On Time and Action Lock Free Description of Timed Systems

Howard Bowman*

Computing Laboratory, University of Kent at Canterbury,
Canterbury, Kent, CT2 7NF, United Kingdom

Email: H.Bowman@ukc.ac.uk

WWW: http://www.cs.ukc.ac.uk/people/staff/hb5/

Abstract

Time and action locks can arise freely in timed automata specification. While both are error situations, time locks are by far the more serious fault. This is because their occurrence prevents any further evolution of the system. First we investigate techniques for avoiding the occurrence of timelocks. The central aspect of our solution is a redefinition of automata parallel composition based on the Timed Automata with Deadlines Framework of Bornot and Sifakis. Then the second result we present is a notion of parallel composition which preserves action lock freeness. In the sense that, if any component automaton is action lock free, then the composition will also be action lock free.

1 Introduction

Deadlocks are the characteristic error situation arising in concurrent systems. In very general terms, they are states in which the system is unable to progress further.

Classically the term deadlock has been seen as synonymous with what we will call *action locks*. These are situations in which, how ever long time is allowed to progress, the system will never be able to perform an action[1]. Such action locks often result from unmatchable action offers, e.g. when a component wishes to perform a synchronisation action, but is unable to because no other process can offer a matching synchronisation. For example, if `?` denotes an input activity and `!` an output activity, the parallel composition `|<A,B>` of the two automata

*Howard Bowman is currently on leave at VERIMAG, Centre Equation, 2 rue Vignate, 38610 GIERES, France with the support of an EU Marie Curie Fellowship.

[1] Note that even if real-time is not modelled explicitly, e.g. in (untimed) process algebra such as CCS, in temporal terms, deadlocks still conceptually have this interpretation.

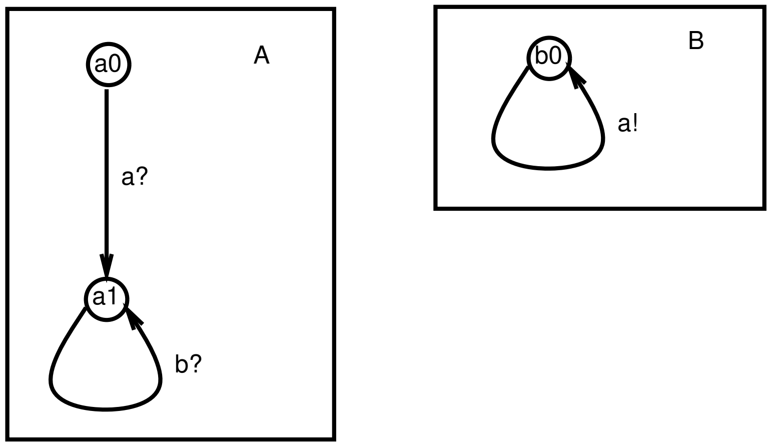

Figure 1: A simple action lock

shown in figure 1 will be action locked in state `a1 b0`. This is because how ever long either party waits they will never be able to fulfil the synchronisation they are requesting. Much of concurrency theory research has been dominated by the issue of deadlocks and their detection.

In the context of timed systems, new locking situations arise. In particular, in this paper, we will be working in an environment with two main types of locking situation. As a result of this, we have had to be careful with our choice of terminology. Thus, in this paper the term deadlock is the most general. It embraces action locks and the form of locking behaviour that comes with timed systems - timelocks[2].

Timelocks are situations in which, informally speaking, time is prevented from passing beyond a certain point. They are highly degenerate occurrences [6] because they yield a *global* blockage of the systems evolution. In particular, if a completely independent component is composed in parallel with a system that is timelocked, then the entire composition will inherit the timelock. This is quite different from an action lock, which cannot affect the evolution of an independent process. These characteristics of timelocks will be illustrated in section 2.

In fact, the issue of whether timelocks are desirable or undesirable features of timed models remains a hotly debated topic. The standard argument in favour of models containing timelocks is that they represent specification inconsistencies (like logical contradictions) and that by discovering and eliminating timelocks specifications can be corrected. However, we take the contrary position for three main reasons:-

1. In fact, detecting timelocks is a difficult and expensive analysis task. The

[2]The reader should be aware that this terminology is not universally used, for example, [12] uses different terminology and [6] uses the term local deadlock instead of action lock.

classic method for demonstrating timelock freeness is to show that a formula such as (in fact, this is the formula that would be used with the Kronos model checker),

$$\texttt{init} \Rightarrow \Box\Diamond_{=1}(\texttt{true})$$

holds over a timed automaton specification. This is an unbounded liveness property which is one of the most difficult classes of formulae to check. Such formulae can be checked with some symbolic real-time model checkers, e.g. Kronos [8][3]. However, the analysis that is required is extremely state space intensive and is only feasible with small and moderate sized specifications. Recent work by Tripakis [12] offers potential improvements in such analysis. However, his algorithm remains unimplemented and furthermore, such improvements will always be thwarted by systems with fundamentally large state spaces.

2. We are also strongly of the opinion that inconsistencies and contradictions fit in the domain of logical description, but are difficult to reconcile with behavioural specification techniques, such as timed automata. Contradictions arise when conflicting *properties* are asserted / conjoined. However, although the mistake is frequently made, parallel composition of processes is not a property composition operator, rather its meaning is operational - two (or more) physical components are *run* in parallel. This reflects the character of behavioural description which is fundamentally operational in nature. Error situations in behavioural techniques should have a behavioural / operational intuition that is justifiable in terms of real world behaviour. This is the case for action locks and live locks. However, there is no real world counter-part for time stopping.

3. The real-world should always be the yardstick for judging formal models and timelocks do not arise in the real-world!

Broadly there are two approaches to responding to the existence of locking errors:

1. *Detection:* provide analysis techniques which can locate such locking situations. Then system developers can detect and rectify the deadlocks.

2. *Prevention:* adapt / limit the specification models used in order to ensure that such locking situations cannot arise, e.g. [6] [3, 4].

As already suggested, a problem that arises with the first of these approaches is that deadlock detection analysis is typically expensive and when state spaces become large the techniques are infeasible. Thus, in this paper we investigate the second option.

It is also important to emphasize that the situation with action locks and timelocks is, in this respect, a little different. In our opinion timelocks are

[3] Although it cannot currently be checked with UPPAAL.

highly counter-intuitive and thus we believe that the second option above of constructively preventing the occurence of timelocks is essential. However, since action locks are not in the same way counter-intuitive, prevention is not in the same sense essential. Nonetheless investigating techniques which ensure action lock freeness is useful since it highlights forms of parallel composition that can be employed when building systems that are "correct by construction".

The model of timed systems that we employ is timed automata [1]. These are an enhancement of automata which enables concurrency, synchronisation and timing aspects to be expressed. Furthermore, because of their amenability to verification (via symbolic model checking), timed automata are now perhaps the most accepted real-time specification notation. We will introduce timed automata shortly.

Although pleasingly simple, the timing model of timed automata has the weakness that timelocks can freely arise and in a number of different ways. Perhaps most problemmatically they can arise through the interplay of urgency and synchronous interaction. We argue that urgency is given too strong an interpretation in timed automata. In the sense that an action can be forced (i.e. it becomes urgent) even if it is not possible (i.e. is not enabled). We will return to this issue a number of times during this paper and particularly in subsection 3.3.

The first of the two main contributions of this paper is to present a re-interpretation of synchronisation that weakens the effect of urgency and thus limits the occurrence of timelocks. The approach borrows heavily from the Timed Automata with Deadlines (TADs) framework of Bornot and Sifakis [3, 4]. However, in the same way as we did in [6] we adapt the TADs definitions to meet our needs.

The timed prioritised choice features offered by the TADs framework yield the possibility that the dynamic enabling of "competing" transitions can be defined statically. Hence we can investigate notions of parallel composition that preserve different dynamic properties. In this vein the second of the main contributions of this paper is to present a notion of parallel composition which preserves action lock freeness, in the sense that, if any of the component TADs is action lock free then the parallel composition will also be action lock free.

Structure of Paper. Section 2 presents background material. We introduce some basic timed automata notation, we clarify the difference between time and action locks and we introduce our running example - a simple timeout behaviour. Then we tackle the issue of timelocks in section 3. We first consider zeno timelocks. Then we illustrate the problem of time action locks (which are perhaps the most degenerate example of timelocks) through an attempt to specify the running example.

Then we consider solutions based on TADs. However using the timeout as an example, we argue that the TADs parallel composition presented in [3, 4] yields an unsatisfactory solution. We thus consider alternative solutions (Sparse TADs and TADs with minimal priority escape transitions) which yield valid solutions based on the paper [6].

In section 4 we consider how to define parallel composition in such a way that if components are free of action locks the composition will also be free of action locks. By way of background material we consider independent parallelism in untimed and timed settings which have this action lock freeness property. However, since it fails to support synchronisation, independent parallelism is of only limited value. Thus, we present a new notion of parallel composition that builds from TADs with minimal priority escape transitions, which preserves action lock freeness in the desired manner.

2 Background

This section introduces background material. Firstly, we define timed automata and some associated notation in subsection 2.1, then (in subsection 2.2) we clarify the difference between time and action locks and finally, in subsection 2.3, we introduce our running example - the specification of a bounded timeout.

2.1 Timed Automata and Basic Notation

Notation. We briefly review some basic timed automata notation. We assume the following items.

- CA is a set of *completed* (or internal) actions, x, x', x_1, x_2, ..., y, y', y_1, y_2, ..., z, z', z_1, z_2, ... range over CA.
- $HA = \{\, x?, x! \,|\, x \in CA \,\}$ is a set of *half* (or *uncompleted*) actions, a, a', a_1, a_2, ... , b, b', b_1, b_2 range over HA. These give a simple CCS style [9] point-to-point communication similar, for example, to the synchronisation primitives found in UPPAAL [2]. Thus, two actions, $x?$ and $x!$ can synchronise and generate a completed action x. For a half action a we let $\downarrow a$ denote the underlying completed action, i.e. $\downarrow(x?) = \downarrow(x!) = x$.
- $\mathbb{A} = HA \cup CA$ is the set of *all* actions, e, e', e_1, e_2, ... range over $\mathbb{A}$.
- We use a complementation notation over elements of $\mathbb{A}$,

$$\begin{aligned} \overline{x} &= x \quad \text{if } x \in CA && (1)\\ \overline{x?} &= x! && (2)\\ \overline{x!} &= x? && (3) \end{aligned}$$

- $\mathbb{R}^+$ denotes the positive reals without zero and $\mathbb{R}^{+0} = \mathbb{R}^+ \cup \{0\}$.
- $\mathbb{C}$ is the set of all clock variables, which take values in $\mathbb{R}^{+0}$. $\mathbb{C}$ is ranged over by c, c', c_1, c_2, etc. CC is a set of clock constraints[4]. Also if $C \subseteq \mathbb{C}$ we write CC_C for the set of clock constraints generated from clocks in C.

[4]The form that such constraints can take is typically limited, however since we are not considering verification this is not an issue for us.

- $\mathbb{V} = \mathbb{C} \rightarrow \mathbb{R}^{+0}$ is the space of possible clock valuations. $\mathbb{V}$ is ranged over by v, v', v_1, v_2, etc and $\mathbb{V}_C = C \rightarrow \mathbb{R}^{+0}$ is the space of clock valuations for clocks in C.

- $\mathbb{L}$ is the set of all possible automata locations (these appear as circles in our timed automata diagrams, e.g. see figure 2), ranged over by l, l', l_1, l_2, etc.

Timed Automata. An arbitrary element of $\mathcal{A}$, the set of all timed automata, has the form:

$$(L, l_0, T, I, C)$$

where,

- $L \subseteq \mathbb{L}$ is a finite set of locations;

- $l_0 \in L$ is a designated *start location*;

- $T \subseteq L \times \mathbb{A} \times CC_C \times \mathbb{P}(C) \times L$ is a transition relation (where $\mathbb{P}(S)$ denotes the powerset of S). A typical element of T would be, (l_1, e, g, r, l_2), where $l_1, l_2 \in L$ are automaton locations; $e \in \mathbb{A}$ labels the transition; $g \in CC_C$ is a guard; and $r \in \mathbb{P}(C)$ is a reset set. $(l_1, e, g, r, l_2) \in T$ is typically written, $l_1 \xrightarrow{e,g,r} l_2$, stating that the automaton can evolve from location l_1 to l_2 if the (clock) guard g holds and in the process action e will be performed and all the clocks in r will be set to zero. When we depict timed automata, we write the action label first, then the guard and then the reset set, see e.g. figure 2. Guards that are *true* or resets that are empty are often left blank.

- $I : L \rightarrow CC_C$ is a function which associates an invariant with every location. Intuitively, an automaton can only stay in a state while its invariant is satisfied. Invariants are shown adjacent to states in our depictions, see e.g. figure 2.

- C is the set of clocks of the timed automaton.

It is important to understand the difference between the role of guards and of invariants. In this respect we can distinguish between *may* and *must* timing. If we consider the TA in figure 2, we can see that the guard, `t>=5`, expresses may behaviour, i.e. it states that the transition is possible or in other words *may* be taken whenever `t>=5`. However, guards cannot "force" transitions to be taken.

In contrast, the invariant, `t<=10`, defines must behaviour, i.e. if `t` reaches `10` in state `b0`, `xxx` *must* be taken immediately. This must aspect corresponds to *urgency*, since an alternative expression of this situation is that at time `t=10` `xxx` becomes urgent - it must be taken straightaway.

Semantics. Timed automata are semantically interpreted over transition systems which are triples, $(S, s_0, \Rightarrow)$, where,

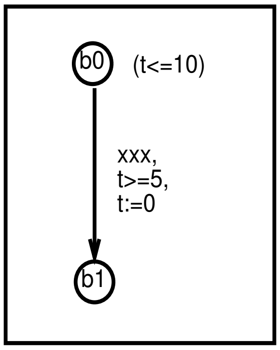

Figure 2: May and must timing

- $S \subseteq \mathbb{L} \times \mathbb{V}$ is a set of states (notice the terminological distinction - timed automata have locations while transition systems have states);
- $s_0 \in S$ is a start state;
- $\Rightarrow \subseteq S \times \mathit{Lab} \times S$ is a transition relation, where $\mathit{Lab} = \mathbb{A} \cup \mathbb{R}^+$. Thus, transitions can be of one of two types: *discrete transitions*, e.g. (s_1, e, s_2), where $e \in \mathbb{A}$ and *time transitions*, e.g. (s_1, d, s_2), where $d \in \mathbb{R}^+$ and denotes the passage of d time units. Transitions are written:

$$s_1 \stackrel{e}{\Longrightarrow} s_2 \quad \text{respectively} \quad s_1 \stackrel{d}{\Longrightarrow} s_2$$

Also, we will use the standard notation, $s \stackrel{e}{\Longrightarrow}$ *iff* $\exists s', e \,.\, s \stackrel{e}{\Longrightarrow} s'$ and $s \stackrel{e}{\not\Longrightarrow}$ *iff* $\neg(s \stackrel{e}{\Longrightarrow})$.

For a clock valuation $v \in \mathbb{V}_C$ and a delay d, $v + d$ is the clock valuation such that $(v + d)(c) = v(c) + d$ for all $c \in C$. For a reset set r, we use $r(v)$ to denote the clock valuation v' such that $v'(c) = 0$ whenever $c \in r$ and $v'(c) = v(c)$ otherwise. v_0 is the clock valuation that assigns all clocks to the value zero.

The semantics of a timed automaton $A = (L, l_0, T, I, C)$ is a transition system, $(S, s_0, \Rightarrow)$, where $S = \{\, s' \in L \times \mathbb{V}_C \mid \exists s \in S,\ y \in \mathit{Lab} \,.\, s \stackrel{y}{\Longrightarrow} s' \,\} \cup \{\, [l_0, v_0] \,\}$, $s_0 = [l_0, v_0]$ and $\Rightarrow$ is defined by the following inference rules:-

$$\frac{l \xrightarrow{e,g,r} l' \quad g(v)}{[l, v] \stackrel{e}{\Longrightarrow} [l', r(v)]} \qquad \frac{\forall d' \leq d \,.\, I(l)(v + d')}{[l, v] \stackrel{d}{\Longrightarrow} [l, v + d]}$$

The semantic map which generates transition systems from timed automata is written $[\![\ ]\!]$. Also, notice that our construction ensures that only reachable states are in S.

Parallel Composition. We assume our system is described as a network of timed automata. These are modelled by a vector of automata[5] denoted,

[5]Although our notation is slightly different, our networks can be related, say, to the process networks used in UPPAAL.

$|A = |\langle A[1], ..., A[n]\rangle$ where $A[i]$ is a timed automaton. In addition, we let u, u' etc, range over the set $\mathbb{U}$ of vectors of locations, which are written, $\langle u[1], ..., u[n]\rangle$, where each $u[i]$ is the current location in the ith automaton, i.e. in $A[i]$. In addition, $|u|$ and $|A|$ denote the length of the corresponding vector. We use a substitution notation as follows: $\langle u[1], ..., u[j], ..., u[n]\rangle[u[j]'/u[j]] = \langle u[1], ..., u[j-1], u[j]', u[j+1], ..., u[n]\rangle$ and we write $[u[j]'/u[j]]$ as $[j'/j]$ and $u[i_1'/i_1]...[i_m'/i_m]$ as $u[i_1'/i_1, ..., i_m'/i_m]$.

If $\forall i(1 \leq i \leq n)\,.\,A[i] = (L_i, l_{i,0}, T_i, I_i, C_i)$ then the product automaton, which characterises the behaviour of $|\langle A[1], ..., A[n]\rangle$ is given by,

$$(L, l_0, T, I, C)$$

where $L = \{\, |u \,|\, u \in L_1 \times ... \times L_n \,\}$, $l_0 = |\langle l_{1,0}, ..., l_{1,n}\rangle$, T is as defined by the following two inference rules, $I(|\langle u[1], ..., u[n]\rangle) = I_1(u[1]) \wedge ... \wedge I_n(u[n])$ and $C = C_1 \cup ... \cup C_n$.

$$\frac{u[i] \xrightarrow{x?, g_i, r_i} u[i]' \quad u[j] \xrightarrow{x!, g_j, r_j} u[j]'}{|u \xrightarrow{x, g_i \wedge g_j, r_i \cup r_j} |u[i'/i, j'/j]} \qquad \frac{u[i] \xrightarrow{x, g, r} u[i]' \quad x \in CA}{|u \xrightarrow{x, g, r} |u[i'/i]}$$

where $1 \leq i \neq j \leq |u|$. Note, we write $x \leq k \neq r \leq y$ in place of $x \leq k \leq y \;\wedge\; x \leq r \leq y \wedge\; k \neq r$.

2.2 Time and Action Locks

Timelocks. We can formulate the notion of a timelock in terms of a testing process. Consider, if we take our system which we denote `System` and compose it completely independently in parallel with the timed automaton, `Tester`, shown in figure 3, where, since it is completed, the `zzz` action is independent of all actions in the system. Then for any `d`$\in \mathbb{R}^+$, if the composition `|<Tester(d),System>` cannot perform `zzz` then the system contains a timelock at time `d`.

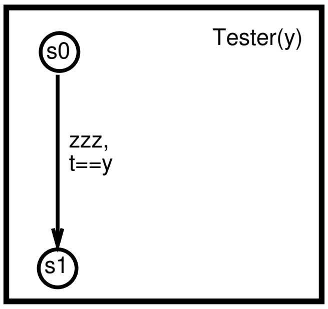

Figure 3: A tester process

This illustration indicates why timelocks represent such degenerate situations - even though the `Tester` is in all respects independent of the system, e.g. it could be that `Tester` is executed on the Moon and `System` is executed on Earth without any co-operation, the fact that the system cannot pass time prevents the tester from passing time as well. Thus, *time really does stop* and it stops everywhere because of a degenerate piece of *local* behaviour.

We can also give a semantic definition of the notion[6]. However, we first need a little notation.

