consider a case where both $D_i(\tau_1) = \tau_1$ and $D_j(\tau_2) = \tau_2$ are integers. Then:

$$D_{ij}(\tau_1 - \tau_2) = \tau_1 - \tau_2 = D_{ij}(\sigma_1 - \sigma_2) = \sigma_1 - \sigma_2,$$

implying that $\delta = \tau_1 - \sigma_1 = \tau_2 - \sigma_2$ satisfies (19).

Next, consider a case $D_i(\tau_1) = \tau_1$ is an integer and $D_j(\tau_2)$ is not, i.e. $lo_j(\tau_2) < \tau_2 < up_j(\tau_2)$, where:

$$lo_i(x) = D_i(x) - \frac{1}{2}, \quad up_i(x) = \begin{cases} \infty & \text{if } d_i \text{ unbounded and } x > \inf(d_i) \\ D_i(x) + \frac{1}{2} & \text{otherwise} \end{cases}.$$

Then:

$$\tau_1 - up_j(\tau_2) < \tau_1 - \tau_2 < \tau_1 - lo_j(\tau_2). \tag{20}$$

Since $\tau_1 - \tau_2$ and $\sigma_1 - \sigma_2$ belong to the same region, the bounds on $\tau_1 - \tau_2$ in (20) must also hold for $\sigma_1 - \sigma_2$, i.e.:

$$\tau_1 - up_j(\tau_2) < \sigma_1 - \sigma_2 < \tau_1 - lo_j(\tau_2).$$

which in turn implies that $\delta = \tau_1 - \sigma_1$ satisfies (19), rewritten as:

$$\sigma_1 + \delta = \tau_1,$$

$$lo_j(\tau_2) - \sigma_2 < \delta < up_j(\tau_2) - \sigma_2.$$

Similarly, if $D_i(\tau_1)$ is not an integer, i.e. $lo_i(\tau_1) < \tau_1 < up_i(\tau_1)$ and $D_j(\tau_2) = \tau_2$ is , then:

$$lo_i(\tau_1) - \tau_2 < \tau_1 - \tau_2 < up_i(\tau_1) - \tau_2 \Longrightarrow$$
$$lo_i(\tau_1) - \tau_2 < \sigma_1 - \sigma_2 < up_i(\tau_1) - \tau_2,$$

and $\delta = \tau_2 - \sigma_2$ satisfies (19).

Finally, if neither $D_i(\tau_1)$ nor $D_j(\tau_2)$ are integers, i.e. $lo_i(\tau_1) < \tau_1 < up_i(\tau_1)$, $lo_j(\tau_2) < \tau_2 < up_j(\tau_2)$ then:

$$lo_i(\tau_1) - up_j(\tau_2) < \tau_1 - \tau_2 < up_i(\tau_1) - lo_j(\tau_2) \Longrightarrow$$
$$lo_i(\tau_1) - up_j(\tau_2) < \sigma_1 - \sigma_2 < up_i(\tau_1) - lo_j(\tau_2),$$

imply that (19), rewritten as:

$$lo_i(\tau_1) - \sigma_1 < \delta < up_i(\tau_1) - \sigma_1,$$
$$lo_j(\tau_2) - \sigma_2 < \delta < up_j(\tau_2) - \sigma_2.$$

has a solutions, because lower bounds on δ are strictly smaller than the upper bounds.□

The following result follows easily from Lemmas 2 and 7, and the simple fact that for any pause i: $x \in d_i$ if and only if $D_i(x) \in d_i$.

Corollary 1: For any $(i,j) \in IPP(T)$ and any $\hat{a} \in \mathcal{L}(\hat{P})$:

a) if $\hat{a}_k < p_i * p_j * l_i^x * l_j^y$, then for every $(\tau^1, \tau^2) \in (r_{ij})_k$ there exists δ such that: $D_i(\tau^1 + \delta) = x$, $D_j(\tau^2 + \delta) = y$ and for any such δ: $(\tau^1 + \delta, \tau^2 + \delta) \in (r_{ij})_{k+1} = (r_{ij})_k$,

