URL: http://www.elsevier.nl/locate/entcs/volume52.html 19 pages

Extending Timed Automata for Compositional Modeling Healthy Timed Systems

Víctor Braberman [1,2]

Computer Science Department, FCEyN,
Universidad de Buenos Aires,
Buenos Aires, Argentina

Alfredo Olivero [3,4]

Department of Information Technology, FIyCE,
Universidad Argentina de la Empresa,
Buenos Aires, Argentina

Abstract

We introduce the notion of Timed I/O Components as Timed Automata "à la" Alur & Dill where an "admissible" I/O interface is declared. That notion has, what we consider, a key modeling property: non-zeno preservation under syntactically-checkable "I/O compatibility" among interacting components. Also a reduced parallel composition is posssible based on the ability of statically detect influence of behavior between components [8,10,11]. On the other hand, with some simple extra conditions, modular assume-guarantee style of reasoning like [15,19] is valid in our model.

1 Introduction: on Non-Zeno and Non-Blocking Models

Well-defined models of timed systems usually are required to be "non-zeno". Roughly speaking, non-zenoness means that any finite run can be extended to a time-divergent infinite run (i.e., no "black-alleys", time can always progress). On the one hand, zenoness is usually a symptom of ill-modeling, on the other hand non-zenoness is required to perform some verification procedures when semantics is restricted to divergent runs.

[1] Research supported by UBACyT grant X156 and TW72
[2] Email: vbraber@dc.uba.ar
[3] Research supported by UADE grant ING6-01
[4] Email: aolivero@uade.edu.ar

Unfortunately, non-zenoness is not preserved by parallel composition. Non-Zeno systems may produce time-locks when connected.

I/O Timed Components (I/O TCs) are compositional models developed for expressing non-zeno timed behavior built on top of Timed Automata (TAs) [8]. They impose a modeling discipline for guaranteeing that parallel composition among "compatible" I/O TCs is a natural way to constrain the behavior of individual components without introducing zeno behavior. Let us pinpoint some interesting aspects of I/O TCs:

- I/O interfaces allow simple syntactic checks that ensure non-zeno preservation under parallel composition.
- By using I/O interfaces it is possible to calculate, in a quite precise way, the influence of a component on the behavior of another component (see [8,10,11]). As far as we know, this is a completely new goal for I/O interfaces.
- Since I/O TCs are built on top of a simple notion of TAs "à la" Alur-Dill -with a communication based on label sharing- they are immediately supported by several checking tools like Kronos [13], Uppaal [6],etc.
- I/O TCs are defined without resorting to "receptiveness games" like in live I/O Timed Automata [15], Reactive Modules [5], etc. Conditions for checking good I/O label division are easy to automate.
- We believe that they are suitable to compactly model high-level non-blocking abstractions (see discussion in [8] and Sect. 3).
- With some extra constraints on I/O TCs, it is possible to apply "assume-guarantee"-style rules (e.g.,[19]) for refinement checking. Like in [19] those constraints are not based on more general but complicated receptiveness games [15,5]). Those constraints are not basic properties for I/O TCs (which are mainly inspired in non-zeno preservation). We think that this separation has theoretical and practical interest.
- I/O notions are rather independent of the underlying timed (or untimed[5]) formalism used to describe the dynamics.

It is worth mentioning related work on preserving "reactivity and activity" of components. In [7], an algebraic framework based on the temporal properties of synchronization operation is presented (they aim at getting high level synchronization facilities). Our point of view is a functional classification of transitions. In that line of research, authors of [19] present non-blocking Timed Processes to get a family of automata where they can apply an assume/guarantee style of reasoning. Communication between components is based on signal change instead of label sharing and it is suited to circuit mod-

[5] We believe that this I/O model can be adapted to the untimed framework by changing timed divergence conditions with fairness constraints [12,15], an usual way to specify progress in the untimed framework.

eling. Differently from our approach, output changes are constrained to be non-transient and the update of inputs is independent from the update of outputs. Since that model is focused on breaking circularity of assume-guarantee rules, the underlying notion of non-zenoness does not need to rule out black-alleys; instead definitions rule out forcing infinitely many transitions within a finite interval.

Liveness and I/O interfaces have been considered in a general setting for simulation proof methods "à la" Lynch-Vaandrager [15] geared towards theorem provers. In that work, Live Timed I/O automata using a notion of "responsiveness" is defined based on games which embeds several proposals for fair I/O timed systems [20,24], etc. A closer model are the Reactive Modules of [5]. Unlike our notions, it is based on receptiveness games to define non-blocking and I/O variables to communicate modules.