A trace of a timed automaton A has the form,

$$\rho = s_0\, y_1\, s_1\, y_2\, s_2\, ...\, s_{n-1}\, y_n\, s_n$$

where,

- $\forall i (0 \leq i \leq n)\, .\, s_i \in [\![\, A \,]\!].1$ (throughout the paper we use the notation $t.i$ to access the ith element of a tuple);
- $s_0 = [l_0, v_0]$;
- $y_i \in \mathbb{A} \cup \mathbb{R}^+$;
- $\forall i (0 \leq i \leq n-1)\, .\, s_i \xRightarrow{y_i} s_{i+1}$.

and we let $Tr(A)$ denote the set of all traces of A. Furthermore, we define the function *delay* as,

$$delay(\rho) = \Sigma\{\, y_i \mid 1 \leq i \leq n \;\wedge\; y_i \in \mathbb{R}^+ \,\}$$

Now we say that A can timelock at time d iff

$$\exists \rho \in Tr(A)\, .\, (delay(\rho) < d \;\wedge\; \forall \sigma \in Tr(A)\, .\, (\rho\; pref\; \sigma \implies delay(\sigma) < d))$$

where $\rho_1\; pref\; \rho_2$ if and only if ρ_1 is a prefix of ρ_2. Intuitively this expresses that there is a state reachable before d time units has passed, from which it is not possible for time to elapse beyond d. Notice this definition does not preclude the system evolving "while timelocked" but it simply prevents time eventually reaching d. Indeed, as will become clear shortly, this is necessary to embrace zeno timelocks within the definition.

Also notice that situations in which time is able to, but *does not have to* evolve beyond a certain point, are not categorised as timelocks, e.g. a timed automaton such as that shown in figure 4 could perform an infinite number of `xxx` actions at time zero but since it is not forced to behave in this way we do not view it as timelocked.

There are two different forms of timelock:-

1. *Zeno Timelocks.* These arise when the system has an infinite behaviour but time cannot pass beyond a certain point. In other terms, an infinite number of discrete transitions are performed in a finite period of time. An example of such a specification is `System1` (see figure 5); this is a zeno

[6] Similar definitions can be found in [12].

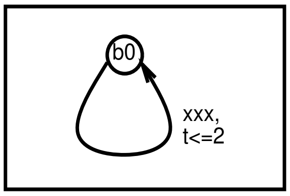

Figure 4: Zeno Behaviour without a Zeno Time Lock

process which performs an infinite number of `xxx` actions at time zero. This system is timelocked at time zero and if we compose it independently in parallel with any other system, the composite system will not be able to pass time.

2. *Time Action Locks.* These are situations in which a state is reached from which neither time or action transitions can be performed. An example of such a lock is the trivial timed automaton shown in figure 6 which timelocks immediately since the system can neither idle in state `b0` or perform an action transition to escape the state.

   However, more problemmatically, time action locks can be generated through mismatched synchronisations, e.g. the network `|<System2,System3>` (from figure 5) contains a timelock at time 2, which arises because `System2` must have performed (and thus, synchronised on) action `xxx` by the time `t` reaches 2 while `System3` does not start offering `xxx` until after `t` has past 2. Technically the timelock is due to the fact that at time 2 `System2` only offers the action transition `xxx` and importantly, it does not offer a time passing transition. Since the synchronisation cannot be fulfilled the system cannot evolve to a point at which it can pass time.

The interesting difference between these two varieties of timelock is that the first one locks time, but it is not action locked, since actions can always be performed. However, the second reaches a state in which neither time passing or action transitions are possible.

A relevant property which appears in the literature is that of time reactivity which is defined as follows.

Definition 1 *A system is said to be time reactive if it can never reach a state in which neither time or action transitions can be performed.*

Clearly if a system is time reactive it cannot contain time action locks. One aspect we investigate in this paper is how to obtain time reactivity in a timed automata setting.

Action Locks. Timelocks are much more serious faults than action locks. For example, the action locked automaton `Stop`, shown in figure 7, generates a *local deadlock*, however, it cannot prevent an independent process from evolving.

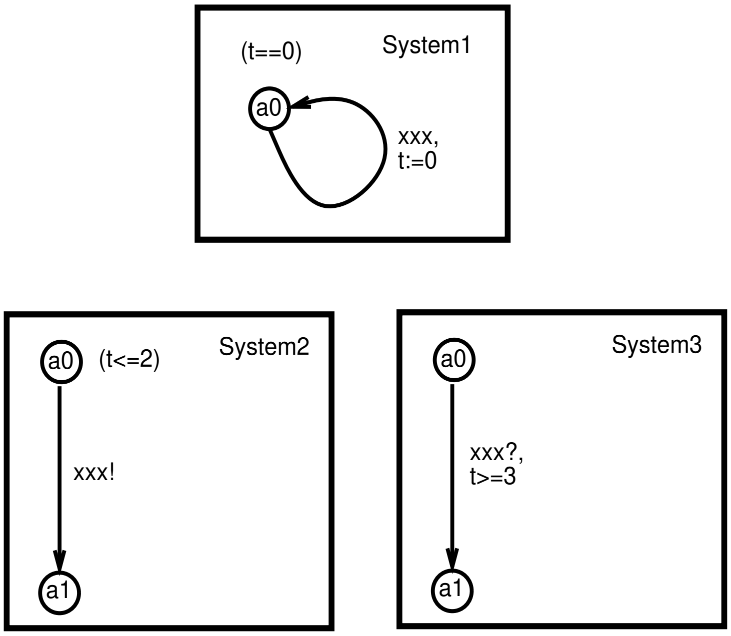

Figure 5: Timelock Illustrations

The natural interpretation of action lock in the setting of timed systems is as follows.

Definition 2 *A state $[l, v]$ of a TA A is an action lock, denoted $AL([l, v])$, if and only if,*

$$\forall t \in \mathbb{R}^{+0} \; ([l, v+t] \in [\![\, A \,]\!] .1 \implies [l, v+t] \overset{e}{\not\Longrightarrow})$$

where $[l, v+t] \in [\![\, A \,]\!] .1$ implies $[l, v+t]$ is reachable from $[l, v]$ by the definition of $[\![\,]\!]$.

The timed automaton A contains an action lock if and only if $\exists s \in [\![\, A \,]\!] .1 \,.\, AL(s)$.

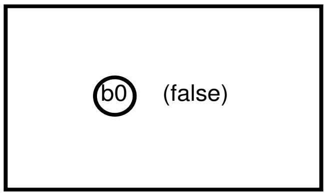

Figure 6: A Trivial Time Action Lock

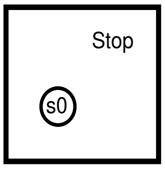

Figure 7: A Trivial Action Lock

Thus, a timed automaton is action locked when it reaches a state from which, however long time is allowed to pass, an action will never be possible. Notice also that if all guards are true and all invariants are true, we obtain the untimed case and action locks reduce to untimed "deadlocks".

2.3 A Bounded Timeout

As an illustrative specification example we will consider the description of a bounded timeout. This has been chosen because, firstly, it is one of the most common real-time specification scenarios and secondly, during timed automata specification and verification of a lip-synchronisation algorithm [7] it was discovered that describing such bounded timeouts in a deadlock free manner was surprisingly difficult.

The general scenario is that a `Timeout` process is monitoring a `Component` and the timeout should expire and enter an error state if the `Component` does not offer a particular action, which we call `good`, within a certain period of time.

The precise functionality that we want the timeout to exhibit is[7]:

1. *Basic behaviour.* Assuming `Timeout` is started at time t, it should generate a `timeout` action at a time $t + D$ if and only if the action `good` has not already occured. Thus, if action `timeout` occurs, it must occur exactly at time $t + D$ and if action `good` occurs, then it must occur at some time from t up to, but not including, $t + D$. Using the terminology of [10] this yields a *strong* timeout. A *weak* timeout would, in contrast, allow a non-deterministic choice between the `good` action and the `timeout` at time $t + D$.

2. *Urgency of* `good` *action.* We also require that if the `good` action is enabled before time $t + D$ then it is taken *urgently*, i.e. as soon as `good` is enabled it happens.

3. *Timelock Free.* Finally we want our composed system to be free of timelocks, for obvious reasons.

[7] Our presentation here is similar to that in [7]. However, although our work here was inspired by that in [7], it is somewhat different. In particular, [7] presents a bounded timeout in a discrete time setting, thus, the final time at which the `good` action can be performed and the time of expiry of the `Timeout` are at different discrete time points.

4. *Simple.* We also require that the solution is not "prohibitively" complex.

Notice that in the first two of these requirements, urgency arises in two ways. Firstly, we require that `timeout` is urgent at time $t+D$ and secondly, we require that `good` is urgent as soon as it is enabled. Without the former requirement the timeout might fail to fire even though it has expired and without the latter, even though the `good` action might be able to happen it might nonetheless not occur and thus, for example, the `timeout` may expire even though `good` was possible.

3 Timelocks

This section considers the issue of timelocks. We begin in subsection 3.1 by considering how to ensure zeno lock freeness based on an approach of Tripakis [12]. Then we move to the more difficult issue of time action locks. We further motivate the problem with time action locks, in subsection 3.2, by considering the specification of the bounded timeout example. Then we argue in subsection 3.3 that the timed automata interpretation of synchronisation should be adapted and we consider possible approaches to do this, including only allowing urgency on internal actions. However, this fails to be a suitably expressive approach and thus, subsection 3.4 considers a revised timed automata framework, due to Bornot and Sifakis, called Timed Automata with Deadlines (TADs). However, specification of the bounded timeout reveals a problem with the TADs framework as it was presented in [3, 4]. We revise the framework in subsection 3.5 in order to resolve this difficulty.

3.1 Zeno Timelocks

As highlighted earlier, zeno timelocks are situations in which an infinite number of discrete transitions are performed in a finite period of time. In contrast to the approach we will present for tackling time action locks, to handle zeno timelocks we will not define a new parallel composition operator. In contrast, we will consider a static construction which ensures zeno timelock freeness.

The standard approach to obtaining zeno timelock freeness in timed process algebra is to ensure that all recursions are time guarded, i.e. that all processes can pass time by at least $\epsilon \in \mathbb{R}^+$ between each recursive invocations. This provides a static mechanism that specifiers can use to ensure zeno timelock freeness.

The approach we advocate in the timed automata setting has a similar flavour. The idea is to ensure that for each loop in an automaton, time must pass by at least ϵ on every iteration.

We follow closely the presentation in [12]. Firstly two definitions.

Definition 3 *For $A \in TA$ we define a structural loop to be a sequence of distinct transitions,*

$$l_0 \xrightarrow{e_1,g_1,r_1} l_1 \xrightarrow{e_2,g_2,r_2} .... \xrightarrow{e_n,g_n,r_n} l_n$$

such that $l_0 = l_n$.

Definition 4 *$A \in TA$ is called strongly non-zeno if, for every structural loop,*

$$l_0 \xrightarrow{e_1,g_1,r_1} l_1 \xrightarrow{e_2,g_2,r_2} \ldots\ldots \xrightarrow{e_n,g_n,r_n} l_n$$

there exists a clock $c \in A.5$, $\epsilon \in \mathbb{R}^+$ and $0 \leq i,j \leq n$ such that,

1. *$c \in r_i$; and*
2. *c is bounded from below in step j, i.e. $(c < \epsilon) \cap g_j = false$.*

Clearly, `System1` of figure 5 fails to be strongly non-zeno since a suitable $\epsilon \in \mathbb{R}^+$ does not exist. However, the automaton in figure 8 is stongly non-zeno.

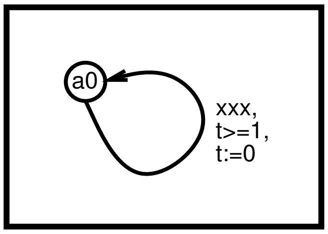

Figure 8: A Strongly Non-Zeno Specification

The following result was presented in [12].

Proposition 1 *If $A \in TA$ is strongly non-zeno then $Tr(A)$ does not contain a path that is both infinite and yields a timelock.*

In addition, strong non-zenoness is well behaved through parallel composition. Specifically, the following result was also presented in [12]. It ensures that we cannot generate new zeno timelocks through parallel composition.

Proposition 2 *If $A_1,...,A_n \in TA$ are strongly non-zeno then $|\langle A_1, ..., A_n\rangle$ is also strongly non-zeno.*

Also although we have no empirical evidence, in accordance with [12], we believe that in practice specifications will almost always be strongly non-zeno.

3.2 Trying to Model the Bounded Timeout

Now we move onto the issue of time action locks. As an illustration of the problem we describe the bounded timeout in timed automata.

Basic Formulation. We begin by considering the `Timeout` shown in figure 9. This process realises the first requirement that we identified for modelling the bounded timeout - `good` is offered at all times in which `t<D`. Then `timeout` is

performed when `t==D`, in which case the system passes into state `a2` which plays the role of an error state. Importantly, the guard `(t<=D)` forces the required urgency on the `timeout` action. Thus, if `good` has not happened earlier, `timeout` *must* happen when `t==D`. Furthermore, it is easy to see that this is indeed a strong timeout - its behaviour is deterministic when `t==D`.

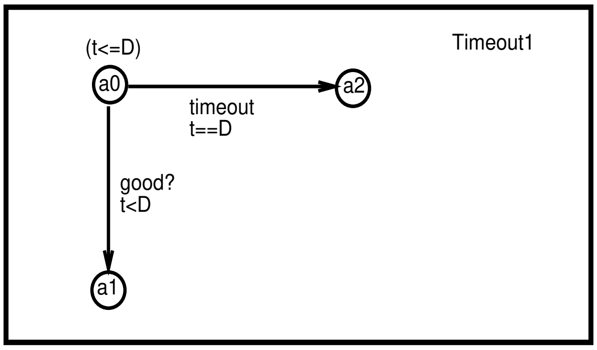

Figure 9: An Automaton for Timeout1

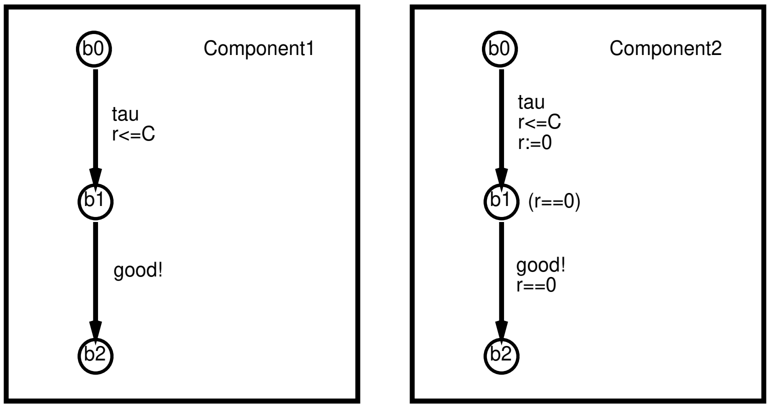

Figure 10: Automata for Component1 and Component2

However on its own, this automaton is not sufficient since nothing forces the `good` action to be taken if it can be. This was our second requirement. For example, consider `Component1` shown in figure 10 which will perform an internal action `tau` at some time `r<=C` and then offer the `good` action. The internal action can be viewed as modelling some internal computation by `Component1`. The completion of which is signalled by offering `good!`. Now if we put `Timeout1` and `Component1` in parallel then even if `good` could occur while `t<D`, it might not be taken. Thus, a possible evolution of the system:

```
|<Timeout1,Component1>
```

is, $(\mathtt{tau}, x_1)\,(\mathtt{timeout}, x_2)$ where, $x_1 < \mathtt{C}$, $x_1 < \mathtt{D}$ and $x_2 = \mathtt{D}$.

Thus, we need some way to make `good` urgent. The standard approach is to enforce urgency in the component. For example, we could use `Component2` shown in figure 10. This automaton will perform the internal action as before and then it *must* immediately perform the `good` action.

Now the problem with the composition:

```
|<Timeout1,Component2>
```

is the relative values of `D` and `C`. In particular, if `C` is larger than `D` then this system can timelock in the following way:-

1. the timeout could fire when `t==D`;

2. then if `tau` happens when `r==C` say, `good!` will become urgent, however it cannot be performed since `Timeout1` is no longer offering it, causing a timelock. `Component2` will not let time pass until `good` is performed, but `good` cannot be performed because of a mis-matched synchronisation.

We would argue that this is a big problem. In particular, it is not generally possible to ensure that `C` is less than `D` since our component behaviour would typically be embedded in the complex functioning of a complete system. In fact, writing `C` as we have done, abstracts from a likely multitude of complexity and deriving such a value from a system would typically require analysis of many components of the complete system, some of which might be time non-deterministic at the level of abstraction being considered.

Furthermore, in some situations we might actually be interested in analysing what happens if the good action arrives after the timeout has fired. Consider, for example, that our timeout behaviour is being used to wait for an acknowledgement in a sender process. The component performing `good` after `timeout` has fired corresponds to the acknowledgement arriving after the timeout has expired, which is of course a possible scenario in practical analysis of communication protocols.

The problem with our `|<Timeout1,Component2>` solution is that it does not enable us to analysis this situation, rather the system timelocks when `Component2` forces the `good` action to happen. Unfortunately, as mere mortals, we are unable to analyse systems after the end of time!

One way to avoid this timelock is to add "escape" transitions in the timeout. For example, consider the timeout behaviour encapsulated by `Timeout2`. Now the composition,

```
|<Timeout2,Component2>
```

cannot block time. However, this is not a satisfactory solution since rather than `Timeout2` just evolving to a single deadlock state, `a2`, after performing `timeout`, it could evolve to a complex behaviour; of course in practice it is

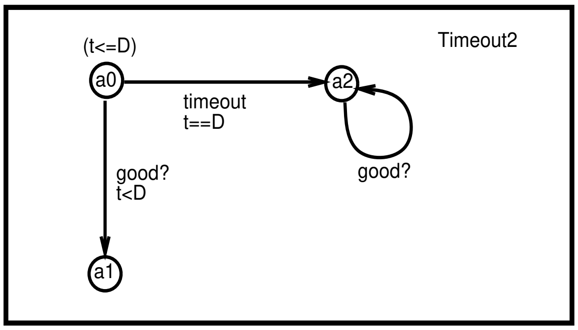

Figure 11: An Automaton for Timeout2

almost certain to do this. However then, escape transitions would have to be scattered throughout the complex behaviour. This would generate significant specification clutter, which would be compounded if the system contained more than one timeout.

The consequences become particularly severe if the timeout is enclosed in some repetitive behaviour, e.g. see figure 12. This is because, since no assumptions can be made about the time at which the component will want to perform the `good` action, escape transitions on `good` will have to be added at `a0`, `a2`, `b0`, `b1` (and actually `a1` as well). Thus, firstly, the behaviour prior to reaching the timeout has been altered, i.e. escape transitions must be added at `b0` and secondly, it is unclear how many escape transitions need to be added to each node in the loop, since state `a2` may be reached many times before the first `good` escape transition is performed.

Other Solutions. In [5] and [7] we have also considered other approaches to obtaining a satisfactory bounded timeout solution. In particular, we considered whether a suitable solution could be obtained using the UPPAAL notion of urgent channels. According to this model, the specifier is allowed to denote a particular channel as urgent, which means that as soon as synchronisation on that channel can take place, it does. However, UPPAAL restricts the use of such urgent channels. In particular, an urgent transition can only have the guard `true`.

Intuitively, urgent channels seem to be what we require in order to avoid enforcing urgency in the component process. In particular, they enforce urgency in a "global" manner, rather than requiring it to be enforced in the component process. However, it turns out that the restriction on guarding of urgent channels that UPPAAL imposes prevents derivation of a suitable solution, see [5] which investigates possible solutions with urgent channels which were inspired by the solutions presented in [7].

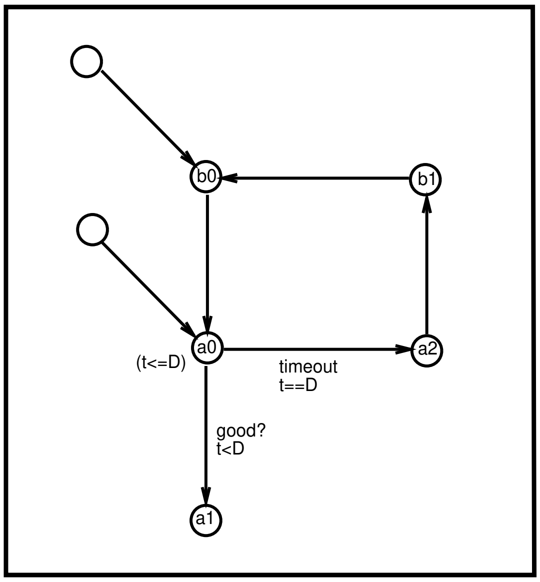

Figure 12: Timeout2 in a repetitive context

3.3 The Nature of Synchronisation

Perhaps the most counter-intuitive aspect of the timelock story is the manner in which timelocks can arise from mis-matched synchronisations, such as the composition `|<System2,System3>` in figure 5 and the timeout / component composition just highlighted. If we consider how this problem arises we can see that it is caused by the particular interpretation of urgent interaction employed in timed automata.