b) if $\hat{a}_k < \bar{f}_i * f_j * l_i^x$ then for every $(\tau^1, \tau^2) \in (r_{ij})_k$ there exists δ such that: $D_i(\tau^1 + \delta) = x$, $\tau^2 + \delta \in d_j$ and for any such δ: $(\tau^1 + \delta, \tau^j_{k+1} = 0) \in (r_{ij})_{k+1} = v^x_{ij}$,

c) if $\hat{a}_k < f_i * f_j$ then for every $(\tau^1, \tau^2) \in (r_{ij})_k$ there exists δ such that: $\tau^1 + \delta \in d_i$, $\tau^2 + \delta \in d_j$ and for any such δ: $(\tau^i_{k+1} = 0, \tau^j_{k+1} = 0) \in (r_{ij})_{k+1} = v^0_{ij}$.

Intuitively, Corollary 1 states that if i and j are both active, then for any point in $(r_{ij})_k$, there exists a point on the same 45° line connected to some point in $(r_{ij})_{k+1}$ by a segment of some proper trajectory. This result is useful to prove the existence of proper timing of subsequence $m+1$, once a timing for subsequence m has been chosen.

We also need the following result.

Lemma 8: For any $(i,j) \in IPP(T)$ and any $\hat{a} \in \mathcal{L}(\hat{P})$:

a) if $\hat{a}_k < l_i^0$ and $\hat{a}_{k+1} < \bar{t}$, then $\hat{a}_{k+1} < \bar{f}_i * l_i^0$,

b) if $\hat{a}_k < p_i * l_i^x * l_j^0$ and $\hat{a}_{k+1} < \bar{t}$, then $\hat{a}_{k+1} < p_i * \bar{f}_j * l_i^x * l_j^0$ or $\hat{a}_{k+1} < f_i * \bar{f}_j * l_i^0 * l_j^0$, the letter being possible only if $x \in d_i$,

c) if $\hat{a}_k < l_i^x * l_j^0$, $\hat{a}_{k'} < p_i$ for all $k' = k, \ldots, k''$ and $\hat{a}_{k'} < t$ for at least one such k', then $\hat{a}_{k''} < l_i^y$ where y is such that for every τ satisfying $D_i(\tau) = x$ there exists $\delta > 0$ satisfying $D_i(\tau+\delta) = y$.

Proof: Part a) follows immediately from the definition of processes Q_i.

By part a), $\hat{a}_{k+1} < \bar{f}_j * l_j^0$ in part b). Furthermore, if $\hat{a}_{k+1} < p_i$, then by Lemma 2h $(r_{ij})_{k+1} = (r_{ij})_{k+2}$, so[3] $\hat{a}_{k+1} < l_i^x$. By Lemma 2 (parts c), g) or f) can be applied in this case) $(r_{ij})_{k+1} = v_{ij}^x$, so by Lemma 2f, $\hat{a}_{k+1} < f_i * \bar{f}_j$ can be possible only if there exists $\tau \in d_i$ such that $(\tau, 0) \in v_{ij}^x$. But the statement $\exists \tau \in d_i : (\tau, 0) \in v_{ij}^x$ is equivalent to $\exists \tau \in d_i : D_{ij}(\tau - 0) = x$, hence equivalent to $\exists \tau \in d_i : D_i(\tau) = x$, and finally, the last statement is equivalent to $x \in d_i$.

For the proof of part c) we first consider a case $\hat{a}_{k'} < \bar{f}_j$ for all $k < k' \leq k''$. If j is not active, part c) follows from parts c) and h) of Lemma 2. If j is active, then by parts d) and h) of Lemma 2 $(r_{ij})_{k'} = v_{ij}^x$ for all k'. Furthermore, after the first k' satisfying $a_{k'} < t$, Q_j will go to v_j^1 and remain there until at least k''. So, $a_{k''} < l_i^y * l_j^z$, where, $(y,z) \in E_{ij}(x)$ (by parts d) and f) of Lemma 2) and $z > 0$ (due to the state of Q_j). It is easy to check that any such y satisfies the condition in part c).