In next section we recall Timed Automata. In Sect. 3, we formally present I/O Timed Components. Some applications are mentioned in Sect. 4. Conditions to get assume-guarantee rule are discussed in Sect. 5. Finally, we summarize the results and mention some future work.

2 Timed Automata

Timed Automata (TA) is one of the most widely used formalism to model and analyze timed systems and is supported by several tools (e.g., [13,6,18], etc.). This presentation partially follows [26]. Given a finite set of clocks (non-negative real variables) $X = \{x_1, x_2, \ldots, x_n\}$, a *valuation* is a total function $v : X \overset{tot}{\rightarrow} \mathbb{R}_{\geq 0}$ where $v(x_i)$ is the value associated with clock x_i. We define V_X as the set $[X \overset{tot}{\rightarrow} \mathbb{R}_{\geq 0}]$ of total functions mapping X to $\mathbb{R}_{\geq 0}$. $\mathbf{0} \in V_X$ denotes the function that evaluates to 0 all clocks. Given $v \in V_X$ and $t \in \mathbb{R}_{\geq 0}$, $v+t$ denotes the valuation that assigns to each clock $x \in X$ the value $v(x)+t$. Given a set of clocks X, a subset $\alpha \subseteq X$ and a valuation v we define $Reset_\alpha(v)$ as a valuation that assigns zero to clocks in α and keeps the same value than v for the remaining clocks. Given a set of clocks X we define the sets of clock constraints Ψ_X according to the grammar: $\Psi_X \ni \psi ::= x \sim c | x - x' \sim c | \psi \wedge \psi | \psi \vee \psi$, where $x, x' \in X, \sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

A valuation $v \in V_X$ satisfies $\psi \in \Psi_X$ ($v \models \psi$) iff the expression evaluates **true** when each clock is replaced with its current value specified in v.

Definition 2.1 [Timed Automata] A timed automaton (TA) is a tuple $A = \langle S, X, \Sigma, E, I, s_0 \rangle$ where S is a finite set of locations, X is a finite set of clocks, Σ is a set of labels, E is a finite set of edges, (each edge $e \in E$ is a tuple $\langle s, a, \psi, \alpha, s' \rangle$ where: $s \in S$ is the source location, $s' \in S$ is the target location, $a \in \Sigma$ is the label, $\psi \in \Psi_X$ is the guard, $\alpha \subseteq X$ is the subset of clocks reset at the edge), $I : S \overset{tot}{\rightarrow} \Psi_X$ is a total function associating with each location a clock constraint called location's Invariant, and $s_0 \in S$ is the initial location.

Given a TA $A = \langle S, X, \Sigma, E, I, s_0 \rangle$ we define $Locs(A) = S$, $Clocks(A) = X$, $Labels(A) = \Sigma$, $Edges(A) = E$, $Inv(A) = I$, $Init(A) = s_0$, and given an edge $e = \langle s, a, \psi, \alpha, s' \rangle \in E$ we define $src(e) = s$, $Label(e) = a$, $Guard(e) = \psi$, $Rst(e) = \alpha$, $tgt(e) = s'$. The *State Space* Q_A of a TA A is the set of states $(s, v) \in S \times V_X$ for which $v \models I(s)$ and $q_0 = (Init(A), \mathbf{0})$ is its *initial state.* Given a state $q = (s, v)$ we denote: $q+t = (s, v+t)$, $q^{@} = s$, and $q(x_i) = v(x_i)$. The semantics of a TA A can be given in terms of the *Labeled Transition System* (LTS) of A, denoted $G_A = \langle Q_A, q_0, \mapsto, \Sigma \rangle$. The relation $\mapsto$ is the set of (time or discrete) transitions between states. Let $t \in \mathbb{R}_{\geq 0}$; the state (s, v) has a *time transition* to $(s, v + t)$ denoted $(s, v) \mapsto_t^{\lambda} (s, v + t)$ if for all $t' \leq t$, $v + t' \models I(s)$, where λ is a fictitius label. Let Σ^* denote $\Sigma \cup \{\lambda\}$. Let $e \in E$ be an edge; the state $(src(e), v)$ has a *discrete transition* to the state $(tgt(e), v')$ denoted $(src(e), v) \mapsto_0^{Label(e)} (tgt(e), v')$ if $v \models Guard(e)$ and $v' = Reset_{Rst(e)}(v)$.

We write $q \mapsto_0^l$ (the label $l \in \Sigma$ is enable at the state $q \in Q_A$) if $q \mapsto_0^l q'$ for some $q' \in Q_A$. Given a subset $\Sigma' \subseteq \Sigma$, we write $q \mapsto_0^{\Sigma'}$ (all labels $l \in \Sigma'$ are enable at the state $q \in Q_A$) if $q \mapsto_0^l$ for all $l \in \Sigma'$.