It is without doubt true that facilities to express urgency are required. In particular, if urgency is not supported, certain important forms of timing behaviour cannot be expressed. For example, as illustrated earlier, urgency plays a pivotal role in the formulation of the bounded timeout and indeed without it, it is unclear how one could describe timeouts in any vaguely sensible way.

Thus, it is necessary to include urgency in the timed automata model. However, it is our perspective that while urgency is needed, currently it is given an excessively strong formulation. We illustrate the issue with the following example.

Example 1 *Consider the specification of the Dying Dining Philosophers problem. The scenario is basically the same as the Dining Philosophers except here we have extra constraints which state that philosophers die if they do not eat within certain time periods.*

For example, if at a particular state, Aristotle must eat within 10 time units to avoid death, in timed automata his situation could be represented as state `l0` *of timed automata* `Aris` *in figure 13. In addition, if say the fork he requires is being used by another philosopher, the environment might not be able to satisfy this requirement. For example, the relevant global behaviour of the rest of the system might correspond to the behaviour of the automaton* `Rest` *in state* `m0` *(see figure 13 again).*

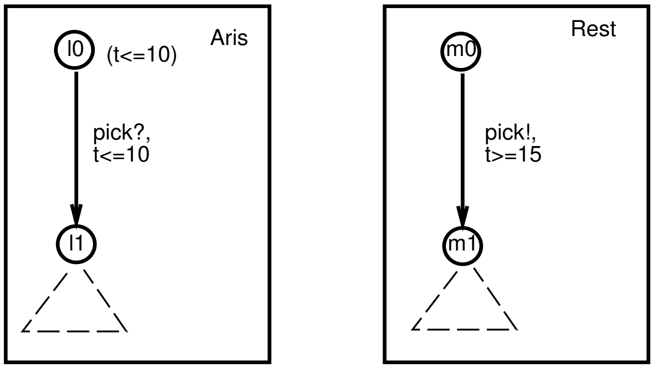

Figure 13: Dying Dining Philosophers Situation

In the present timed automata formulation the composition `|<Aris,Rest>` *will timelock when* `t` *reaches* `10`*. But, this seems counter-intuitive.*

`Aristotle` *knows he must pick-up his fork by a certain time otherwise drastic consequences will result for him (this is why he "registers" his* `pick` *request as urgent). However, if he locally fails to have his requirement satisfied, he cannot globally prevent the rest of the world from progressing, rather a local deadlock should result. As a consequence* `Aristotle` *might be dead, but as we all know, "the world will go on!".*

Conceptually what is happening is that `Aristotle` is enforcing that his `pick` action must be taken *even if it is not possible*, i.e. it is not enabled. However, we would argue that urgency can only be forced if an action is possible / enabled.

The situation is the same with our bounded timeout example - it is reasonable to state that `good` occurs urgently if both parties are able to perform it, but it is not reasonable to give urgency precedence over enabling. We would argue that it should only be possible to make an action urgent if it is enabled, i.e.

> *must requires may or, in other terms, you can only force what is possible.*

One way in which such an interpretation of urgency has previously been obtained is through only allowing urgency to be applied to internal actions. This is the so called *as soon as possible* (asap) principle [11], much discussed in the timed process algebra community. According to this principle internal actions are scheduled to occur as soon as they are possible, i.e. urgently, while, since they are subject to control by the environment, external actions (which closely correspond to our half actions) are not subject to such an interpretation - they can not be made urgent.

This property indeed prevents the occurrence of timelocks due to synchronisation mismatches, but unfortunately, it is not a suitable solution for timed automata. This is because TA do not have a hiding operator. In timed process algebra with asap the hiding operator, which turns observable into internal actions, has an important role since (implicitly) it makes actions urgent.

The absence of hiding in TA means that we cannot (selectively) take an observable action that results from synchronising half actions and turn it into an (urgent) internal action. This is for example what we would like to do with the synchronisation on the `good` action in our bounded timeout example.

Consequently, in the next section, we consider a new framework for timed automata specification - *Timed Automata with Deadlines* (TADs) which was initially devised by Bornot and Sifakis [3, 4] and with which we can obtain the synchronisation interpretation we desire.

3.4 Timed Automata with Deadlines

Components of the Framework. For a full introduction to TADs, we refer the interested reader to [3, 4]; here we highlight the main principles. The material and results included in this subsection borrow heavily from the previous work of Bornot and Sifakis. However, in our presentation we revise the Bornot and Sifakis definitions in order that they fit with the timed automata notation we are using and furthermore we present some new results that will be used in the sequel.

- *Deadlines on Transitions.* Rather than placing invariants on states, deadlines are associated with transitions. Transitions are annotated with 4-tuples:

  $$(e, g, d, r)$$

  where e is the transition label, e.g. `good`; g is the guard, e.g. `t<=D`; d is the deadline, e.g. `t==D`; and r is the reset set, e.g. `t:=0`. e, g and r are familiar from timed automata and the deadline is new. Conceptually, deadlines state when transitions *must* be taken and taken immediately. Since we have deadlines on transitions there is no need for invariants on states. Thus, they are not included in the framework.

  It is also assumed that the constraint,

  $$d \Rightarrow g$$

  holds, which ensures that if a transition is forced to happen it is also able to happen. Clearly, if this constraint did not hold then we could obtain timelocks because a transition is forced to happen, but it is not enabled.

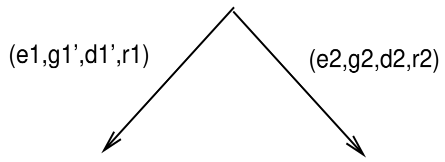

Figure 14: A Prioritised Choice

- *(Timewise) Priorities.* By restricting guards and deadlines in choice contexts, prioritised choice can be expressed. For example, if we have two transitions:

  $$\mathtt{b1} = (\mathtt{e1}, \mathtt{g1}, \mathtt{d1}, \mathtt{r1}) \quad \text{and} \quad \mathtt{b2} = (\mathtt{e2}, \mathtt{g2}, \mathtt{d2}, \mathtt{r2})$$

  then when placing them in a choice context we can give `b2` priority over `b1` by restricting the guards and deadlines of `b1`, see figure 14. [3] considers a variety of priority operators, which ensure that if the higher priority action will *eventually* be enabled within a particular period of time then it takes precedence over competing actions. These different priority mechanisms are obtained by including timed temporal operators in the restricted guards and deadlines. The extreme example of which is to enforce the following restricted guard and deadline:

  $$\mathtt{g1}' = \mathtt{g1} \ \wedge \ \Box\neg\mathtt{g2} \ \text{ and } \ \mathtt{d1}' = \mathtt{d1} \ \wedge \ \mathtt{g1}'$$

  which ensures that `b1` is only enabled if `g1` holds and there is *no point in the future* at which `g2` will hold.

- *Parallel Composition with Escape Transitions.* The TADs framework employs a different parallel composition operator to that arising in standard timed automata. The key idea is that of an *escape transition.* These are the local transitions of automaton components that are combined when generating a synchronisation transition. Thus, not only are synchronisations included, but component transitions of the synchronisation are as well. The timewise priority mechanism is then used to give the synchronisation transition highest priority. Intuitively, the escape transitions can only happen if the synchronisation transition will *never* be enabled. We will illustrate this aspect of TADs shortly.

- *Synchronisation Strategies.* [3] also consider a number of different synchronisation strategies, but these are not relevant to our discussion. In terms of [3] we only consider *AND* synchronisation.

In fact, in addition to ensuring time reactivity, the TADs framework limits the occurrence of action locks. Specifically, the escape transitions allow the components of a parallel composition to escape a potential action lock by evolving locally. Associated with such avoidance of action locks is the enforcement of *maximal progress*[8], which exactly requires that if a synchronisation is possible, it is always taken in preference to a corresponding escape transition.

Basic Definitions. We now briefly review the definition of timed automata with deadlines. In order to preserve some continuity through the paper, even though it is different to that used in [3], we build our definitions out of the timed automata notation and constructs that we have already introduced.

An arbitrary element of *TAD*, the set of timed automata with deadlines, has the form:

$$(L, l_0, \rightarrow, C)$$

where, L is a finite set of locations; l_0 is the *start location*; C is the set of clocks and

- $\rightarrow \subseteq L \times \mathbb{A} \times CC_C \times CC_C \times \mathbb{P}(C) \times L$ is a transition relation. A typical element of which is, (l_1, e, g, d, r, l_2), where $l_1, l_2 \in L$ are automata locations; $e \in \mathbb{A}$ labels the transition; $g \in CC_C$ is a guard; $d \in CC_C$ is a deadline; and $r \in \mathbb{P}(C)$ is a reset set. $(l_1, e, g, d, r, l_2) \in \rightarrow$ is typically written,

$$l_1 \xrightarrow{e,g,d,r} l_2$$

Also, as was the case in [3, 4], for technical reasons, we will require that all deadlines have closed lower bounds.

As was the case with TAs, TADs are semantically interpreted as transition systems. The following two inference rules are used for this,

$$(S1)\ \frac{l \xrightarrow{e,g,d,r} l' \quad g(v)}{[l,v] \stackrel{e}{\Longrightarrow} [l', r(v)]} \qquad (S2)\ \frac{\forall l' \,.\, l \xrightarrow{e,g,d,r} l' \implies \forall t' < t \,.\, \neg d(v + t')}{[l,v] \stackrel{t}{\Longrightarrow} [l, v + t]}$$

Now we define the semantic map $[\![\,]\!]$ from TADs to transition systems as follows[9]:

$$[\![(L, l_0, \rightarrow, C)\,]\!] = (S, s_0, \Rightarrow)$$

where,

- $s_0 = [l_0, v_0]$;
- $\Rightarrow$ is the subset of $(\mathbb{L} \times \mathbb{V}) \times Lab \times (\mathbb{L} \times \mathbb{V})$ that satisfies the above two rules; and
- $S = \{\, s' \in L \times \mathbb{V}_C \mid \exists s \in S,\, y \in Lab \,.\, s \stackrel{y}{\Longrightarrow} s' \,\} \cup \{\, [l_0, v_0]\,\}$.

[8]Note, the term is used in a related but somewhat different way in the timed process algebra setting [11].

[9]The overloading of $[\![\,]\!]$, i.e. to interpret both TAs and TADs, will not cause any confusion.

Notice that, once again, S only contains reachable states.

In addition, we will use the function:

$$\theta_B(l) = \{ (e, g, d, r) \mid \exists l' . l \xrightarrow{e,g,d,r} l' \ \wedge \ e \in B \}$$

Properties of TADs. Now we consider a number of basic properties of TADs. The first two are well known from previous TA and TADs work.

A standard property when considering dense time models is time continuity. We prove that TADs are time continuous in the following proposition.

Proposition 3 *(Time Continuity)*
$\forall A \in TAD . \forall s_1, s_2 \in [\![A]\!] .1 . \forall t, t' \in \mathbb{R}^+ .$

$$s_1 \stackrel{t+t'}{\Longrightarrow} s_2 \quad \Longleftrightarrow \quad \exists s_1' \, (s_1 \stackrel{t}{\Longrightarrow} s_1' \ \wedge \ s_1' \stackrel{t'}{\Longrightarrow} s_2)$$

Proof
Assuming that $s_1 = [l, v]$ and $s_2 = [l, v + t + t']$,

$$
\begin{array}{ll}
& [l, v] \stackrel{t+t'}{\Longrightarrow} [l, v + t + t'] \\
\Leftrightarrow & \{ \text{ Rule } (S_2) \text{ ; equivalence holds since no other rules overlap } \} \\
& \forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \forall y < t + t' . \neg d(v + y)) \\
\Leftrightarrow & \{ \text{ Distributivity of } \vee \text{ over } \forall \text{ ; } y \text{ does not appear on left of } \implies \} \\
& \forall l' \ \forall y < t + t' (l \xrightarrow{e,g,d,r} l' \implies \neg d(v + y)) \\
\Leftrightarrow & \{ \text{ Reordering quantifiers ; rewriting range } \} \\
& \forall y (y < t \ \vee \ t \leq y < t + t') \ \forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \neg d(v + y)) \\
\Leftrightarrow & \{ \text{ Range split } \} \\
& \forall y < t \ \forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \neg d(v + y)) \ \wedge \\
& \forall y (t \leq y < t + t') \ \forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \neg d(v + y)) \\
\Leftrightarrow & \{ \text{ Rearranging conjuncts } \} \\
& \forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \forall y < t . \neg d(v + y)) \ \wedge \\
& \forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \forall y < t' . \neg d(v + t + y)) \\
\Leftrightarrow & \{ \text{ Rule } (S_2) \ \} \\
& [l, v] \stackrel{t}{\Longrightarrow} [l, v + t] \ \wedge \ [l, v + t] \stackrel{t'}{\Longrightarrow} [l, v + t + t']
\end{array}
$$

○

Now we consider formally why the property $d \Rightarrow g$ is important. The following proposition shows that it guarantees time reactivity.

Proposition 4 *If $d \Rightarrow g$ on all transitions, TADs are time reactive.*

Proof
If a state $[l, v]$ can be reached such that

$$\forall t \in \mathbb{R}^+ . [l, v] \stackrel{t}{\not\Longrightarrow}$$

then the condition of the inference rule S_2 fails, i.e.,

$$\begin{array}{ll}
& \forall t \in \mathbb{R}^+ \neg\forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \forall t' < t . \neg d(v + t')) \\
\Leftrightarrow & \{ \text{ Logic } \} \\
& \forall t \in \mathbb{R}^+ \exists l' \, (l \xrightarrow{e,g,d,r} l' \;\wedge\; \exists t' < t . d(v + t')) \\
\Rightarrow & \{ \text{ Instantiating outer quantifier with an arbitrarily small } t \text{ } \} \\
& \exists l' \, (l \xrightarrow{e,g,d,r} l' \;\wedge\; \exists \epsilon . d(v + \epsilon)) \\
\Rightarrow & \{ \text{ Deadlines have closed lower bounds, thus, } \exists \epsilon . d(v + \epsilon) \Rightarrow d(v) \text{ } \} \\
& \exists l' \, (l \xrightarrow{e,g,d,r} l' \;\wedge\; d(v)) \\
\Rightarrow & \{ \; d \Rightarrow g \text{ ; logic } \} \\
& \exists l' \, (l \xrightarrow{e,g,d,r} l' \;\wedge\; g(v)) \\
\Rightarrow & \{ \text{ Rule } (S_1) \text{ } \} \\
& [l, v] \stackrel{e}{\Longrightarrow} [l', r(v)]
\end{array}$$

Thus, if any state cannot pass time, it can perform an action transition. The result follows.
○

We will use the following result later, it states that either time can pass forever, it cannot pass at all or there exists an upper bound beyond which time cannot pass.

Proposition 5
$\forall A \in \mathit{TAD} \, \forall s \in [\![A]\!].1$.

$$\begin{array}{l}
\forall t \in \mathbb{R}^+ . s \stackrel{t}{\Longrightarrow} \;\vee \\
\forall t \in \mathbb{R}^+ . s \stackrel{t}{\not\Longrightarrow} \;\vee \\
\exists t \in \mathbb{R}^+ \, (s \stackrel{t}{\Longrightarrow} \;\wedge\; \forall t' \in \mathbb{R}^+ \, (s \stackrel{t'}{\Longrightarrow} \iff t' \leq t))
\end{array}$$

Proof
First we can reason as follows:-

$$\begin{array}{ll}
& \forall t \in \mathbb{R}^+ . s \stackrel{t}{\Longrightarrow} \;\vee\; \forall t \in \mathbb{R}^+ . s \stackrel{t}{\not\Longrightarrow} \;\vee \\
& \exists t \in \mathbb{R}^+ \, (s \stackrel{t}{\Longrightarrow} \;\wedge\; \forall t' \in \mathbb{R}^+ \, (s \stackrel{t'}{\Longrightarrow} \iff t' \leq t)) \\
\Leftrightarrow & \{ \text{ Logic } \} \\
& \exists t \in \mathbb{R}^+ . s \stackrel{t}{\not\Longrightarrow} \implies (\exists t \in \mathbb{R}^+ . s \stackrel{t}{\Longrightarrow} \implies \\
& \exists t \in \mathbb{R}^+ \, (s \stackrel{t}{\Longrightarrow} \;\wedge\; \forall t' \in \mathbb{R}^+ \, (s \stackrel{t'}{\Longrightarrow} \iff t' \leq t)))
\end{array}$$

$$\Leftrightarrow \quad \{ \text{ Logic } \}$$

$$\exists t_1, t_2 \in \mathbb{R}^+ \, (s \overset{t_1}{\not\Longrightarrow} \ \wedge \ s \overset{t_2}{\Longrightarrow}) \implies$$
$$\exists t \in \mathbb{R}^+ \, (s \overset{t}{\Longrightarrow} \ \wedge \ \forall t' \in \mathbb{R}^+ \, (s \overset{t'}{\Longrightarrow} \iff t' \leq t))$$

and this is what we prove. So, assume that,

$$\exists t_1, t_2 \in \mathbb{R}^+ \, (s \overset{t_1}{\not\Longrightarrow} \ \wedge \ s \overset{t_2}{\Longrightarrow})$$

and take $s = [l, v]$ and $t \in \mathbb{R}^+$ as the smallest value (hence $t \leq t_1$) such that,

$$\exists l' \, (l \xrightarrow{e,g,d,r} l' \ \wedge \ d(v + t))$$

Such a t must exist since we have assumed that all deadlines have closed lower bounds.

Thus, we have,

$$\exists l' \, (l \xrightarrow{e,g,d,r} l' \ \wedge \ d(v + t)) \ \wedge \ \neg(\exists t' < t \, \exists l' \, (l \xrightarrow{e,g,d,r} l' \ \wedge \ d(v + t')))$$
$$\Leftrightarrow \quad \{ \text{ logic } \}$$
$$\exists l' \, (l \xrightarrow{e,g,d,r} l' \ \wedge \ d(v + t)) \ \wedge \ \forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \forall t' < t \, . \, \neg d(v + t'))$$

Now the second conjunct gives us $[l, v] \overset{t}{\Longrightarrow}$ and also because of time continuity we have,

$$\forall t' \, (t' \leq t \implies [l, v] \overset{t'}{\Longrightarrow})$$

Furthermore, if we take $t' \in \mathbb{R}^+$ such that $t' > t$ then (since t itself is a suitable value for t'') the first conjunct ensures that,

$$\forall t' > t \, \exists t'' < t' \, \exists l' \, (l \xrightarrow{e,g,d,r} l' \ \wedge \ d(v + t''))$$
$$\Leftrightarrow \quad \{ \text{ Interchange of existentials ; distributivity of } \wedge \text{ over } \exists \ \}$$
$$\forall t' > t \, \exists l' \, (l \xrightarrow{e,g,d,r} l' \ \wedge \ \exists t'' < t' \, . \, d(v + t''))$$
$$\Leftrightarrow \quad \{ \text{ Logic } \}$$
$$\forall t' > t \, \neg(\forall l' \, (l \xrightarrow{e,g,d,r} l' \implies \forall t'' < t' \, . \, \neg d(v + t'')))$$
$$\Leftrightarrow \quad \{ \text{ Rule } (S_2) \ \}$$
$$\forall t' > t \, . \, [l, v] \overset{t'}{\not\Longrightarrow}$$

The contrapositive of which is, $\forall t' \in \mathbb{R}^+ \, ([l, v] \overset{t'}{\Longrightarrow} \implies t' \leq t)$.

Now if we put everything together we obtain,

$$\exists t \in \mathbb{R}^+ \, (s \overset{t}{\Longrightarrow} \ \wedge$$
$$\forall t' \in \mathbb{R}^+ \, (t' \leq t \implies s \overset{t'}{\Longrightarrow}) \ \wedge$$
$$\forall t' \in \mathbb{R}^+ \, (s \overset{t'}{\Longrightarrow} \implies t' \leq t))$$

which is as required.
○

In addition, the following proposition gives an alternative characterisation of time reactivity.