If $a_{k'} < f_j$ for some $k < k' \leq k''$, we first apply the same argument to the subsequence $k, \ldots, k'-1$. We can do that because by Lemma 9a at least one element of $\hat{a}$ between k and k' must be contained in t. It follows from Lemma 2g and the fact $(r_{ij})_{k'-1} = v_{ij}^x$, that $a_{k'} < l_i^y$ where y satisfies the condition in part c). Since $a_{k'} < f_j$ implies $a_{k'} < l_j^0$, we can now repeat the same argument for the rest of the sequence (this time with relaxed condition $\delta \geq 0$).□

Informally, part a) states that if elapsed time of pause i is 0, i can not finish as long as the following elements of the sequence are contained in $\bar{t}$ (e.g. in the epilogue of some subsequence). Part b) states that a pause can finish in the epilogue only if the signaled discretized value of elapsed time (i.e. x in l_i^x) satisfies consistency condition 3 ($x \in d_i$). These two result are useful to show that we can indeed assign the same time to all elements of some subsequence. Part c) states that the signaled discretized value of elapsed time in pause i can only grow between beginning of epilogues of two consecutive subsequences, unless i finishes in between. We use this result to show that the

[3] Here, we use the fact that for all regions, if we fix the discretized value of elapsed time in one pause in the next state (in this case we fix j to 0), then the discretized value of elapsed time in the other pause in the next state is uniquely determined.

time assigned to subsequence $m+1$ (following the procedure described bellow) is indeed larger than the time assigned to subsequence m.

The next Lemma provides a basis for an inductive construction of proper timing of $\hat{a}$.

Lemma 9: Assume that $\hat{a} \in \mathcal{L}(\hat{P})$ has been partitioned into subsequences and assume that subsequences $1,\ldots,m$ (all finite) have been properly timed with $t_1,\ldots,t_m$ (the same time for all elements of subsequence). Also, assume that subsequence $m+1$ has a finite (possibly empty) prologue, and that all elements of subsequence $m+1$ have been assigned the time $t_{m+1} = t_m + \delta$, where δ satisfies:

$$\delta > 0 \tag{21}$$

$$D_i(\tau^i_{end(m)} + \delta) = x_i \text{ whenever } \hat{a}_{epi(m)} < p_i * l_i^{x_i} \tag{22}$$

$$\tau^i_{end(m)} + \delta \in d_i \text{ whenever } \hat{a}_{epi(m)} < f_i \tag{23}$$

Then, the following holds:

a) $t_1,\ldots,t_{m+1}$ is a proper timing of subsequences $1,\ldots,m+1$,

b) if for all $(i,j) \in IPP(T)$:

$$(\tau^i_{end(m)}, \tau^j_{end(m)}) \in (r_{ij})_{end(m)}, \tag{24}$$

then there exists δ satisfying (21)–(23),

c) if (24) is satisfied and δ satisfies (21)–(23), then

$$(\tau^i_{end(m+1)}, \tau^j_{end(m+1))}) \in (r_{ij})_{end(m+1)},$$

for all $(i,j) \in IPP(T)$.

Proof:

Part a) : Consistency condition 1 is trivially satisfied by (21). Consistency condition 4 is trivially satisfied by (22)–(23) for pauses that satisfy respective conditions. If the prologue is not empty, every pause must satisfy either the condition in (22) or the condition in (23). If the prologue is empty, there might be some pauses satisfying $\hat{a}_{epi(m)} < \bar{p}_i * \bar{f}_i * l_i^0$, but for these pauses elapsed time is 0 throughout the subsequence, so consistency condition 4 is satisfied for all pauses.

Consistency condition 2 is trivially satisfied for all k in the subsequence, except $k = end(m+1)$, because $t_{k+1} = t_k$. From the definition of processes U_n it follows that enabling condition of all changes of states outside a pause are contained in $\bar{t}$. Since $\hat{a}_{end(m+1)} < t$, we conclude that no changes of state outside a pause can occur in U at $\hat{a}_{end(m+1)}$, so consistency condition 2 is satisfied for $k = end(m+1)$ as well.

Consistency condition 3 is obviously satisfied for all pauses that meet the condition in (23). By Lemma 9b it is satisfied for all of the pauses that meet the condition in (22), and by Lemma 9a no other pauses (i.e. pauses satisfying $\hat{a}_{epi(m)} < \bar{p}_i * \bar{f}_i * l_i^0$) can finish in this subsequence.