A *finite run* r of A starting at q is a finite sequence $q \mapsto_{t_0}^{a_0} q_1 \mapsto_{t_1}^{a_1} ... \mapsto_{t_{n-1}}^{a_{n-1}} q_n$ of states and transitions in G_A. The *time of occurrence* of the k^{th} $(k \leq n-1)$ transition is equal to $\sum_{i=o}^{k-1} t_i$ and is denoted as $\tau_r(k)$. The *time length* of the run (denoted as τ_r) is equal to $\tau_r(n)$. An infinite run is just an infinit sequence of states and transitions in G_A. The set of finite and infinite runs starting at q is denoted as $R_A(q)$. We call $Lab(r)$ the set of all labels in the run r.

A *divergent run* is an infinite run such that $\sum_{i=o}^{\infty} t_i = \infty$. The set of divergent runs of a TA A starting at state q is denoted $R_A^{\infty}(q)$. A TA is *non-zeno* when any finite run starting at the initial state can be extended to a divergent run, that is, the set of finite runs is equal to the set of finite prefixes of divergent runs. We say that the state q is *reachable* if there is a finite run starting at the initial state which ends at q; we denote the set of all reachable states in a TA A as $Reach(A)$.

Given a run $r = q \mapsto_{t_0}^{a_0} q_1 \mapsto_{t_1}^{a_1} ... \mapsto_{t_{n-1}}^{a_{n-1}} q_n ... \in R_A(q)$, the exhibited *timed-event sequence* of r, is a sequence $r_{\Sigma^*} = (a_0, \tau_r(0)), (a_1, \tau_r(1)), (a_2, \tau_r(2)), ...(a_{n-1}, \tau_r(n-1)), ...$ of pairs $(l, t) \in (\Sigma^*) \times \mathbb{R}_{\geq 0}$.

Given a run $r \in R_A(q)$ and a set of labels $L \subseteq \Sigma$, the exhibited *timed-event sequence over* L, denoted as r_L, is the maximum subsequence of r_{Σ^*} containing pairs (l, t) such that $l \in L$ (the sequence r_L shows the L-labeled transitions and their time stamps). Given a timed-event sequence over L named r_L, its length is denoted as $\#r_L$, its k-th pair (with $k < \#r_L$) is denoted as $r_L[k]$ and its prefix up to the k-th pair (with $k < \#r_L$) is denoted as $r_L[0...k]$. Given a pair $p = (l, t)$ in r_L, we define $lab(p) = l$ and $time(p) = t$.

Given two TAs A and A' and a set of labels $L \subseteq \Sigma \cap \Sigma'$, we say that $A \leq_L A'$ (A is a *refinement* of A' w.r.t L) iff for all finite run $r \in R_A(q_0)$ there exists a run $r' \in R_{A'}(q'_0)$ such that $\tau_r = \tau_{r'}$ and $r_L = r'_L$.

The parallel composition of TAs is defined over classical synchronous product of automata.

Definition 2.2 [Parallel composition] Given two TA $A_1 = \langle S_1, X_1, \Sigma_1, E_1, I_1, s_{0_1}\rangle$, and $A_2 = \langle S_2, X_2, \Sigma_2, E_2, I_2, s_{0_2}\rangle$ where $X_1 \cap X_2 = \emptyset$. Let E' be the set of edges defined over the $S_1 \times S_2$ as follows:

$\langle (s_1, s_2), a, \psi, \alpha, (s'_1, s'_2)\rangle \in E' \iff$

$\langle s_1, a, \psi, \alpha, s'_1\rangle \in E_1 \wedge a \notin \Sigma_{||} \wedge s_2 = s'_2$, or

$\langle s_2, a, \psi, \alpha, s'_2\rangle \in E_2 \wedge a \notin \Sigma_{||} \wedge s_1 = s'_1$, or

$\langle s_i, a, \psi_i, \alpha_i, s'_i\rangle \in E_i \wedge a \in \Sigma_{||} \wedge \psi = (\psi_1 \wedge \psi_2) \wedge \alpha = \alpha_1 \cup \alpha_2$

where $\Sigma_{||} = \Sigma_1 \cap \Sigma_2$.

The parallel composition $A_1||A_2$ is defined as: $A = \langle S, X_1 \cup X_2, \Sigma_1 \cup \Sigma_2, E, I, (s_{0_1}, s_{0_2})\rangle$ where $S \subseteq S_1 \times S_2$ is the set of locations reachable traversing the edges of E' from the initial location (s_{0_1}, s_{0_2}), $E \subseteq E'$ is the subset of edges with source and target in S, and for all $(s_1, s_2) \in E$, $I((s_1, s_2)) = I(s_1) \wedge I(s_2)$.