Proposition 6
$\forall A \in TAD \,.\, A$ is time reactive if and only if,

$\forall [l,v] \in [\![A]\!].1$
$\exists t\,([l,v] \overset{t}{\not\Longrightarrow} \;\Longrightarrow\; \exists e\,([l,v] \overset{e}{\Longrightarrow} \;\vee\; \exists t' \leq t\,([l,v] \overset{t'}{\Longrightarrow} [l,v+t'] \;\wedge\; [l,v+t'] \overset{e}{\Longrightarrow})))$

Proof
($\Longrightarrow$)
Assume A is time reactive and take $[l,v] \in [\![A]\!].1$ such that $\exists t \,.\, [l,v] \overset{t}{\not\Longrightarrow}$. Now proposition 5 implies that either,

$$\forall t' \,.\, [l,v] \overset{t'}{\not\Longrightarrow} \quad \text{or} \quad \exists t'\,([l,v] \overset{t'}{\Longrightarrow} \;\wedge\; \forall t''\,([l,v] \overset{t''}{\Longrightarrow} \;\Longleftrightarrow\; t'' \leq t'))$$

Consider these in turn.

$$\begin{array}{ll}
 & \forall t' \,.\, [l,v] \overset{t'}{\not\Longrightarrow} \\
\Rightarrow & \{\ \text{Definition of time reactivity}\ \} \\
 & \exists e \in \mathbb{A} \,.\, [l,v] \overset{e}{\Longrightarrow}
\end{array}$$

which is as required. In addition,

$$\begin{array}{ll}
 & \exists t'\,([l,v] \overset{t'}{\Longrightarrow} \;\wedge\; \forall t''\,([l,v] \overset{t''}{\Longrightarrow} \;\Longleftrightarrow\; t'' \leq t')) \\
\Rightarrow & \{\ \text{Otherwise time continuity would give } [l,v] \overset{t}{\Longrightarrow}\ \} \\
 & \exists t' \leq t\,([l,v] \overset{t'}{\Longrightarrow} \;\wedge\; \forall t''\,([l,v] \overset{t''}{\Longrightarrow} \;\Longleftrightarrow\; t'' \leq t')) \\
\Rightarrow & \{\ \text{Otherwise time continuity causes contradiction of second conjunct}\ \} \\
 & \exists t' \leq t\,([l,v] \overset{t'}{\Longrightarrow} [l,v+t'] \;\wedge\; \forall r \in \mathbb{R}^+\,([l,v+t'] \overset{r}{\not\Longrightarrow})) \\
\Rightarrow & \{\ \text{Definition of time reactivity}\ \} \\
 & \exists t' \leq t\,([l,v] \overset{t'}{\Longrightarrow} [l,v+t'] \;\wedge\; [l,v+t'] \overset{e}{\Longrightarrow})
\end{array}$$

which is also as required.

($\Longleftarrow$)
Take $[l,v] \in [\![A]\!].1$, now,

$$\forall t \,.\, [l,v] \overset{t}{\not\Longrightarrow}$$

$$
\begin{array}{ll}
\Rightarrow & \{\ \mathbb{R}^+ \neq \emptyset\ \} \\
& \exists r \in \mathbb{R}^+ \,.\, [l,v] \stackrel{r}{\not\Longrightarrow} \ \wedge\ \forall t \,.\, [l,v] \stackrel{t}{\not\Longrightarrow} \\
\Rightarrow & \{\ \text{From Assumptions}\ \} \\
& \exists e\,([l,v] \stackrel{e}{\Longrightarrow} \ \vee\ \exists t' \leq r\,([l,v] \stackrel{t'}{\Longrightarrow} \ \wedge\ [l,v+t'] \stackrel{e}{\Longrightarrow})) \ \wedge\ \forall t \,.\, [l,v] \stackrel{t}{\not\Longrightarrow} \\
\Leftrightarrow & \{\ \text{Distributivity of Existentials}\ \} \\
& (\exists e \,.\, [l,v] \stackrel{e}{\Longrightarrow} \ \vee\ \exists e\, \exists t' \leq r\,([l,v] \stackrel{t'}{\Longrightarrow} \ \wedge\ [l,v+t'] \stackrel{e}{\Longrightarrow})) \\
& \wedge\ \forall t \,.\, [l,v] \stackrel{t}{\not\Longrightarrow} \\
\Leftrightarrow & \{\ \text{Distributivity of } \wedge \text{ over } \vee\ \} \\
& (\exists e \,.\, [l,v] \stackrel{e}{\Longrightarrow} \ \wedge\ \forall t \,.\, [l,v] \stackrel{t}{\not\Longrightarrow}) \ \vee \\
& (\exists e\, \exists t' \leq r\,([l,v] \stackrel{t'}{\Longrightarrow} \ \wedge\ [l,v+t'] \stackrel{e}{\Longrightarrow}) \ \wedge\ \forall t \,.\, [l,v] \stackrel{t}{\not\Longrightarrow}) \\
\Leftrightarrow & \{\ \text{Second disjunct is contradictory}\ \} \\
& \exists e \,.\, [l,v] \stackrel{e}{\Longrightarrow} \ \wedge\ \forall t \,.\, [l,v] \stackrel{t}{\not\Longrightarrow} \\
\Rightarrow & \{\ \text{Logic}\ \} \\
& \exists e \,.\, [l,v] \stackrel{e}{\Longrightarrow}
\end{array}
$$

which is as required.
○

Standard TADs. We will introduce a number of different TADs approaches in this paper. These are distinguished by their rules of parallel composition. Here we consider the basic approach, as introduced in [3, 4], which we call *standard TADs*. A TADs expansion theorem for deriving the product behaviour from a parallel composition is given in [3]. Here we give an equivalent inference rule definition for our state vector notation (we denote the standard TADs vector as $||\langle u[1], ...u[n]\rangle$):-

$$
(R1) \quad \frac{u[i] \xrightarrow{x?,g_i,d_i,r_i} u[i]' \quad u[j] \xrightarrow{x!,g_j,d_j,r_j} u[j]'}{\begin{array}{l} ||u \xrightarrow{x,g',d',r_i \cup r_j} ||u[i'/i, j'/j] \\ ||u \xrightarrow{x?,g_i',d_i',r_i} ||u[i'/i] \\ ||u \xrightarrow{x!,g_j',d_j',r_j} ||u[j'/j] \end{array}}
$$

where $1 \leq i \neq j \leq |u|$ and,

$$
\begin{array}{rcl}
g' & = & g_i \wedge g_j \\
d' & = & g' \wedge (d_i \vee d_j) \\
g_i' & = & g_i \wedge \Box\neg(g_i \wedge g_j) \\
d_i' & = & g_i' \wedge d_i \\
g_j' & = & g_j \wedge \Box\neg(g_i \wedge g_j) \\
d_j' & = & g_j' \wedge d_j
\end{array}
$$

$$(R2)\quad \frac{u[i] \xrightarrow{e,g,d,r} u[i]' \qquad \neg(e \in HA \wedge \exists k \neq i \,.\, u[k] \overset{\overline{e}}{\Longrightarrow})}{||u \xrightarrow{e,g,d,r} ||u[i'/i]}$$

where $1 \leq i \leq |u|$. (R1) generates synchronisation and escape transitions with the constrained guards and deadlines ensuring that synchronisation has priority in the required manner. (R2) is the interleaving rule, which is straightforward apart from the second condition which ensures that transitions on incomplete actions are only generated by this rule if synchronisation, and hence rule (R1), is not possible.

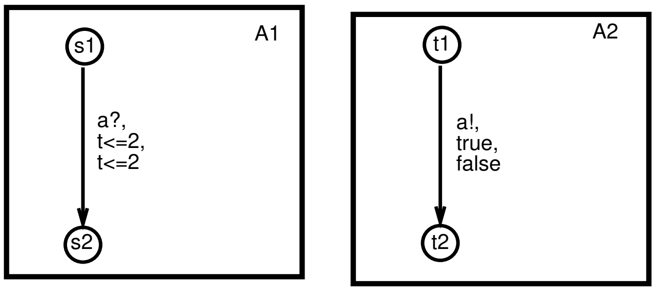

Figure 15: TADs `A1` and `A2`

As an illustration of these inference rules consider `||<A1,A2>` where `A1` and `A2` are shown in figure 15. The unreduced composition arising from directly applying the inference rules is shown in figure 16(a) ($\Box$ is denoted `[]` and $\neg$ is denoted $\sim$) and figure 16(b) depicts the resulting composed TAD when guards and deadlines have been reduced by expanding out temporal operators and applying propositional logic. In addition, transitions with unfulfillable guards, e.g. `false`, have been removed.

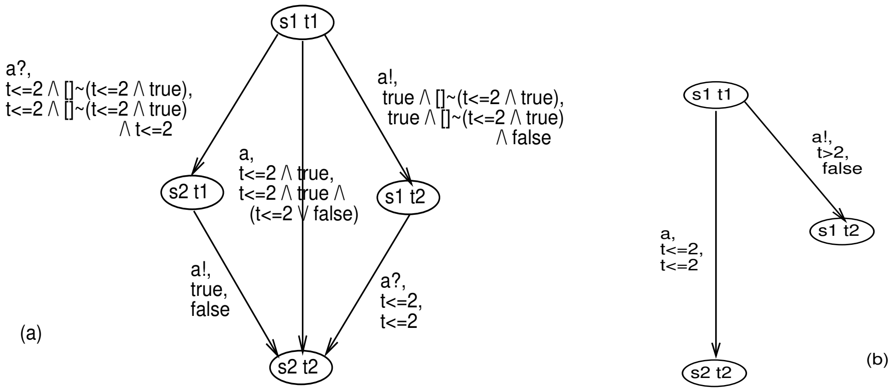

Figure 16: Unreduced and reduced composition of `A1` and `A2`

We can observe the following:-

1. In figure 16(a) and (b) the transition coming from `s1 t1` labelled `a` is the synchronisation transition.

2. In figure 16(a) the two transitions coming from `s1 t1` labelled `a?` and `a!` respectively, are the escape transitions. The first arises from automaton `A1` and the second from automaton `A2`. The guards of these escape transitions ensure that they can only fire if the synchronisation will never be possible in the future. Thus, synchronisation transitions have priority over escape transitions.

3. Figure 16(b) shows that since the synchronisation transition inherits the guards of `a?` from `A1`, no escape transition on `a?` is possible. If `s1 t1` is entered with `t>2` then the escape transition on `a!` can be taken, enabling `A2` to escape its action lock.

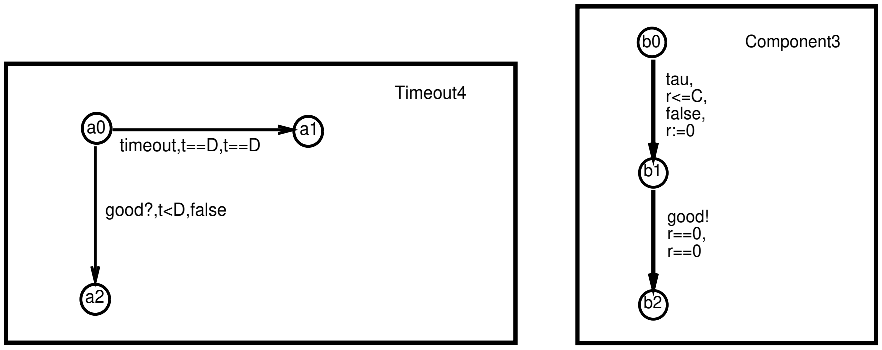

Figure 17: TADs for `Timeout4` and `Component3`

Bounded Timeout in Standard TADs. Now we reformulate our bounded timeout in standard TADs. The component that we consider is `Component3` and the timeout is `Timeout4` both shown in figure 17.

In the terminology of [3], a transition such as `good?` is *lazy* since nothing ever forces it to happen. In contrast, the transition `good!`, say, is *eager* [3], since its guard and deadline are the same. This implies that as soon as the transition can happen it will happen.

Now by applying the above inference rules and removing impossible transitions, the composite automaton shown in figure 18 results.

If we first focus on state `a0 b1` then we can see that this composite behaviour gives priority to the synchronisation between `good?` and `good!` which is indicated by the transition labelled `good`. Thus, while `t<D` this is the only transition that can fire (notice `r==0` automatically when entering state `a0 b1`) and furthermore it is eager.

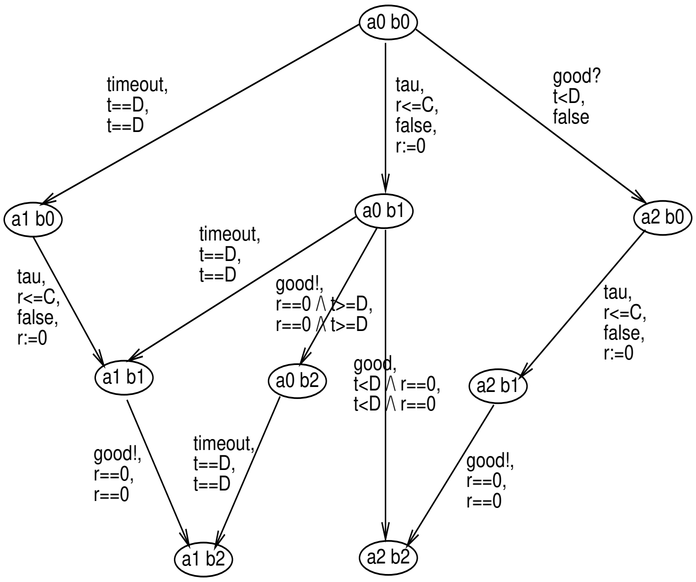

Figure 18: `||<Timeout4,Component3>` in standard TADs

Also, if state `a0 b1` is entered with `t==D` then `timeout` is urgent. Furthermore, from this state the action `good!` happens. This is the escape transition, which allows `Component3` to move out of state `b1`. Remember the timelock that we obtained previously arose because the component could not exit the state where it wished to perform `good!`.

This solution seems to fulfil our requirements - it is a strong timeout, urgency is enforced as required on both `timeout` and `good` and the solution is timelock free. However, there are some peculiarities with the resulting composite behaviour. Consider for example, the transition from `a0 b0` labelled `good?`. This represents the timeout performing its `good` escape transition. However, conceptually it is being performed too early - before the synchronisation on `good` is even offered and if this transition is taken the `good` synchronisation does not even have the chance to occur. The problem is the rule *(R2)* which adds escape transitions too liberally. In response to this observation we consider alternative TADs formulations in the next section.

3.5 Alternative TAD Formulations

We consider two alternative TAD formulations[10]. [5] actually considers a third formulation, but this turns out to be unsatisfactory. Both satisfy the require-

[10] We still call these timed automata with deadlines, because the basic principles, as concieved by Bornot et al [3, 4], still apply, i.e. placing deadlines on transitions and using prioritised choice.

ments that we identified for our bounded timeout. Thus, in particular, they are both time reactive. However, the solutions vary in the extent to which they limit action locks.

3.5.1 Sparse Timed Automata with Deadlines

This is a minimal TADs approach, in which we do not generate *any* escape transitions. Furthermore, since escape transitions are not generated, we do not have to enforce any priority between the synchronisation and escape transitions.

With sparse TADs the following parallel composition (denoted $||^s$) rules are used:

$$\frac{u[i] \xrightarrow{x?,g_i,d_i,r_i} u[i]' \quad u[j] \xrightarrow{x!,g_j,d_j,r_j} u[j]'}{||^s u \xrightarrow{x,g',d',r_i \cup r_j} ||^s u[i'/i, j'/j]} \qquad \frac{u[i] \xrightarrow{x,g,d,r} u[i]' \quad x \in CA}{||^s u \xrightarrow{x,g,d,r} ||^s u[i'/i]}$$

where $1 \leq i \neq j \leq |u|$, $g' = g_i \wedge g_j$ and $d' = g' \wedge (d_i \vee d_j)$.

These rules prevent uncompleted actions from arising in the composite behaviour; they only arise in the generation of completed actions, while (already) completed actions offered by components of the parallel composition can be performed independently. This definition has the same spirit as the normal UPPAAL rules of parallel composition [2]. The difference being that here we have deadlines which we constrain during composition to preserve the property $d \Rightarrow g$, and hence to preserve time-reactivity.

Let us consider once again the behaviour,

$||^s$`<Timeout4,Component3>`

which is the network we were focussing on in the previous section. Now with our new parallel composition rules, we obtain the composite behaviour shown in figure 19. This is an interesting and very reasonable solution. Firstly, it meets all the requirements identified at the start of this paper for our bounded timeout. Thus, in particular, it is time-reactive. However, it makes no effort to limit action locks, so communication mis-matches yield action locks rather than timelocks.

Furthermore as a consequence of these characteristics of sparse TADs we have revised the interpretation of synchronisation in the manner we proposed in subsection 3.3. For example, if we consider again the Dying Dining Philosophers illustration from that subsection, the obvious TADs formulation of the automata of figure 13 are those shown in figure 20. Now sparse TADs composition of the two automata yields the behaviour shown in figure 21, which is action locked.

This is the outcome that we were seeking - since the `pick` synchronisation is not enabled, urgency cannot be enforced. This is reflected in both the guard and deadline in figure 21 being *false*. This, in turn, is caused by the deadline constraint $d' = g' \wedge (d_i \vee d_j)$ in the Sparse TADs product rule, whereby the generated deadline is "pruned" according to the enabling of the guard.

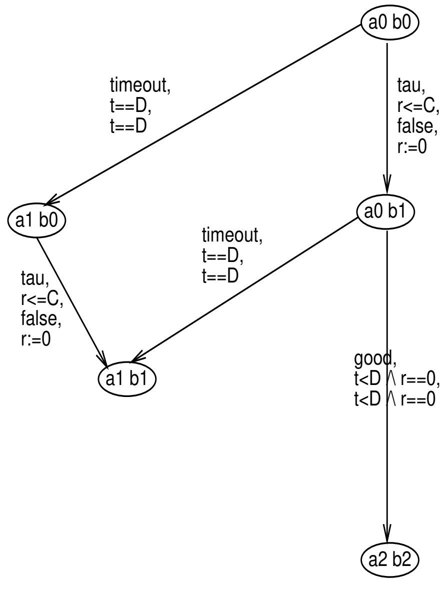

Figure 19: ||^s<Timeout4,Component3> in Sparse TADs

3.5.2 TADs with Minimal Priority Escape Transitions

The idea here is to ensure maximal progress as standard TADs do, but rather than just giving escape transitions lower priority than their corresponding synchronisation, we also give them lower priority than other completed transitions. Thus, a component can only perform an escape transition if the component will never be able to perform a completed transition. This seems appropriate as our view of escape transitions is that they should only be performed as a very last resort - when the choice is between performing them or reaching an "error" state.

The parallel composition (denoted $||^m$) rules are:

$$(R1) \quad \frac{u[i] \xrightarrow{x?,g_i,d_i,r_i} u[i]' \quad u[j] \xrightarrow{x!,g_j,d_j,r_j} u[j]'}{||^m u \xrightarrow{x,g',d',r_i \cup r_j} ||^m u[i'/i, j'/j]}$$

where, $1 \leq i \neq j \leq |u|$, $g' = g_i \wedge g_j$, $d' = g' \wedge (d_i \vee d_j)$. and,

$$(R2) \quad \frac{u[i] \xrightarrow{x,g,d,r} u[i]' \quad x \in CA}{||^m u \xrightarrow{x,g,d,r} ||^m u[i'/i]} \qquad (R3) \quad \frac{u[i] \xrightarrow{a,g,d,r} u[i]' \quad a \in HA}{||^m u \xrightarrow{a,g'',d'',r} ||^m u[i'/i]}$$

where, $1 \leq i \leq |u|$ and,

$$\begin{aligned} g'' \quad = \quad & g \wedge \bigwedge\{ \Box\neg q.2 \mid q \in \theta_{CA}(u[i]) \} \quad \wedge \\ & \bigwedge\{ \Box\neg(q.2 \wedge q'.2) \mid q \in \theta_{HA}(u[i]) \ \wedge \ q' \in \theta_{\overline{\{q.1\}}}(u[j]) \ \wedge \ 1 \leq j \neq i \leq |u| \} \end{aligned}$$

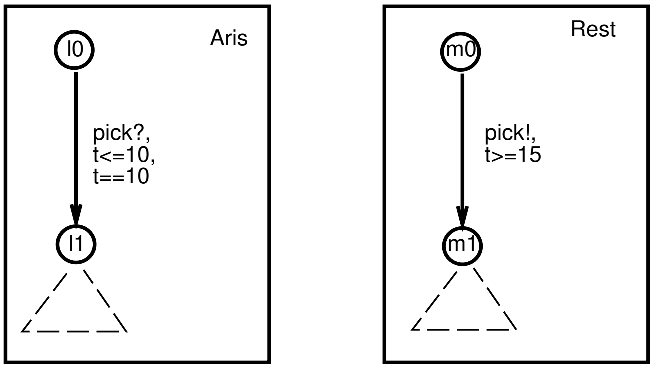

Figure 20: Dying Dining Philosophers Situation in TADs

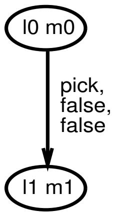

Figure 21: TADs Composition of Dying Dining Philosophers

$$d'' \quad = \quad d \ \wedge \ g''$$

(R1) is the normal synchronisation rule; *(R2)* defines interleaving of completed transitions; and *(R3)* defines interleaving of incomplete, i.e. escape, transitions. In this final rule, g'' holds when,

1. g holds; and
2. it is not the case that an already completed transition from $u[i]$ could eventually become enabled; and
3. it is not the case that an incomplete transition (including a itself) offered at state $u[i]$ could eventually be completed.