Part b) : Each expression of type (21)–(23), defines an interval of possible values of δ, so to show there is a solution satisfying all of these constraints, it is enough to show that each pair of constraints has a solution. First, we observe that $(r_{ij})_{epi(m+1)-1} = (r_{ij})_{end(m)}$ either because the prologue is empty, i.e. $end(m) = epi(m+1) - 1$, or by repeatedly applying parts d) and h) of Lemma 2 through the prologue of subsequence $m+1$ (since all pauses are active in the prologue, only these two cases apply). Hence, we can rewrite a condition (24) as:

$$(\tau^i_{end(m)}, \tau^j_{end(m)}) \in (r_{ij})_{epi(m+1)-1} \tag{25}$$

Now, we can apply Corollary 1, to show that if (25) is satisfied, there exists δ satisfying any pair of constraints, each being either of type (22) or (23).

The constraint (21) obviously has a solution with a constraint of type (22) if $\tau^i_{end(m)} = 0$. Also, for such i it must be true tha $a_{end(m)-1} < l_i^0$, because it either has finished in the epilogue of subsequence m, or was not active at the beginning of the epilogue of m. In either case, at least one a_k in the epilogue of m was contained in l_i^0 (by the definition of Q_i), and so is $a_{end(m)-1}$ by Lemma 8a. Note that there always exists at least one such pause.

If this is not the case, then $\tau^i_{end(m)} > 0$, and there exist j such that $a_{end(m)-1} < l_j^0$. From (24) and parts b) to d) of Lemma 2, it follows that $a_{end(m)-1} < l_i^{D_i(\tau^i_{end(m)})}$. Now we can apply Lemma 8c to show that there exists δ satisfying (21) and any constraint of type (22).

The constraint (21) obviously has a solution with a constraint of type (23) if $\tau^i_{end(m)} \neq \sup(d_i)$. But, if $\tau^i_{end(m)} = \sup(d_i)$ the only region satisfying (24) is a region containing a single point $(\sup(d_i), 0)$, and from Lemma 2 all edges from that state are contained in $\bar{t}$. Since $a_{end(m)} < t$, we conclude that the case $\tau^i_{end(m)} = \sup(d_i)$ is not possible.

Part c) : First we prove:

$$(\tau^i_{epi(m+1)-1}, \tau^j_{epi(m+1)-1}) \in (r_{ij})_{epi(m+1)-1}.$$

If the prologue of m is empty it is trivially true because $epi(m+1) - 1 = end(m)$. If the prologue is not empty all pauses are active, so none of the R_{ij} change state, and:

$$\left.\begin{array}{c} (\tau^i_{end(m)}, \tau^j_{end(m)}) \in (r_{ij})_{end(m)} \\ (r_{ij})_{end(m)} = (r_{ij})_{epi(m+1)-1} \\ \tau^i_{epi(m+1)-1} = \tau^i_{end(m)} + \delta \\ \tau^j_{epi(m+1)-1} = \tau^j_{end(m)} + \delta \end{array}\right\} \Rightarrow (\tau^i_{epi(m+1)-1}, \tau^j_{epi(m+1)-1}) \in (r_{ij})_{epi(m+1)-1}.$$

Next, we show:

$$(\tau^i_{epi(m+1)}, \tau^j_{epi(m+1)}) \in (r_{ij})_{epi(m+1)}). \tag{26}$$

If the prologue is not empty, it is true by Corollary 1, however if the prologue is empty we need to consider cases $a_{end(m)} < p_i * \bar{p}_j * \bar{f}_j$ and $a_{end(m)} < \bar{p}_i * \bar{p}_j * \bar{f}_i * \bar{f}_j$. The latter case is simple because $\tau^i_{end(m)} = \tau^j_{end(m)} = \tau^i_{epi(m+1)} = \tau^j_{epi(m+1)} = 0$ and $(r_{ij})_{end(m)} = (r_{ij})_{epi(m+1)} = v^0_{ij}$. If $a_{end(m)} < p_i * \bar{p}_j * \bar{f}_j * l^{x_i}_i * l^0_j$, then (22) is equivalent to

$$D_{ij}((\tau^i_{end(m)} + \delta) - 0) = x_i,$$

and since $\tau^i_{epi(m+1)} = \tau^i_{end(m)} + \delta$ and $\tau^j_{epi(m+1)} = 0$, this is equivalent to:

$$(\tau^i_{epi(m+1)}, \tau^j_{epi(m+1)}) \in v^{x_i}_{ij}$$

and finally, by Lemma 2 $(r_{ij})_{epi(m+1)} = v^{x_i}_{ij}$.