Furthermore, the definition of d'' ensures that the rules preserve the property $d \Rightarrow g$ and thus, the product is time reactive.

Applying these rules to the composition:

$||^m$`<Timeout4,Component3>`

and removing impossible transitions yields the composition shown in figure 22. This solution removes the excessively early escape transition from `a0 b0`, but

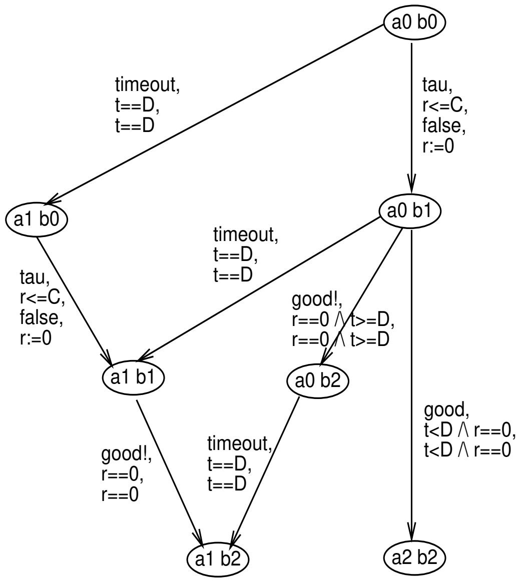

Figure 22: `||`^{m}`<Timeout4,Component3>` in TADs with minimum priority escape transitions

preserves all other transitions. In addition, we again obtain the "weaker" handling of urgency in synchronisation that subsection 3.3 proposed.

3.6 Discussion

This section has presented a number of means to "constructively" ensure timelock freeness in a timed automata setting. We can summarise our results as follows:-

- we highlighted a static specification device which can be used to ensure that zeno timelocks cannot arise;
- we considered the standard TADs framework. However, this proved unsatisfactory as it generated too many escape transitions;
- in response, we presented two new TADs formulations - Sparse TADs and TADs with minimal priority escape transitions, which do not allow time action locks to be generated and are thus, time reactive; and secondly, resolve the problem of escape transitions being generated excessively early;
- furthermore, the TADs parallel composition that we present "weakens" the interpretation of urgency in synchronisation. Specifically, we obtain a situation in which urgency can only be enforced if a synchronisation is possible.

You should also note that all these approaches are compositionally well behaved, in the sense that, if component automata satisfy the particular property, e.g. zeno timelock freeness or time action lock freeness, then the product will satisfy the same property.

4 Action Locks

One of the main results of the last section and of the TADs framework in general is to provide a means to compose automata together without generating timelocks. This then raises the issue of whether the same can be done for action locks, i.e. can a notion of parallel composition be defined which cannot introduce action locks. It turns out that by manipulating guards and deadlines appropriately such a notion of compositionality can be obtained. This is the subject matter of this section.

As an indication of the background to the problem of action locks we review the issue of action lock freeness in untimed systems in subsection 4.1. Then we consider a simple way to obtain action lock compositionality in subsection 4.2. However, this approach is very limited. Finally, in subsection 4.3 we consider a more satisfactory approach.

4.1 Independent Parallelism in Untimed Systems

We consider automata / transition systems, $(L, l_0, \rightarrow)$ where L is a set of locations, $\rightarrow$ is a transition relation on actions in $\mathbb{A}$ and $l_0 \in L$ is a start location.

Now we can define *untimed* action lock freeness. It is a straightforward extrapolation from (timed) action lock freeness which was definition 2.

Definition 5 *An automaton, $(L, l_0, \rightarrow)$ is action lock free iff*

$$\forall l \in L\, (l_0 \Rightarrow l \implies \exists e \in \mathbb{A} \,.\, l \xrightarrow{e}\,)$$

where $\Rightarrow$ is the obvious reachability relation, i.e.

$$l \Rightarrow l' \text{ iff } (l = l') \lor (\exists e_1, ..., e_n\, \exists l_1, ..., l_{n+1} \,.\, l_i \xrightarrow{e_i} l_{i+1} \;\land\; l = l_1 \;\land\; l' = l_{n+1})$$

Now we can easily identify a notion of parallel composition that preserves action lock freeness:-

$$(L_1, l_{1,0}, \rightarrow_1) \,|||\, (L_2, l_{2,0}, \rightarrow_2) = (L, l_0, \rightarrow)$$

where,

- $L = L_1 \times L_2$;
- $\rightarrow$ is defined by,

$$\frac{l_1 \xrightarrow{e}_1 l_1'}{(l_1, l_2) \xrightarrow{e} (l_1', l_2)} \qquad \frac{l_2 \xrightarrow{e}_2 l_2'}{(l_1, l_2) \xrightarrow{e} (l_1, l_2')}$$

- $l_0 = (l_{1,0}, l_{2,0})$

i.e. $|||$ gives the independent parallel composition of two automata.
$|||$ ensures the property,

> If either $(L_1, l_{1,0}, \rightarrow_1)$ or $(L_2, l_{2,0}, \rightarrow_2)$ are action lock free then so is $(L_1, l_{1,0}, \rightarrow_1) \,|||\, (L_2, l_{2,0}, \rightarrow_2)$

To prove this property we need a small lemma.

Lemma 1 *Assuming* $(L, l_0, \rightarrow) = (L_1, l_{1,0}, \rightarrow_1) \,|||\, (L_2, l_{2,0}, \rightarrow_2)$ *then,*

$$(l_{1,0}, l_{2,0}) \Rightarrow (l_1, l_2) \text{ implies } l_{1,0} \Rightarrow l_1 \ \wedge \ l_{2,0} \Rightarrow l_2$$

Proof
We prove just $l_{1,0} \Rightarrow l_1$, the other case is symmetric. $(l_{1,0}, l_{2,0}) \Rightarrow (l_1, l_2)$ implies $\exists e_1, ..., e_n \, \exists l_{1,1}, ..., l_{1,n+1} l_{2,1}, ..., l_{2,n+1} \, . \, (l_{1,0}, l_{2,0}) = (l_{1,1}, l_{2,1}) \wedge (l_1, l_2) = (l_{1,n+1}, l_{2,n+1}) \ \wedge \ (l_{1,i}, l_{2,i}) \xrightarrow{e_i} (l_{1,i+1}, l_{2,i+1})$. Now we work by induction.

Base Case. Assume $n = 1$. Then $(l_{1,0}, l_{2,0}) \xrightarrow{e_1} (l_1, l_2)$ and by the inference rules of $|||$ either $l_{1,0} = l_1$ or $l_{1,0} \xrightarrow{e_1}{}_1 l_1$, but in either case we are done.

Inductive Step. Assume the result holds for $n - 1$ and that $(l_{1,0}, l_{2,0}) \Rightarrow (l_{1,n+1}, l_{2,n+1})$ which implies $(l_{1,0}, l_{2,0}) \Rightarrow (l_{1,n}, l_{2,n}) \wedge (l_{1,n}, l_{2,n}) \xrightarrow{e_n} (l_{1,n+1}, l_{2,n+1})$ which by induction gives $l_{1,0} \Rightarrow l_{1,n}$ and by the inference rules gives us either $l_{1,n} = l_{1,n+1}$ or $l_{1,n} \xrightarrow{e_n}{}_1 l_{1,n+1}$, either of which gives us $l_{1,0} \Rightarrow l_{1,n+1}$ as required.
○

Proposition 7 *If* $(L_1, l_{1,0}, \rightarrow_1)$ *or* $(L_2, l_{2,0}, \rightarrow_2)$ *are action lock free then so is* $(L, l_0, \rightarrow) = (L_1, l_{1,0}, \rightarrow_1) \,|||\, (L_2, l_{2,0}, \rightarrow_2)$.

Proof
Wlog assume $(L_1, l_{1,0}, \rightarrow_1)$ is action lock free. Take $(l_1, l_2) \in L$ such that $(l_{1,0}, l_{2,0}) \Rightarrow (l_1, l_2)$ then by lemma 1 we know that $l_{1,0} \Rightarrow l_1$ and also since $(L_1, l_{1,0}, \rightarrow_1)$ is action lock free we have $l_1 \xrightarrow{e}{}_1$. But then the inference rules immediately give us that $(l_1, l_2) \xrightarrow{e}$ and we are done.
○

However, independent parallelism is not very interesting because it does not allow any synchronisation. Unfortunately synchronisation brings the possibility that new action locks can be introduced in the product. For example, the CCS parallel composition operator would ensure action lock freedom preservation if you could ensure that only actions that successfully synchronise are restricted. However, restriction is a static operator and determining whether actions synchronise is a dynamic property. This is why we need to use the TADs priority mechanisms, because they enable us to define parallel composition where the choice between the transitions is tied to the dynamic evolution of the system. This point will become clearer in subsection 4.3. First though, in subsection 4.2, we show that the results for untimed independent parallelism that we have deduced in this subsection can be extrapolated to the timed setting.

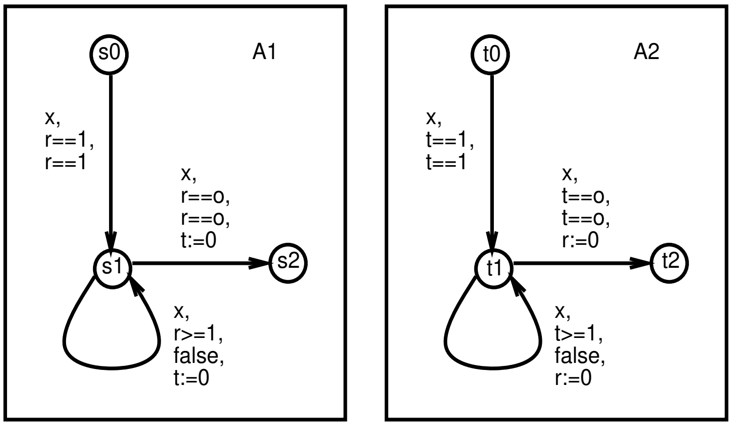

Figure 23: Automata illustrating the need for disjoint clock sets

4.2 Independent Parallelism in Timed Systems

We can easily define independent parallelism in the timed setting. Although it would be easy to give a definition for TA, here we give a definition for TADs. The independent product, denoted $||^i$, of a vector of TADs is defined,

$$||^i\langle A[1], ..., A[n]\rangle = (L, l_0, \longrightarrow, \bigcup_{1 \leq i \leq n} A[i].4)$$

where,

- $l_0 = ||^i\langle A[1].2, ..., A[n].2\rangle$;
- $L = \{\, l_0 \,\} \cup \{\, ||^i u' \mid ||^i u \xrightarrow{e,g,d,r} ||^i u' \ \wedge \ ||^i u \in L \,\}$;
- $\longrightarrow$ is defined by,

$$(RIP) \quad \frac{u[i] \xrightarrow{e,g,d,r} u[i]'}{||^i u \xrightarrow{e,g,d,r} ||^i u[i'/i]}$$

Once again since we assume $d \Rightarrow g$ throughout each component automaton, *(RIP)* ensures that $d \Rightarrow g$ in the independent product and thus we have time reactivity.

It turns out that in order to obtain action lock freeness preservation we will have to assume that the clock sets of our component automata are disjoint. The two automata in figure 23 indicate why we must make this assumption. Individually, these are both action lock free since once entering state `s1` (respectively `t1`) the clock `r` (respectively `t`) is already too high to allow the `s2` (respectively `t2`) branch. However, the independent product of the two will evolve to state `s2t2` (i.e. an action lock) since each resets the other's clock to zero.

Consequently, we will assume that the component automata in a vector have disjoint clock sets, i.e.

$$\langle A[1], ..., A[n]\rangle \text{ is only defined if, } \forall i, j(1 \leq i, j \leq n) \,.\, A[i].4 \cap A[j].4 = \emptyset$$

Now we introduce some notation related to disjointness of clock sets.

$$v\lceil C$$

is the restriction of the (larger) clock valuation v to the valuation on clocks of C, i.e.

$$v\lceil C = v \cap (C \times \mathbb{R}^{+0})$$

We can also build up larger clock valuations from smaller ones by taking the union of the two functions (note, disjointness of clock valuations prevents this from being dangerous). Also, we will often write $v\lceil(A[i].4)$ as $v\lceil^i$, i.e. to restrict the valuation v to the clocks of $A[i]$.

We have the following two straightforward lemmas concerning restriction of clock sets.

Lemma 2
$C' \subseteq C \implies C'(v\lceil C) = (C'(v))\lceil C$

Proof

$$
\begin{aligned}
& C'(v\lceil C) \\
= & \quad \{ \text{ Definitions of } \lceil \text{ and clock reset } \} \\
& ((v \cap (C \times \mathbb{R}^{+0}))\backslash(C' \times \mathbb{R}^{+0})) \cup (C' \times \{0\}) \\
= & \quad \{ \text{ Definition of } \backslash \ \} \\
& ((v \cap (C \times \mathbb{R}^{+0})) \cap (\mathbb{C}\backslash C' \times \mathbb{R}^{+0})) \cup (C' \times \{0\}) \\
= & \quad \{ \text{ Associativity and commutativity of } \cap \ \} \\
& ((v \cap (\mathbb{C}\backslash C' \times \mathbb{R}^{+0})) \cap (C \times \mathbb{R}^{+0})) \cup (C' \times \{0\}) \\
= & \quad \{ \text{ Distributivity of } \cup \text{ over } \cap \text{ ; definition of } \backslash \ \} \\
& ((v\backslash(C' \times \mathbb{R}^{+0})) \cup (C' \times \{0\})) \cap ((C \times \mathbb{R}^{+0}) \cup (C' \times \{0\})) \\
= & \quad \{ \ C' \subseteq C \ \} \\
& ((v\backslash(C' \times \mathbb{R}^{+0})) \cup (C' \times \{0\})) \cap (C \times \mathbb{R}^{+0}) \\
= & \quad \{ \text{ Definitions of } \lceil \text{ and clock reset } \} \\
& (C'(v))\lceil C
\end{aligned}
$$

○

Lemma 3
$(v\lceil C) + t = (v + t)\lceil C$

Proof
Trivial.
$\bigcirc$

We will need the following lemma. It states that if the independent product can reach a state then all components can reach a corresponding state. In particular, this correspondence ensures that all clock valuations that the independent product can reach, can (with appropriate restriction) also be reached by all component automata.

Lemma 4
$\forall i(1 \leq i \leq |u|) \, . \, [\, ||^i u, v] \in [\![\, ||^i A \,]\!] .1 \implies [u[i], v\lceil^i] \in [\![\, A[i] \,]\!] .1$

Proof
We prove this by induction over the rules for generating time/action transition systems for TADs. Take $i \in \mathbb{N}$ such that $1 \leq i \leq n$ and $n = |u|$.

Base Case:
$[\, ||^i \langle A[1].2, ..., A[n].2\rangle, v_0] \in [\![\, ||^i A \,]\!] .1$ and $[A[i].2, v_0\lceil^i] \in [\![\, A[i] \,]\!] .1$ by construction.

Inductive Step:
Assume $[\, ||^i u, v] \in [\![\, ||^i A \,]\!] .1$ and $[u[i], v\lceil^i] \in [\![\, A[i] \,]\!] .1$ (this is the inductive hypothesis). We need to show that the next state reachable from $[\, ||^i u, v]$ also corresponds to a state in $[\![\, A[i] \,]\!] .1$. We argue by case analysis of the means by which $[\, ||^i u, v]$ can reach a new state.

Case 1 $[\ [\, ||^i u, v] \stackrel{e}{\Longrightarrow} [\, ||^i u', v'] \ ]$

$$
\begin{aligned}
& [\, ||^i u, v\,] \stackrel{e}{\Longrightarrow} [\, ||^i u', v'\,] \\
\Leftrightarrow \quad & \{ \text{ Rule } (S_1) \ \} \\
& ||^i u \xrightarrow{e,g,d,r} ||^i u' \ \wedge \ g(v) \ \wedge \ v' = r(v)
\end{aligned}
$$

Case 1.1 $[\ u[i] = u[i]' \ ]$

$$
\begin{aligned}
& u[i] = u[i]' \\
\Leftrightarrow \quad & \{ \text{ Clock sets are disjoint, i.e. } r \cap (A[i].4) = \emptyset \ \} \\
& [u[i]', v'\lceil^i] = [u[i], v\lceil^i] \\
\Rightarrow \quad & \{ \text{ By inductive hypothesis } \} \\
& [u[i]', v'\lceil^i] \in [\![\, A[i] \,]\!] .1
\end{aligned}
$$

which is as required.

Case 1.2 $[\ u[i] \neq u[i]' \ ]$

$$
\begin{aligned}
& u[i] \neq u[i]' \\
\Rightarrow \quad & \{ \text{ Rule } (RIP) \text{ ; case assumption } \} \\
& u[i] \xrightarrow{e,g,d,r} u[i]' \ \wedge \ g(v) \ \wedge \ v' = r(v)
\end{aligned}
$$

$\Rightarrow$ { Inductive hypothesis; disjoint clock sets gives $g(v\lceil i)$; rule (S_1) }

$[u[i], v\lceil^i] \overset{e}{\Longrightarrow} [u[i]', r(v\lceil^i)] \;\wedge\; v' = r(v)$

$\Leftrightarrow$ { Disjointness of clock sets, so $r \subseteq (A[i].4)$; Lemma 2 }

$[u[i], v\lceil^i] \overset{e}{\Longrightarrow} [u[i]', r(v)\lceil^i] \;\wedge\; v' = r(v)$

$\Rightarrow$ { Substitution }

$[u[i], v\lceil^i] \overset{e}{\Longrightarrow} [u[i]', v'\lceil^i]$

$\Rightarrow$ { Transition system construction }

$[u[i]', v'\lceil^i] \in [\![\, A[i] \,]\!].1$

which is as required.

Case 2 $[\; [\,||^i u, v] \overset{t}{\Longrightarrow} [\,||^i u, v+t] \;]$
We seek to show that

$$[u[i], v\lceil^i] \overset{t}{\Longrightarrow} [u[i], (v+t)\lceil^i]$$

which will require us to show that,

$$\forall u[i]' \,(u[i] \xrightarrow{e', g_i, d_i, r_i} u[i]' \implies \forall t' < t \,.\, \neg d_i(v+t'))$$

Thus, we take $u[i]' \in A[i].1$ such that $u[i] \xrightarrow{e', g_i, d_i, r_i} u[i]'$ and proceed as follows:-

$u[i] \xrightarrow{e', g_i, d_i, r_i} u[i]' \;\wedge\; [\,||^i u, v] \overset{t}{\Longrightarrow} [\,||^i u, v+t]$

$\Leftrightarrow$ { Rule (S_2) }

$u[i] \xrightarrow{e', g_i, d_i, r_i} u[i]' \;\wedge\; \forall (||^i u')\, (||^i u \xrightarrow{e, g, d, r} ||^i u' \implies \forall t' < t \,.\, \neg d(v+t'))$

$\Leftrightarrow$ { Rule (RIP) }

$||^i u \xrightarrow{e', g_i, d_i, r_i} ||^i u[i'/i] \;\wedge$

$\forall (||^i u')\, (||^i u \xrightarrow{e, g, d, r} ||^i u' \implies \forall t' < t \,.\, \neg d(v+t'))$

$\Rightarrow$ { Instantiating universal }

$\forall t' < t \,.\, \neg d_i(v+t')$

which gives us that,

$$\forall u[i]' \,(u[i] \xrightarrow{e', g_i, d_i, r_i} u[i]' \implies \forall t' < t \,.\, \neg d_i(v+t'))$$

but then by (S_2) and our inductive hypothesis, we have,

$$[u[i], v\lceil^i] \overset{t}{\Longrightarrow} [u[i], (v\lceil^i) + t]$$

and by lemma 3 it follows that,

$$[u[i], v\lceil^i] \overset{t}{\Longrightarrow} [u[i], (v+t)\lceil^i]$$

and hence,

$$[u[i], (v+t)\lceil^i] \in [\![\, A[i] \,]\!].1$$

which is as required and completes the inductive case.

○

Now we show that action lock freeness is indeed preserved when taking the independent product.

Proposition 8

$\exists i\,(1 \leq i \leq |A|)\,.\,A[i]$ is action lock free $\Longrightarrow ||^i A$ is action lock free.

Proof

We prove the contrapositive,

> $||^i A$ contains an action lock implies $\forall i\,(1 \leq i \leq |A|)\,.\,A[i]$ is action locked.

So, assume $||^i A$ contains an action lock, i.e.,

Property (*)

$\exists [\,||^i u, v] \in [\![\,||^i A\,]\!]\,.1\,.\,\forall t \in \mathbb{R}^{+0}\;([\,||^i u, v+t] \in [\![\,||^i A\,]\!]\,.1 \implies [\,||^i u, v+t] \stackrel{e}{\not\Longrightarrow}\,)$

Take i such that $1 \leq i \leq |A|$, we need to show that $A[i]$ is action locked. Now by lemma 4 we know that,

$$[u[i], v\lceil^i] \in [\![\,A[i]\,]\!]\,.1$$

and we will show that this state is action locked. We proceed by contradition. Thus, assume the state is not action locked. There are two possibilities:-

1. $[u[i], v\lceil^i] \stackrel{e}{\Longrightarrow}$ or
2. $\exists t\,([u[i], v\lceil^i] \stackrel{t}{\Longrightarrow} [u[i], v\lceil^i{+}t] \;\wedge\; [u[i], v\lceil^i{+}t] \stackrel{e}{\Longrightarrow}\,)$.

We consider these cases in turn.

Case 1 [$[u[i], v\lceil^i] \stackrel{e}{\Longrightarrow}$]

$$
\begin{array}{ll}
 & [u[i], v\lceil^i] \stackrel{e}{\Longrightarrow} \\
\Rightarrow & \{\ \text{Rule } (S_1)\ \} \\
 & u[i] \xrightarrow{e,g,d,r} \;\wedge\; g(v\lceil^i) \\
\Rightarrow & \{\ \text{Rule } (RIP)\text{; disjointness of clock sets}\ \} \\
 & ||^i u \xrightarrow{e,g,d,r} \;\wedge\; g(v) \\
\Rightarrow & \{\ \text{Rule } (S_1)\ \} \\
 & [\,||^i u, v] \stackrel{e}{\Longrightarrow}
\end{array}
$$

which would contradict property (*). Thus, this case is not possible.

Case 2

[$\exists t\,([u[i], v\lceil^i] \stackrel{t}{\Longrightarrow} [u[i], v\lceil^i{+}t] \;\wedge\; [u[i], v\lceil^i{+}t] \stackrel{e}{\Longrightarrow}\,)$]

Let us consider the behaviour of the independent product. Firstly, can it pass time by t?

Case 2.1 [$[\,||^i u, v] \stackrel{t}{\not\Rightarrow}$]

$$[\,||^i u, v] \stackrel{t}{\not\Rightarrow}$$

$\Rightarrow$ { Since $||^i A$ will be time reactive we can use proposition 6 }

$$\exists e\,([\,||^i u, v] \stackrel{e}{\Longrightarrow} \;\vee\; \exists t' \le t\,([\,||^i u, v] \stackrel{t'}{\Longrightarrow} [\,||^i u, v+t'] \;\wedge\; [\,||^i u, v+t'] \stackrel{e}{\Longrightarrow}))$$

which would contradict property (*). Thus, this case is not possible.

Case 2.2 [$[\,||^i u, v] \stackrel{t}{\Longrightarrow} [\,||^i u, v+t]$]

$$[\,||^i u, v] \stackrel{t}{\Longrightarrow} [\,||^i u, v+t]$$

$\Rightarrow$ { Introducing our case 2 assumption }

$$[\,||^i u, v] \stackrel{t}{\Longrightarrow} [\,||^i u, v+t] \;\wedge\; [u[i], v\lceil^i + t] \stackrel{e}{\Longrightarrow}$$

$\Leftrightarrow$ { Lemma 3 }

$$[\,||^i u, v] \stackrel{t}{\Longrightarrow} [\,||^i u, v+t] \;\wedge\; [u[i], (v+t)\lceil^i] \stackrel{e}{\Longrightarrow}$$

$\Rightarrow$ { Rule (S_1) }

$$[\,||^i u, v] \stackrel{t}{\Longrightarrow} [\,||^i u, v+t] \;\wedge\; u[i] \xrightarrow{e,g,d,r} u[i]' \;\wedge\; g((v+t)\lceil^i)$$

$\Rightarrow$ { Rule (RIP); disjointness of clock sets }

$$[\,||^i u, v] \stackrel{t}{\Longrightarrow} [\,||^i u, v+t] \;\wedge\; ||^i u \xrightarrow{e,g,d,r} ||^i u[i'/i] \;\wedge\; g(v+t)$$

$\Rightarrow$ { Rule (S_1) }

$$[\,||^i u, v] \stackrel{t}{\Longrightarrow} [\,||^i u, v+t] \;\wedge\; [\,||^i u, v+t] \stackrel{e}{\Longrightarrow}$$

which would also contradict property (*). Thus, this case is also not possible.

This completes all the possibilities that would arise if,

$$[u[i], v\lceil^i]$$

were not action locked and all these possibilities generate contradictions. Thus, it must be the case that the state is action locked and the result follows.
○

4.3 Timed Case with Synchronisation

4.3.1 Composition Rules

As stated earlier, independent parallelism is theoretically interesting, but such interaction free parallel composition is of limited value. Thus, here we consider how the same action lock compositionality property can be obtained but while allowing interaction between processes. Our definition builds upon the parallel composition arising in TADs with Minimum Priority Escape Transitions, which has a number of the required characteristics. However, it does not go far enough in its generation of escape transitions. Particularly in respect of preserving component deadlines. These issues will become clear shortly.

Consider the following composition rules where u is a vector of TADs locations. The product that is generated is denoted $||^a A$.

$$(RCA) \quad \frac{u[i] \xrightarrow{x?,g_i,d_i,r_i} u[i]' \quad u[j] \xrightarrow{x!,g_j,d_j,r_j} u[j]'}{||^a u \xrightarrow{x,g',d',r_i \cup r_j} ||^a u[i'/i, j'/j]}$$

where, $1 \leq i \neq j \leq |u|$, $g' = g_i \wedge g_j$, $d' = g' \wedge (d_i \vee d_j)$ and,

$$(RIA) \quad \frac{u[i] \xrightarrow{x,g,d,r} u[i]' \quad x \in CA}{||^a u \xrightarrow{x,g,d,r} ||^a u[i'/i]} \qquad (RHA) \quad \frac{u[i] \xrightarrow{a,g,d,r} u'[i] \quad a \in HA}{||^a u \xrightarrow{a,g'',d'',r} ||^a u[i'/i]}$$

where $(1 \leq i \leq |u|)$ and,

$$\begin{aligned}
g'' \quad &= \quad (g \wedge \bigwedge\{ \Box\neg q.2 \,|\, q \in \theta_{CA}(u[i]) \} \quad \wedge \\
&\quad\quad \bigwedge\{ \Box\neg(q.2 \wedge q'.2) \,|\, q \in \theta_{HA}(u[i]) \ \wedge \ q' \in \theta_{\overline{\{q.1\}}}(u[j]) \ \wedge \ j \neq i \}) \vee \\
&\quad\quad d'' \\
d'' \quad &= \quad d \ \wedge \ \bigwedge\{ \neg q.3 \,|\, q \in \theta_{CA}(u[i]) \} \quad \wedge \\
&\quad\quad \bigwedge\{ \neg(q.2 \wedge q'.2 \wedge (q.3 \vee q'.3)) \,|\, q \in \theta_{HA}(u[i]) \ \wedge \ q' \in \theta_{\overline{\{q.1\}}}(u[j]) \ \wedge \ j \neq i \}
\end{aligned}$$

Now we give an explanation of the components of the definition.

(RCA). This is the (now) familiar "conjunctive" synchronisation rule, with the deadline constraint ensuring that $d \Rightarrow g$ and thus preserving time reactivity.

(RIA). This gives the also familiar interleaving modelling of independent parallelism, i.e. non-synchronizing internal actions.

(RHA). This generates escape transitions in order to avoid action locks, with the guard and deadline constructions controlling when the escape transitions can occur. We justify our guard and deadline definitions as follows:-

1. The guard is constructed as a disjunction between the guard construction first proposed in [6] for escape transitions and re-iterated in subsection 3.5.2 and the deadline. We justify the guard based disjunct (i.e. the first) here. A later point justifies disjoining with the deadline.

   The basic idea of this first disjunct,

   $$\begin{aligned}
   &g \wedge \bigwedge\{ \Box\neg q.2 \,|\, q \in \theta_{CA}(u[i]) \} \quad \wedge \\
   &\bigwedge\{ \Box\neg(q.2 \wedge q'.2) \,|\, q \in \theta_{HA}(u[i]) \ \wedge \ q' \in \theta_{\overline{\{q.1\}}}(u[j]) \ \wedge \ j \neq i \}
   \end{aligned}$$

   is to enable the product to escape action locks resulting from mismatched synchronisations. As was motivated in subsection 3.5 the construction refines the escape transition construction presented by [3, 4]. It does this by constraining escape transitions to only occur when the component automaton from which the escape transition originates can never perform any other transition.

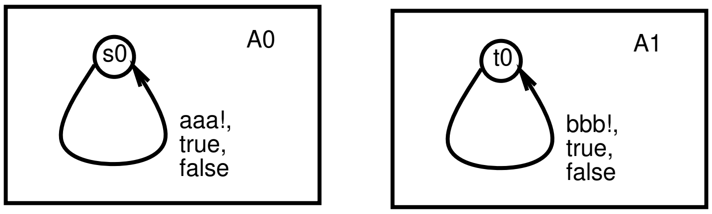

Figure 24: Action lock free TADs

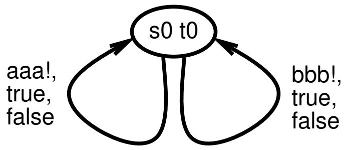

Figure 25: Composition of `A0` and `A1`

As a simple illustration of why this disjunct is required consider the two automata in figure 24. Both of these TADs are action lock free since in their single state they can always perform their respective transition and then evolve back into the same state.

However, if just rules *(RCA)* and *(RIA)* are used the composition of `A0` and `A1` will action lock immediately as neither synchronisation can be fulfilled. Also notice, this is not an issue of deadlines as both automata have unsatisfiable deadlines.

However, application of the rule *(RHA)* in conjunction with *(RCA)* and *(RIA)* will allow the action lock to be escaped as shown in the composition in figure 25. Thus, as a consequence of failing to synchronise, both automata evolve locally.

2. Now we justify the deadline construction in *(RHA)*. The construction,

$$d \;\wedge\; \bigwedge\{\, \neg q.3 \,|\, q \in \theta_{CA}(u[i]) \,\} \;\;\wedge$$
$$\bigwedge\{\, \neg(q.2 \,\wedge\, q'.2 \,\wedge\, (q.3 \,\vee\, q'.3)) \,|\, q \in \theta_{HA}(u[i]) \;\wedge\; q' \in \theta_{\overline{\{q.1\}}}(u[j]) \;\wedge\; j \neq i \,\}$$

has a similar shape to the guard construction we just considered, however, the temporal operators are not included. To explain the construction in words, it states that,

the deadline (d'') of the escape transition holds if and only if,

(a) the deadline of the corresponding component transition (d) holds;

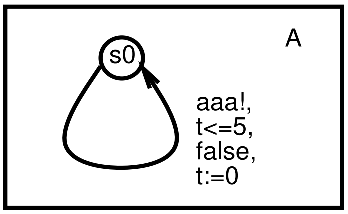

Figure 26: A strongly connected TAD that can action lock

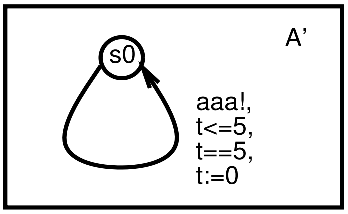

Figure 27: A strongly connected TAD that is action lock free

(b) no internal transition of the component is at its deadline; and

(c) no synchronisation which includes a half action of the component is at its deadline.

The intuition behind the rule is that any (non competing) deadline that appears in the component but that does not arise in the product (because of a failed synchronisation) has its deadline preserved in an escape transition of the product. A deadline of a transition is *competing* at a state if the deadline of an alternative transition also holds at that state.

This deadline construction is motivated by the observation that in the majority of cases it is the deadline that ensures action lock freeness of an automaton. For example, although the automaton `A` in figure 26 is strongly connected it is not action lock free. In particular, assuming `s0` is first entered with `t==0`, if it stays in state `s0` for longer than 5 time units, it will action lock.

Furthermore, there is nothing constraining the length of time the automaton can idle in state `s0` as the deadline of the `aaa!` transition is `false`. However, (assuming `s0` is entered with `t<=5`) the automaton shown in figure 27 is action lock free, since the deadline on the `aaa!` transition prevents excessive idling in state `s0`.

Now in order to obtain the action lock freeness property that we desire we need to guarantee that deadlines that ensure action lock freeness of component automata are preserved in the product (either through appearing

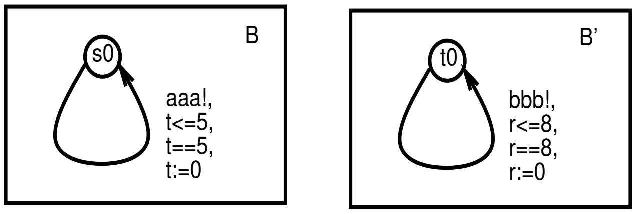

Figure 28: Two action lock free TADs

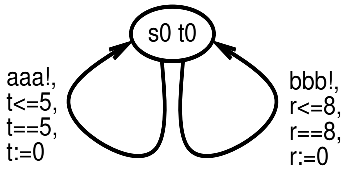

Figure 29: Composition of `B` and `B'`

as a result of rules *(RCA)* or *(RIA)* or by including relevant escape transitions). Our rule does this. Firstly, consider the two action lock free automata `B` and `B'` shown in figure 28. With just rules *(RCA)* and *(RIA)* the product of `B` and `B'` would be action locked. However, with *(RHA)* as well, the product automaton shown in figure 29 would result.

In fact, this product would have resulted from application of the rules presented in subsection 3.5 where the deadline is simply $d'' = d \wedge g''$. However, the example in figure 30 of two more action lock free TADs (`C0` and `C1`) shows that this is not sufficient in the general case. This is because according to the rules of subsection 3.5, the parallel composition of `C0` and `C1` would be as shown in figure 31 which will action lock at state `s1 t1`.

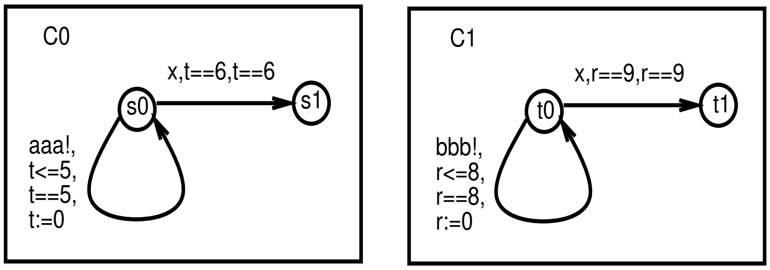

Figure 30: Two more action lock free TADs

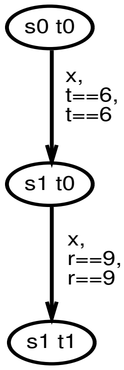

Figure 31: Composition of `C0` and `C1` without preserving deadlines

The problem is that the guards of the `aaa!` and `bbb!` escape transitions that the rules of subsection 3.5 generate, are false. This is because in both automata an internal action can eventually be taken and this internal action will take priority.

However, if we apply the rules *(RCA)*, *(RIA)* and *(RHA)* of the product $||^a$ then the left most product in figure 32 results which is "behaviourally equivalent" to the right most product. This is because the deadline prevents clock `t` passing `5` and clock `r` passing `8`. Notice that the guard has been pruned to match the deadline. This ensures that the enabling of `aaa!` and `bbb!` is minimised to only what is required to preserve the desired action lock freeness property.

Also notice that this example illustrates why the priority enforced in the deadline has to be immediate and including temporal operators is inappropriate. Specifically, if a deadline d ensures action lock freedom then even if later transitions are possible the deadline must be preserved exactly in the product in order to prevent later transitions from being enabled which allow an action lock to be reached, e.g. the internal transition above[11].

3. Finally, we need to disjoin the deadline in the guard in order to ensure that $d \Rightarrow g$ everywhere and thus to preserve time reactivity. For example, without such a disjunct, the product of `C0` and `C1` would be the composition shown in figure 33 which timelocks when `t` reaches `5`.

   Also notice that the standard approach, used e.g. by [3, 4], for obtaining $d \Rightarrow g$ which is to conjoin the guard with the deadline, will not work since it could remove some part of a deadline that is needed to ensure action

[11] This may not be the most refined solution since we might add an escape transition even though a later transition may prevent the action lock. But, such a more refined solution is very difficult to analyse, since you must be sure that the later deadline prevents an action lock and this is very difficult to analyse.

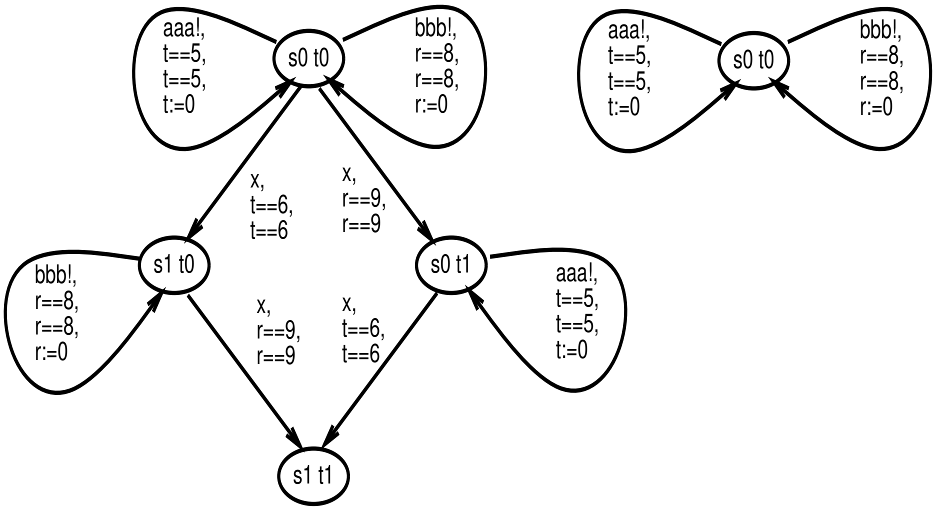

Figure 32: Composition of `C0` and `C1` with deadlines preserved

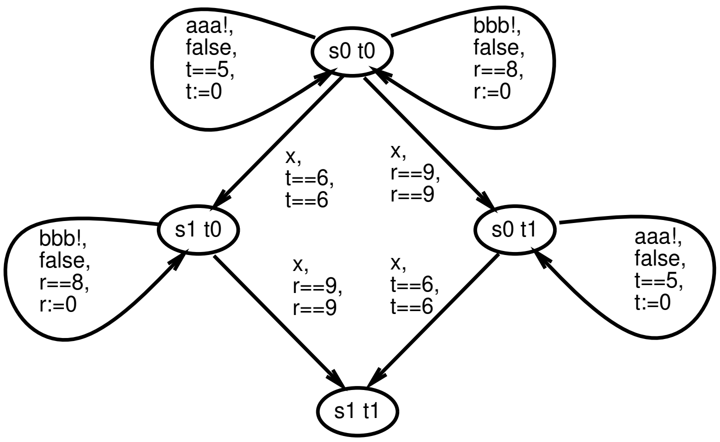

Figure 33: Non time reactive composition of `C0` and `C1`

lock freeness. This can again be seen in the above example. In particular, if we conjoined the guard,

$$g'' \;=\; g \wedge \bigwedge\{\, \Box\neg q.2 \,|\, q \in \theta_{CA}(u[i])\,\} \;\; \wedge$$
$$\bigwedge\{\, \Box\neg(q.2 \;\wedge\; q'.2) \,|\, q \in \theta_{HA}(u[i]) \;\wedge\; q' \in \theta_{\overline{\{q.1\}}}(u[j]) \;\wedge\; j \neq i\,\}$$

to the deadline,

$$d'' \;=\; d \;\wedge\; \bigwedge\{\, \neg q.3 \,|\, q \in \theta_{CA}(u[i])\,\} \;\; \wedge$$
$$\bigwedge\{\, \neg(q.2 \;\wedge\; q'.2 \;\wedge\; (q.3 \;\vee\; q'.3)) \,|\, q \in \theta_{HA}(u[i]) \;\wedge\; q' \in \theta_{\overline{\{q.1\}}}(u[j]) \;\wedge\; j \neq i\,\}$$

in order to ensure that $d \Rightarrow g$ then the deadlines of `aaa!` and `bbb!` would be false and the product could evolve to an action lock at state `s1 t1`.