Finally, we observe that (26) implies $(\tau^i_k, \tau^j_k) \in (r_{ij})_k$ for all k in the epilogue of m (hence, also for $k = end(m)$, because of the part b) and c) of Corollary 1 and the fact that if $a_k < \bar{f}_i * \bar{f}_j$ neither elapsed times of i and j nor the state of R_{ij} can change in the prologue of m.□

Now, we are able to prove the second part of the equivalence theorem.

Lemma 10: $Untime(\mathcal{L}(T)) \supseteq \Pi_L(\mathcal{L}(\hat{P}))$.

Proof: We need to show that there exists a proper timing for every $\Pi_L(\hat{a}) \in \Pi_L(\mathcal{L}(\hat{P}))$, or equivalently, for every $\hat{a} \in \mathcal{L}(\hat{P})$. Lemma 9 provides basic inductive argument for the construction of such a timing. However, we yet need to prove the base case, i.e. that $t_1 = 0$ is a proper timing of subsequence 1 satisfying (24) with $m = 1$, and the case where the last subsequence has the infinite prologue.

A timing $t_1 = 0$ is a proper timing of subsequence 1, because consistency conditions 1 and 4 are obviously satisfied, consistency condition 2 is satisfied by the same argument as in the proof of Lemma 9 and we claim that the consistency condition 3 is satisfied because no pauses can finish in subsequence 1. This is true, because all processes Q_i are initially in the state v^0_i, which does not accept the finish of pause i, and Q_i will remain there until at least $end(1)$ (since all other elements of subsequence 1 are contained in $\bar{t}$).

Since $a_k < \bar{f}_i * \bar{f}_j * \bar{t}$ for all $k < end(1)$, $(r_{ij})_k = v^0_{ij}$ for all $k \leq end(1)$, so (24) is satisfied, because:

$$(\tau^i_{end(1)}, \tau^j_{end(1)}) = (0,0) \in v^0_{ij} = (r_{ij})_{end(1)}.$$

If the last subsequence (say m) has an infinite prologue, we assign it a time $t_m = t_{m-1}$, which obviously satisfies consistency conditions 1 and 2, but also consistency condition 4 by inductive hypothesis, and consistency condition 3, because no pauses can finish in the infinite prologue.□

Finally, Lemma 6 and Lemma 9 prove the equivalence theorem.

Acknowledgment

The authors would like to thank Prof. R. Brayton, R. Murgai and T. Villa for many useful discussions. We also acknowledge R. Kurshan for his presentations at UCBerkeley that sparked our interest in the subject. This work has been supported by DARPA under contract JFBI90-073.

References

[AD90] Rajeev Alur and David L. Dill. Automata for modelling real-time systems. In M.S. Paterson, editor, *ICALP 90 Automata, languages, and programming: 17th international colloquium*. Springer-Verlag, 1990. LNCS vol. 443.

[Dil89] David L. Dill. Timing assumptions and verifications of finite-state concurrent systems. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite-State Systems*. Springer-Verlag, 1989. LNCS vol. 407.

[HK88] Z. Har'El and R. P. Kurshan. Software for analysis of coordination. In *Proceedings of the International Conference on System Science*, pages 382–385, 1988.

[Kur90] R. P. Kurshan. Analysis of discrete event coordination. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems : Models, Formalisms, Correctness*, pages 414–453. Springer-Verlag, 1990. LNCS vol. 430.

[Kur91] R. P. Kurshan, 1991. private communications.

[New87] A. R. Newton. Symbolic layout and procedural design. In G. De Micheli and Alberto L. Sangiovanni-Vincentelli, editors, *Design Systems for VLSI Circuits : Logic Synthesis and Silicon Compilation*. Martin Nijhoff, 1987.