4.3.2 Verification of Rules

Now we give a formal verification that the parallel composition $||^a$ does indeed preserve action lock freeness. Before coming to our main theorem, we need two results. The first is a simple consequence of a state being action locked in a TADs setting. It states that time can pass arbitrarily in any action locked state.

Proposition 9
$\forall A \in TAD \; \forall [l, v] \in [\![\, A \,]\!].1$
$([l, v]$ *is action locked* $\implies \forall t \in \mathbb{R}^+ \,.\, [l, v + t] \in [\![\, A \,]\!].1)$

Proof
Assume $[l, v]$ is action locked, i.e.

$$[l, v] \stackrel{e}{\not\Rightarrow} \;\wedge$$

$$\forall t' \in \mathbb{R}^+ \; ([l, v] \stackrel{t'}{\Longrightarrow} [l, v + t'] \implies [l, v + t'] \stackrel{e}{\not\Rightarrow}\,)$$

Now we know from proposition 5 that either,

$$\forall t \in \mathbb{R}^+ . \, [l, v] \stackrel{t}{\Longrightarrow} \;\vee$$
$$\forall t \in \mathbb{R}^+ . \, [l, v] \stackrel{t}{\not\Rightarrow} \;\vee$$
$$\exists t \in \mathbb{R}^+ \; ([l, v] \stackrel{t}{\Longrightarrow} \;\wedge\; \forall t' \in \mathbb{R}^+ \, ([l, v] \stackrel{t'}{\Longrightarrow} \iff t' \leq t))$$

Now disjunct 2 is not possible since if $[l, v]$ cannot pass time and cannot perform an action transition we have contradicted time reactivity. So, consider disjunct 3. We can reason as follows:-

$$\exists t\,([l,v] \overset{t}{\Longrightarrow} [l,v+t] \;\wedge\; \forall t'\,([l,v] \overset{t'}{\Longrightarrow} \iff t' \leq t))$$

$\Rightarrow$ { Logic }

$$\exists t\,([l,v] \overset{t}{\Longrightarrow} [l,v+t] \;\wedge\; \forall t' > t\,([l,v] \overset{t'}{\not\Longrightarrow}))$$

$\Leftrightarrow$ { Otherwise time continuity would contradict 2nd conjunct }

$$\exists t\,([l,v] \overset{t}{\Longrightarrow} [l,v+t] \;\wedge\; \forall t'\,.\,[l,v+t] \overset{t'}{\not\Longrightarrow})$$

But this yields a contradiction since $[l,v+t]$ cannot let time pass and (as $[l,v]$ is action locked) it cannot perform an action transition, which invalidates time reactivity.

Thus, our third disjunct is also impossible. This implies that the first disjunct must hold, i.e.

$$\forall t\,.\,[l,v] \overset{t}{\Longrightarrow} [l,v+t]$$

which is as required.

○

Now we consider the corresponding lemma to lemma 4 which we used to prove that the independent product preserved action lock freeness. The lemma states that if the product can reach a state then all components can reach a corresponding state. In particular, this correspondence ensures that all clock valuations that the product can reach, can (with appropriate restriction) also be reached by all component automata.

Lemma 5

$\forall i(1 \leq i \leq |u|)\,.\,[\,||^a u, v] \in [\![\, ||^a A \,]\!].1 \implies [u[i], v\lceil^i] \in [\![\, A[i] \,]\!].1$

Proof

We prove this by induction on the rules for generating transition systems from TADs. So, consider an arbitrary component automaton, say automaton $i \in \mathbb{N}$ such that $1 \leq i \leq n$,

Base Case:

If $[\langle A[1].2, ..., A[n].2\rangle, v_0] \in [\![\, ||^a A \,]\!].1$ then, by construction, we know that, $[A[i].2, v_0\lceil^i] \in [\![\, A[i] \,]\!].1$.

Inductive Step:

Assume $[\,||^a u, v] \in [\![\, ||^a A \,]\!].1$ and $[u[i], v\lceil^i] \in [\![\, A[i] \,]\!].1$ (this is the inductive hypothesis). We need to show that the next state reachable from $[\,||^a u, v]$ also corresponds to a state in $[\![\, A[i] \,]\!].1$. We argue via a case analysis of how a new state can be reached.

Case 1 $[\,[\,||^a u, v] \overset{e}{\Longrightarrow} [\,||^a u', v']\,]$

$$[\,||^a u, v] \overset{e}{\Longrightarrow} [\,||^a u', v']$$

$\Leftrightarrow$ { Rule (S_1) }

$$||^a u \xrightarrow{e,g,d,r} ||^a u' \;\wedge\; g(v) \;\wedge\; v' = r(v)$$

Case 1.1 [$u[i] = u[i]'$]

$$
\begin{array}{ll}
& u[i] = u[i]' \\
\Rightarrow & \{ \text{ Disjointness of clock sets, i.e. } r \cap (A[i].4) = \emptyset \ \} \\
& [u[i]', v'\lceil^i] = [u[i], v\lceil^i] \\
\Rightarrow & \{ \text{ Inductive hypothesis } \} \\
& [u[i]', v'\lceil^i] \in [\![A[i]]\!].1
\end{array}
$$

which is as required.

Case 1.2 [$u[i] \neq u[i]'$]

Case 1.2.1 [$e \in CA$ by an application of rule *(RIA)*]

$$
\begin{array}{ll}
& u[i] \neq u[i]' \ \wedge \ e \in CA \\
\Rightarrow & \{ \text{ Rule } (RIA) \text{ ; case assumption } \} \\
& u[i] \xrightarrow{e,g,d,r} u[i]' \ \wedge \ g(v) \ \wedge \ v' = r(v) \\
\Rightarrow & \{ \text{ Inductive hypothesis ; rule } (S_1) \text{ ; disjoint clock sets ensure } g(v\lceil^i) \ \} \\
& [u[i], v\lceil^i] \stackrel{e}{\Longrightarrow} [u[i]', r(v\lceil^i)] \ \wedge \ v' = r(v) \\
\Leftrightarrow & \{ \text{ Disjointness of clock sets, i.e. } r \subseteq A[i].4 \text{ ; lemma 2 } \} \\
& [u[i], v\lceil^i] \stackrel{e}{\Longrightarrow} [u[i]', r(v)\lceil^i] \ \wedge \ v' = r(v) \\
\Rightarrow & \{ \text{ Substitution } \} \\
& [u[i], v\lceil^i] \stackrel{e}{\Longrightarrow} [u[i]', v'\lceil^i] \\
\Rightarrow & \{ \text{ Transition system construction } \} \\
& [u[i]', v'\lceil^i] \in [\![A[i]]\!].1
\end{array}
$$

which is as required.

Case 1.2.2 [$e \in CA$ by an application of rule *(RCA)*]

$$
\begin{array}{ll}
& u[i] \neq u[i]' \ \wedge \ e \in CA \\
\Rightarrow & \{ \text{ Rule } (RCA) \text{ ; wlog use } e! \text{ rather than } e? \ \} \\
& u[i] \xrightarrow{e!,g_i,d_i,r_i} u[i]' \ \wedge \ (g_i \ \wedge \ g_j)(v) \ \wedge \ v' = (r_i \cup r_j)(v) \\
\Rightarrow & \{ \text{ Definition of guards } \} \\
& u[i] \xrightarrow{e!,g_i,d_i,r_i} u[i]' \ \wedge \ g_i(v) \ \wedge \ v' = (r_i \cup r_j)(v) \\
\Rightarrow & \{ \text{ Inductive hypothesis ; disjoint clock sets ; rule } (S_1) \ \} \\
& [u[i], v\lceil^i] \stackrel{e!}{\Longrightarrow} [u[i]', r_i(v\lceil^i)] \ \wedge \ v' = (r_i \cup r_j)(v) \\
\Leftrightarrow & \{ \text{ Lemma 2 ; } r_j \cap A[i].4 = \emptyset \text{ ; substitution } \} \\
& [u[i], v\lceil^i] \stackrel{e!}{\Longrightarrow} [u[i]', v'\lceil^i] \\
\Rightarrow & \{ \text{ Transition system construction } \} \\
& [u[i]', v'\lceil^i] \in [\![A[i]]\!].1
\end{array}
$$

which is as required.

Case 1.2.3 [$e \in HA$]

$$
\begin{array}{ll}
 & u[i] \neq u[i]' \ \wedge \ e \in HA \\
\Rightarrow & \{ \text{ Rule } \mathit{(RHA)} \text{ ; with } X \text{ and } Y \text{ s.t. } g = (g_i \ \wedge X) \vee (d_i \ \wedge Y) \ \} \\
 & u[i] \xrightarrow{e, g_i, d_i, r_i} u[i]' \ \wedge \ ((g_i \ \wedge X) \vee (d_i \ \wedge Y))(v) \ \wedge \ v' = r_i(v) \\
\Rightarrow & \{ \ d_i \Rightarrow g_i \text{ by time reactivity } \ \} \\
 & u[i] \xrightarrow{e, g_i, d_i, r_i} u[i]' \ \wedge \ g_i(v) \ \wedge \ v' = r_i(v) \\
\Rightarrow & \{ \text{ Inductive hypothesis ; disjoint clock sets ; rule } (S_1) \ \} \\
 & [u[i], v\lceil^i] \stackrel{e}{\Longrightarrow} [u[i]', r_i(v\lceil^i)] \ \wedge \ v' = r_i(v) \\
\Leftrightarrow & \{ \text{ Lemma 2 ; substitution } \ \} \\
 & [u[i], v\lceil^i] \stackrel{e}{\Longrightarrow} [u[i]', v'\lceil^i] \\
\Rightarrow & \{ \text{ Transition system construction } \ \} \\
 & [u[i]', v'\lceil^i] \in [\![A[i]]\!].1
\end{array}
$$

which is as required.

Case 2 [$[\,||^a u, v] \stackrel{t}{\Longrightarrow} [\,||^a u, v+t]$]

$$
\begin{array}{ll}
 & [\,||^a u, v] \stackrel{t}{\Longrightarrow} [\,||^a u, v+t] \\
\Leftrightarrow & \{ \text{ Rule } (S_2) \ \} \\
 & \forall ||^a u' \, (||^a u \xrightarrow{e,g,d,r} ||^a u' \implies \forall t' < t \,.\, \neg d(v+t')) \qquad - (\times)
\end{array}
$$

We seek to show that,

$$\forall u[i]' \, (u[i] \xrightarrow{e'_i, g'_i, d'_i, r'_i} u[i]' \implies \forall t' < t \,.\, \neg d'_i(v+t'))$$

Thus, we take $u[i]' \in A[i].1$ and assume,

$$u[i] \xrightarrow{e'_i, g'_i, d'_i, r'_i} u[i]'$$

Then we have two cases dependent upon the nature of e'_i.

Case 2.1 [$e'_i \in CA$]

$$
\begin{array}{ll}
 & e'_i \in CA \\
\Rightarrow & \{ \text{ Case assumption } \ \} \\
 & u[i] \xrightarrow{e'_i, g'_i, d'_i, r'_i} u[i]' \ \wedge \ e'_i \in CA \\
\Rightarrow & \{ \text{ Rule } \mathit{(RIA)} \text{ ; assumption } (\times) \ \} \\
 & ||^a u \xrightarrow{e'_i, g'_i, d'_i, r'_i} ||^a u' \ \wedge \ e'_i \in CA \ \wedge \ (\times) \\
\Rightarrow & \{ \text{ Instantiating universal in } (\times) \ \} \\
 & \forall t' < t \,.\, \neg d'_i(v+t')
\end{array}
$$

which is as required.

Case 2.2 [$e'_i \in HA$]

$$
\begin{array}{ll}
& e'_i \in HA \\
\Rightarrow & \{ \text{ Case assumption } \} \\
& u[i] \xrightarrow{e'_i, g'_i, d'_i, r'_i} u[i]' \ \wedge \ e'_i \in HA \\
\Rightarrow & \{ \text{ Rule } \mathit{(RHA)} \ (X \text{ as in } \mathit{(RHA)}) \ ; \text{ assumption } (\times) \ \} \\
& ||^a u \xrightarrow{e'_i, g'', d'', r_i} ||^a u' \ \wedge \ e'_i \in HA \ \wedge \ (\times) \ \wedge \ g'' = X \ \wedge \\
& d'' = d'_i \ \wedge \ \bigwedge \{ \neg q.3 \,|\, q \in \theta_{CA}(u[i]) \} \ \wedge \\
& \bigwedge \{ \neg(q.2 \ \wedge \ q'.2 \ \wedge \ (q.3 \ \vee \ q'.3)) \,|\, q \in \theta_{HA}(u[i]) \ \wedge \\
& q' \in \theta_{\{\overline{q.1}\}}(u[j]) \ \wedge \ j \neq i \} \\
\Rightarrow & \{ \text{ Instantiating universal in } (\times) \ ; \text{ def. of timing constraints ; logic } \} \\
& \forall t' < t \, \neg(d'_i(v+t') \ \wedge \ \bigwedge \{ \neg q.3(v+t') \,|\, q \in \theta_{CA}(u[i]) \} \ \wedge \\
& \bigwedge \{ \neg(q.2 \ \wedge \ q'.2 \ \wedge \ (q.3 \ \vee \ q'.3))(v+t') \,|\, q \in \theta_{HA}(u[i]) \ \wedge \\
& q' \in \theta_{\{\overline{q.1}\}}(u[j]) \ \wedge \ j \neq i \} \\
\Rightarrow & \{ \text{ De Morgan's } \} \\
& \forall t' < t \, (\neg d'_i(v+t') \ \vee \ \bigvee \{ q.3(v+t') \,|\, q \in \theta_{CA}(u[i]) \} \ \vee \\
& \bigvee \{ (q.2 \ \wedge \ q'.2 \ \wedge \ (q.3 \ \vee \ q'.3))(v+t') \,|\, q \in \theta_{HA}(u[i]) \ \wedge \\
& q' \in \theta_{\{\overline{q.1}\}}(u[j]) \ \wedge \ j \neq i \} \qquad -(*)
\end{array}
$$

Our strategy from here is to show that the second two disjuncts cannot hold for any $t' < t$.

Case 2.2.1 [$\exists t' < t \bigvee \{ q.3(v+t') \,|\, q \in \theta_{CA}(u[i]) \}$]

$$
\begin{array}{ll}
& \exists t' < t \bigvee \{ q.3(v+t') \,|\, q \in \theta_{CA}(u[i]) \} \\
\Rightarrow & \{ \text{ Definition of } \theta \ ; \text{ evaluating disjunct } \} \\
& \exists t' < t \, \exists u[i]'' \, (u[i] \xrightarrow{x, g''_i, d''_i, r''_i} u[i]'' \ \wedge \ d''_i(v+t')) \\
\Rightarrow & \{ \text{ Rule } (RIA) \ ; \text{ assumption } (\times) \ \} \\
& \exists t' < t \, \exists u[i]'' \, (||^a u \xrightarrow{x, g''_i, d''_i, r''_i} ||^a u[i''/i] \ \wedge \ d''_i(v+t')) \ \wedge \ (\times) \\
\Rightarrow & \{ \text{ Instantiating universal in } (\times) \ ; \text{ logic } \} \\
& \exists t' < t \, \exists u[i]'' \, (\forall t'' < t \, . \, \neg d''_i(v+t'') \ \wedge \ d''_i(v+t')) \\
\Rightarrow & \{ \text{ Reducing contradiction to false } \} \\
& \mathit{false}
\end{array}
$$

So, this subcase is contradictory and hence impossible.

Case 2.2.2
[$\exists t' < t \bigvee \{ (q.2 \ \wedge \ q'.2 \ \wedge \ (q.3 \ \vee \ q'.3))(v+t') \,|$
$q \in \theta_{HA}(u[i]) \ \wedge \ q' \in \theta_{\{\overline{q.1}\}}(u[j]) \ \wedge \ j \neq i \}$]

$$
\begin{array}{ll}
& \exists t' < t \bigvee \{ (q.2 \ \wedge \ q'.2 \ \wedge \ (q.3 \ \vee \ q'.3))(v+t') \,| \\
& q \in \theta_{HA}(u[i]) \ \wedge \ q' \in \theta_{\{\overline{q.1}\}}(u[j]) \ \wedge \ j \neq i \} \\
\Rightarrow & \{ \text{ Definition of } \theta \ ; \text{ evaluating disjuncts } \}
\end{array}
$$

$\exists t' < t\, \exists u[i]'', u[j]''\, (u[i] \xrightarrow{a, g_i'', d_i'', r_i''} u[i]'' \;\wedge\; u[j] \xrightarrow{\overline{a}, g_j'', d_j'', r_j''} u[j]'' \;\wedge$
$g_i''(v+t') \;\wedge\; g_j''(v+t') \;\wedge\; (d_i''(v+t') \;\vee\; d_j''(v+t')))$
$\Rightarrow$ { Rule (RCA) ; assumption $(\times)$ }
$\exists t' < t\, \exists ||^a u''\, (||^a u \xrightarrow{\uparrow a, g', d', r'} ||^a u'' \;\wedge\; g' = (g_i'' \;\wedge\; g_j'') \;\wedge$
$d' = (g' \;\wedge\; (d_i'' \;\vee\; d_j'')) \;\wedge\; g_i''(v+t') \;\wedge\; g_j''(v+t') \;\wedge$
$(d_i''(v+t') \;\vee\; d_j''(v+t'))) \;\wedge\; (\times)$
$\Rightarrow$ { Instantiating universal in $(\times)$; logic ; substitution }
$\exists t' < t\, (\forall t'' < t\,.\, \neg d'(v+t'') \;\wedge\; d' = (g_i'' \;\wedge\; g_j'' \;\wedge\; (d_i'' \;\vee\; d_j'')) \;\wedge$
$g_i''(v+t') \;\wedge\; g_j''(v+t') \;\wedge\; (d_i''(v+t') \;\vee\; d_j''(v+t')))$
$\Rightarrow$ { Substitution }
$\exists t' < t\, (\forall t'' < t\,.\, \neg d'(v+t'') \;\wedge\; d'(v+t'))$
$\Rightarrow$ { Reducing contradiction to false }
false

So, this subcase is also contradictory and hence impossible.

Thus, the last two disjuncts of (*) cannot hold for any $t' < t$ and hence we know that for all $t' < t$ the first disjunct must be true. Thus,

(*)
$\Rightarrow$ { Above cases 2.2.1 and 2.2.2 }
$\forall t' < t\, (\neg d_i'(v+t'))$

as required to complete case 2.2.

Now bringing cases 2.1 and 2.2 together, we have,

$$\forall u[i]'\, (u[i] \xrightarrow{e_i', g_i', d_i', r_i'} u[i]' \implies \forall t' < t\,.\, \neg d_i'(v+t'))$$

However, due to disjointness of clock sets we can deduce that,

$$\forall u[i]'\, (u[i] \xrightarrow{e_i', g_i', d_i', r_i'} u[i]' \implies \forall t' < t\,.\, \neg d_i'(v\lceil^i + t'))$$

and by our inductive hypothesis we know that,

$$[u[i], v\lceil^i] \in [\![\, A[i] \,]\!].1$$

from which (by S_2) it follows that,

$$[u[i], v\lceil^i] \xrightarrow{t} [u[i], v\lceil^i + t]$$

and thus,

$$[u[i], v\lceil^i + t] \in [\![\, A[i] \,]\!].1$$

which is as required to complete our proof of the inductive case.
○

Now we turn to the central result of this section. It states that $||^a$ preserves action lock freeness.

Theorem 1
$\exists i(1 \leq i \leq |A|) \,.\, A[i]$ *is action lock free* $\Longrightarrow ||^a A$ *is action lock free.*

Proof
We can express the desired property as follows,

$$\exists i \,.\, \forall [u[i], v\lceil^i] \in [\![A[i]]\!].1 \, \exists t \in \mathbb{R}^{+0} \, ([u[i], v\lceil^i + t] \in [\![A[i]]\!].1 \ \wedge \ [u[i], v\lceil^i + t] \overset{e}{\Longrightarrow})$$

implies

$$\forall [\, ||^a u, v] \in [\![\, ||^a A]\!].1 \, \exists t \in \mathbb{R}^{+0} \, ([\, ||^a u, v + t] \in [\![\, ||^a A]\!].1 \ \wedge \ [\, ||^a u, v + t] \overset{e}{\Longrightarrow})$$

Thus, we assume,

$$\exists i \,.\, \forall [u[i], v\lceil^i] \in [\![A[i]]\!].1 \, \exists t \in \mathbb{R}^{+0} \, ([u[i], v\lceil^i + t] \in [\![A[i]]\!].1 \ \wedge \ [u[i], v\lceil^i + t] \overset{e}{\Longrightarrow})$$

and then we take, $[\, ||^a u, v] \in [\![\, ||^a A]\!].1$, However,

$$
\begin{array}{ll}
 & [\, ||^a u, v] \in [\![\, ||^a A]\!].1 \\
\Rightarrow & \{ \ \text{Lemma 5} \ \} \\
 & [u[i], v\lceil^i] \in [\![A[i]]\!].1
\end{array}
$$

Now we will show that $\neg AL([u[i], v\lceil^i])$ implies that $\neg AL([\, ||^a u, v])$. We consider two cases ($t = 0$ and $t > 0$) depending upon whether $[u[i], v\lceil^i]$ can immediately perform an action or only after passing time.

Case 1 [$t = 0$]

$$
\begin{array}{ll}
 & t = 0 \\
\Leftrightarrow & \{ \ \text{Case assumption} \ \} \\
 & [u[i], v\lceil^i] \overset{e_i}{\Longrightarrow} \\
\Leftrightarrow & \{ \ \text{Rule } (S_1) \ ; \ g_i \text{ only uses clocks in } A[i].4 \ \} \\
 & u[i] \xrightarrow{e_i, g_i, d_i, r_i} \ \wedge \ g_i(v) \qquad - (+)
\end{array}
$$

which yields subcases dependent upon the nature of e_i.

Case 1.1 [$e_i \in CA$]

$$
\begin{array}{ll}
 & e_i \in CA \\
\Rightarrow & \{ \ \text{Rule } \mathit{(RIA)} \ ; \ \text{case assumption } (+) \ \} \\
 & ||^a u \xrightarrow{e_i, g_i, d_i, r_i} ||^a u' \ \wedge \ g_i(v) \ \wedge \ e_i \in CA \\
\Leftrightarrow & \{ \ \text{Rule } (S_1) \ \} \\
 & [\, ||^a u, v] \overset{e_i}{\Longrightarrow}
\end{array}
$$

which is as required.

Case 1.2 [$e_i \in HA$]

Case 1.2.1 [$\exists u[i]' \, (u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \ \wedge \ x \in CA \ \wedge \ \exists t \,.\, g_x((v + t)\lceil^i))$]

Case 1.2.1.1 [$[\,||^a u, v] \overset{t}{\not\Longrightarrow}$]

$$[\,||^a u, v] \overset{t}{\not\Longrightarrow}$$

$\Rightarrow$ { Proposition 6 }

$$\exists e\,([\,||^a u, v] \overset{e}{\Longrightarrow} \;\vee\; \exists t' \leq t\,([\,||^a u, v] \overset{t'}{\Longrightarrow} [\,||^a u, v+t'] \;\wedge\; [\,||^a u, v+t'] \overset{e}{\Longrightarrow}))$$

which is as required.

Case 1.2.1.2 [$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t]$]

$$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t]$$

$\Rightarrow$ { Rule *(RIA)* ; assumptions ; disjoint clock sets }

$$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t] \;\wedge\; ||^a u \xrightarrow{x, g_x, d_x, r_x} \;\wedge\; g_x(v+t)$$

$\Rightarrow$ { Rule (S_1) }

$$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t] \;\wedge\; [\,||^a u, v+t] \overset{x}{\Longrightarrow}$$

which is as required.

Case 1.2.2 [$\neg\exists u[i]'\,(u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \;\wedge\; x \in CA \;\wedge\; \exists t\,.\, g_x((v+t)\lceil^i))$]

Case 1.2.2.1 [$\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \wedge u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \wedge$ $\exists t\,.\,(g_y \;\wedge\; g_z)(v+t))$]

Case 1.2.2.1.1 [$[\,||^a u, v] \overset{t}{\not\Longrightarrow}$]
Similar to case 1.2.1.1.

Case 1.2.2.1.2 [$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t]$]

$$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t]$$

$\Rightarrow$ { Rule *(RCA)* ; assumptions ; disjointness of clock sets }

$$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t] \;\wedge\; ||^a u \xrightarrow{\downarrow a, g', d', r'} \;\wedge\; g' = (g_y \;\wedge\; g_z) \;\wedge$$
$$d' = (g' \;\wedge\; (d_y \;\vee\; d_z)) \;\wedge\; (g_y \;\wedge\; g_z)(v+t)$$

$\Rightarrow$ { Rule (S_1) }

$$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t] \;\wedge\; [\,||^a u, v+t] \overset{\downarrow a}{\Longrightarrow}$$

which is as required.

Case 1.2.2.2 [$\neg(\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \wedge u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \wedge$ $\exists t\,.\,(g_y \;\wedge\; g_z)(v+t)))$]

$$\neg(\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \;\wedge\; u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \;\wedge$$
$$\exists t\,.\,(g_y \;\wedge\; g_z)(v+t)))$$

$\Rightarrow$ { Accumulating assumptions ; disjointness of clock sets }

$$\neg(\exists u[i]'\,(u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \;\wedge\; \exists t\,.\, g_x(v+t))) \;\wedge$$
$$\neg(\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \;\wedge\; u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \;\wedge$$

$\exists t \,.\, (g_y \;\wedge\; g_z)(v+t)))$

$\Rightarrow \quad \{ \;$ Definition of temporal operators $\; \}$

$\neg(\exists u[i]' \, (u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \;\wedge\; \Diamond g_x(v))) \;\wedge$

$\neg(\exists u[i]', u[k], u[k]'(k \neq i) \, (u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \;\wedge\; u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \;\wedge$

$\Diamond(g_y \;\wedge\; g_z)(v)))$

$\Leftrightarrow \quad \{ \;$ Logic $\; \}$

$\forall u[i]' \, (u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \implies \Box\neg g_x(v)) \;\wedge$

$\forall u[i]', u[k], u[k]'(k \neq i) \, ((u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \wedge u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]') \implies$

$\Box\neg(g_y \;\wedge\; g_z)(v))$

$\Rightarrow \quad \{ \;$ Definition of $\wedge$ and θ $\; \}$

$\bigwedge\{ \, \Box\neg q.2(v) \mid q \in \theta_{CA}(u[i]) \, \} \;\wedge$

$\bigwedge\{ \, \Box\neg(q.2 \wedge q'.2)(v) \mid q \in \theta_{HA}(u[i]) \wedge q' \in \theta_{\{\overline{q.1}\}}(u[k]) \wedge k \neq i \, \} \qquad - (\#)$

However, in addition, we are in case 1 (with assumption (+)) and 1.2 which gives us,

$u[i] \xrightarrow{e_i, g_i, d_i, r_i} u[i]' \;\wedge\; g_i(v) \;\wedge\; e_i \in HA$

$\Rightarrow \quad \{ \;$ Rule *(RHA)* $\; \}$

$||^a u \xrightarrow{e_i, g, d, r_i} ||^a u' \;\wedge\; g = ((g_i \;\wedge\; \bigwedge\{ \, \Box\neg q.2 \mid q \in \theta_{CA}(u[i]) \, \} \;\wedge$

$\bigwedge\{ \, \Box\neg(q.2 \;\wedge\; q'.2) \mid q \in \theta_{HA}(u[i]) \;\wedge\; q' \in \theta_{\{\overline{q.1}\}}(u[j]) \;\wedge\; j \neq i \, \}) \vee d_i) \;\wedge$

$g_i(v) \;\wedge\; e_i \in HA$

$\Rightarrow \quad \{ \;$ Assumption $(\#)$; logic $\; \}$

$||^a u \xrightarrow{e_i, g, d, r_i} ||^a u' \;\wedge\; g(v)$

$\Rightarrow \quad \{ \;$ Rule (S_1) $\; \}$

$[\, ||^a u, v] \overset{e_i}{\Longrightarrow}$

which is as required and completes case 1.

Case 2 $[\; t > 0 \;]$

$t > 0$

$\Leftrightarrow \quad \{ \;$ Case assumption $\; \}$

$\exists t \, ([u[i], v\lceil^i] \overset{t}{\Longrightarrow} [u[i], (v\lceil^i) + t] \;\wedge\; [u[i], (v\lceil^i) + t] \overset{e_i}{\Longrightarrow})$

$\Rightarrow \quad \{ \;$ Lemma 3 $\; \}$

$\exists t \, ([u[i], v\lceil^i] \overset{t}{\Longrightarrow} [u[i], (v+t)\lceil^i] \;\wedge\; [u[i], (v+t)\lceil^i] \overset{e_i}{\Longrightarrow})$

$\Leftrightarrow \quad \{ \;$ Rule (S_1) ; g_i only uses clocks in $A[i].4$ $\; \}$

$u[i] \xrightarrow{e_i, g_i, d_i, r_i} \;\wedge\; g_i(v+t) \qquad - (++)$

Case 2.1 $[\; [\, ||^a u, v] \overset{t}{\not\Longrightarrow} \;]$

Similar to case 1.2.1.1.

Case 2.2 [$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t]$]
Now we have subcases dependent upon the nature of e_i.

Case 2.2.1 [$e_i \in CA$]

$$e_i \in CA$$
$$\Rightarrow \quad \{ \text{ Rule } \textit{(RIA)} \text{ ; case assumption } (++) \; \}$$
$$||^a u \xrightarrow{e_i, g_i, d_i, r_i} ||^a u' \;\wedge\; g_i(v+t) \;\wedge\; e_i \in CA$$
$$\Leftrightarrow \quad \{ \text{ Case 2.2 assumption ; rule } (S_1) \; \}$$
$$[\,||^a u, v] \overset{t}{\Longrightarrow} [\,||^a u, v+t] \;\wedge\; [\,||^a u, v+t] \overset{e_i}{\Longrightarrow}$$

which is as required.

Case 2.2.2 [$e_i \in HA$]

Case 2.2.2.1 [$\exists u[i]' \,(u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \;\wedge\; x \in CA \;\wedge\; \exists t' \,.\, g_x((v+t+t')\lceil^i))$]

Case 2.2.2.1.1 [$[\,||^a u, v+t] \overset{t'}{\not\Longrightarrow}$]
Similar to case 1.2.1.1.

Case 2.2.2.1.2 [$[\,||^a u, v+t] \overset{t'}{\Longrightarrow} [\,||^a u, v+t+t']$]

$$[\,||^a u, v+t] \overset{t'}{\Longrightarrow} [\,||^a u, v+t+t']$$
$$\Rightarrow \quad \{ \text{ Time cont. ; rule } \textit{(RIA)} \text{ ; case 2.2.2.1 assumption ; disjoint clocks } \}$$
$$[\,||^a u, v] \overset{t+t'}{\Longrightarrow} [\,||^a u, v+t+t'] \;\wedge\; ||^a u \xrightarrow{x, g_x, d_x, r_x} \;\wedge\; g_x(v+t+t')$$
$$\Rightarrow \quad \{ \text{ Rule } (S_1) \; \}$$
$$[\,||^a u, v] \overset{t+t'}{\Longrightarrow} [\,||^a u, v+t+t'] \;\wedge\; [\,||^a u, v+t+t'] \overset{x}{\Longrightarrow}$$

which is as required.

Case 2.2.2.2 [$\neg\exists u[i]' \,(u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \wedge x \in CA \wedge \exists t' \,.\, g_x((v+t+t')\lceil^i))$]

Case 2.2.2.2.1 [$\exists u[i]', u[k], u[k]' (k \neq i)\, (u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \wedge u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \wedge$
$\exists t' \,.\, (g_y \;\wedge\; g_z)(v+t+t'))$]

Case 2.2.2.2.1.1 [$[\,||^a u, v+t] \overset{t'}{\not\Longrightarrow}$]
Similar to case 1.2.1.1.

Case 2.2.2.2.1.2 [$[\,||^a u, v+t] \overset{t'}{\Longrightarrow} [\,||^a u, v+t+t']$]

$$[\,||^a u, v+t] \overset{t'}{\Longrightarrow} [\,||^a u, v+t+t']$$
$$\Rightarrow \quad \{ \text{ Time continuity; rule } \textit{(RCA)} \text{ ; assumptions } \}$$
$$[\,||^a u, v] \overset{t+t'}{\Longrightarrow} [\,||^a u, v+t+t'] \;\wedge\; ||^a u \xrightarrow{\downarrow a, g', d', r'} \;\wedge\; g' = (g_y \;\wedge\; g_z) \;\wedge$$
$$d' = (g' \;\wedge\; (d_y \;\vee\; d_z)) \;\wedge\; (g_y \;\wedge\; g_z)(v+t+t')$$
$$\Rightarrow \quad \{ \text{ Rule } (S_2) \; \}$$

$$[\,||^a u, v] \xRightarrow{t+t'} [\,||^a u, v+t+t'] \ \wedge\ [\,||^a u, v+t+t'] \xRightarrow{\downarrow a}$$

which is as required.

Case 2.2.2.2.2 $[\,\neg(\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \wedge u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \wedge \exists t' \,.\, (g_y \ \wedge\ g_z)(v+t+t')))\,]$

$$\begin{aligned}
& \neg(\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \ \wedge\ u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \ \wedge \\
& \exists t' \,.\, (g_y \ \wedge\ g_z)(v+t+t'))) \\
\Rightarrow\ & \{\ \text{Accumulating assumptions ; disjointness of clock sets}\ \} \\
& \neg(\exists u[i]'\,(u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \ \wedge\ \exists t' \,.\, g_x(v+t+t'))) \ \wedge \\
& \neg(\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \ \wedge\ u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \ \wedge \\
& \exists t' \,.\, (g_y \ \wedge\ g_z)(v+t+t'))) \\
\Rightarrow\ & \{\ \text{Definition of temporal operators}\ \} \\
& \neg(\exists u[i]'\,(u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \ \wedge\ \Diamond g_x(v+t))) \ \wedge \\
& \neg(\exists u[i]', u[k], u[k]'(k \neq i)\,(u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \ \wedge\ u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k]' \ \wedge \\
& \Diamond(g_y \ \wedge\ g_z)(v+t))) \\
\Leftrightarrow\ & \{\ \text{Logic}\ \} \\
& \forall u[i]'\,(u[i] \xrightarrow{x, g_x, d_x, r_x} u[i]' \ \Longrightarrow\ \Box\neg g_x(v+t)) \ \wedge \\
& \forall u[i]', u[k], u[k]'(k \neq i)\,((u[i] \xrightarrow{a, g_y, d_y, r_y} u[i]' \wedge u[k] \xrightarrow{\overline{a}, g_z, d_z, r_z} u[k])' \ \Longrightarrow \\
& \Box\neg(g_y \ \wedge\ g_z)(v+t)) \\
\Rightarrow\ & \{\ \text{Definition of } \wedge \text{ and } \theta\ \} \\
& \bigwedge\{\, \Box\neg q.2(v+t) \,|\, q \in \theta_{CA}(u[i]) \,\} \ \wedge \\
& \bigwedge\{\, \Box\neg(q.2 \ \wedge\ q'.2)(v+t) \,|\, q \in \theta_{HA}(u[i]) \ \wedge\ q' \in \theta_{\{\overline{q.1}\}}(u[k]) \ \wedge \\
& k \neq i \,\} \qquad - \ (\#\#)
\end{aligned}$$

However, in addition, we are in case 2 and 2.2.2 which gives us,

$$\begin{aligned}
& u[i] \xrightarrow{e_i, g_i, d_i, r_i} u[i]' \ \wedge\ g_i(v+t) \ \wedge\ e_i \in HA \\
\Rightarrow\ & \{\ \text{Rule } \mathit{(RHA)}\ \} \\
& ||^a u \xrightarrow{e_i, g, d, r_i} ||^a u' \ \wedge\ g = ((g_i \ \wedge\ \bigwedge\{\, \Box\neg q.2 \,|\, q \in \theta_{CA}(u[i]) \,\} \ \wedge \\
& \bigwedge\{\, \Box\neg(q.2 \ \wedge\ q'.2) \,|\, q \in \theta_{HA}(u[i]) \ \wedge\ q' \in \theta_{\{\overline{q.1}\}}(u[j]) \ \wedge\ j \neq i \,\}) \ \vee\ d_i) \ \wedge \\
& g_i(v+t) \ \wedge\ e_i \in HA \\
\Rightarrow\ & \{\ \text{Assumption } (\#\#) \text{ ; logic}\ \} \\
& ||^a u \xrightarrow{e_i, g, d, r_i} ||^a u' \ \wedge\ g(v+t) \\
\Rightarrow\ & \{\ \text{Case 2.2 assumption ; Rule } (S_1)\ \} \\
& [\,||^a u, v] \xRightarrow{t} [\,||^a u, v+t] \ \wedge\ [\,||^a u, v+t] \xRightarrow{e_i}
\end{aligned}$$

which is as required and completes case 2 and thus, the whole proof.
○

Acknowledgements

The author has benefited greatly from discussions with Sebastian Bornot, Joseph Sifakis and Stavros Tripakis and would also like to recognise the contribution of Giorgio Faconti, Joost-Pieter Katoen, Diego Latella and Meike Massink who were involved in preliminary discussions from which this paper has grown.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, pages 183–235, 1994.

[2] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, and Paul Pettersson amd Wang Yi. Uppaal - a tool suite for automatic verification of real-time system. In *Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 1995.

[3] S. Bornot and J. Sifakis. On the composition of hybrid systems. In *Hybrid Systems: Computation and Control*, LNCS 1386, pages 49–63, 1998.

[4] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. In *Compositionality, COMPOS'97*, LNCS 1536, 1997.

[5] H. Bowman. Discussion document - modelling timeout behaviour in timed automata. Technical report, Available from author, 1998.

[6] H. Bowman. Modelling timeouts without timelocks. In *ARTS'99, Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop*, LNCS 1601, pages 335–353. Springer-Verlag, 1999.

[7] H. Bowman, G. Faconti, J-P. Katoen, D. Latella, and M. Massink. Automatic verification of a lip synchronisation algorithm using UPPAAL. In *Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems*, 1998. To Appear in Special Issue of Formal Aspects of Computing.

[8] C.Daws, A.Olivero, S.Tripakis, and S.Yovine. The tool KRONOS. In *Hybrid Systems III, Verification and Control*, LNCS 1066. Springer-Verlag, 1996.

[9] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[10] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebra. In *Real-time Theory in Practice*, LNCS 600, pages 549–572. Springer-Verlag, June 1991.

[11] T. Regan. Multimedia in temporal LOTOS: A lip synchronisation algorithm. In *PSTV XIII, 13th Protocol Specification, Testing and Verification.* North-Holland, 1993.

[12] S. Tripakis. Verifying progress in timed systems. In *ARTS'99, Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop*, LNCS 1601. Springer-Verlag, 1999.