The $||$ operator is commutative and associative. We will denote $||_{i\in I}\, A_i$ the parallel composition of an indexed set of TA. If q is a state of that parallel composition $\Pi_i(q)$ will denote the local state of TA A_i (location and local-clocks values).

3 I/O Timed Components

In this section we define I/O concepts formally.

Given a TA A, we will divide $Labels(A)$ (its set of labels) into three sets: In_A (input-labels), Out_A (output-labels) and ϵ_A (internal-labels), such that $\{In_A, Out_A, \epsilon_A\} \in Part(Labels(A))$, where $Part(S)$ is the set of all partitions of the set S. We define the set Exp_A of *exported* labels (or interface labels) of A as $Exp_A = In_A \cup Out_A$.

A set of *input selections* of A is a set $I^A = \{I^A_1, I^A_2, \ldots, I^A_k\} \in Part(In_A)$, a set of *output selections* of A is a set $O^A = \{O^A_1, O^A_2, \ldots, O^A_h\} \in Part(Out_A)$. Note that $I^A \cup O^A \cup \{\epsilon_A\} \in Part(Labels(A))$.

Let us define what is a correct I/O (uncontrollable/controllable) interface labels for a TA.

Definition 3.1 [Admissible Input/Output interface for a TA] Given a non-zeno TA A, and the sets I^A, O^A of input and output selections of A, the pair (I^A, O^A) is an admissible input/output interface for A iff the following conditions hold:

For any state $q \in Reach(A)$

(i) for any input selection $I^A_n \in I^A$ there exists a label $i \in I^A_n$ such that $q \mapsto^i_0$. That is, given any input selection $I^A_n \in I^A$, the TA can always synchronize using some of the labels of I_n^A (there is always at least one alternative of every input selection enabled at each state).

(ii) there exists a run $r \in R_A^\infty(q)$ such that $Lab(r) \cap In_A = \emptyset$. Input is not mandatory and thus non-zenoness must be guaranteed without them [6].

(iii) for any output selection $O_m^A \in O^A$, if there exists a label $o \in O_m^A$ such that $q \mapsto_0^o$ then $q \mapsto_0^{O_m^A}$. All labels of an output selection are simultaneously enabled or disabled.

(iv) for any run $r \in R_A(q)$, if a label $o \in Out_A$ appears an infinite number of times in r, then necessarily $r \in R_A^\infty(q)$ (non-transientness of outputs [7]).

In the Appendix A.1 we show how to check I/O admissibility.

Definition 3.2 [I/O TCs] An I/O Timed Component (or I/O TC) is a tuple $(A, (I^A, O^A))$ where A is a non-zeno TA and (I^A, O^A) is an admissible I/O interface for A.

An output selection of size greater than one models alternative behaviors of the component according to the state of the component exporting those labels as input selection (similar to an external non-deterministic choice in Process Algebra-like notations, see example 3.4).

Given an I/O TC $C = (A, (I^A, O^A))$, C may also denote the underlying TA A when it can be deduced from the context. Thus, operations performed on I/O TCs should be understood as operations on its underlying TAs.

Definition 3.3 [Compatible Components] Given two I/O TCs $C_1 = (A_1, (I^{A_1}, O^{A_1}))$ and $C_2 = (A_2, (I^{A_2}, O^{A_2}))$, they are compatible components if and only if:

(i) $Labels(A_1) \cap Labels(A_2) \subseteq Exp_{A_1} \cap Exp_{A_2}$ (i.e., all common labels are exported by both A_1 and A_2),

(ii) for all $I_n^{A_1} \in I^{A_1}$ and $I_m^{A_2} \in I^{A_2}$ if $\#I_n^{A_1} > 1$ and $\#I_m^{A_2} > 1$ then $I_n^{A_1} \cap I_m^{A_2} = \emptyset$ (intersection of input selections of size greater than one must be empty).

(iii) $Out_{A_1} \cap Out_{A_2} = \emptyset$ (the components don't share output labels).

(iv) for all $I \in I^{A_1} \cup I^{A_2}$ and $O \in O^{A_1} \cup O^{A_2}$ then either $I \cap O = \emptyset$ or $I \subseteq O$ (output selection covers all input alternatives).

We refer to a set of pair-wise compatible components as a *compatible set of components*. I/O compatibility means that underlying TAs can not block each other and moreover, we will show that the composition of compatible components is itself a component and therefore a non-zeno automata.

[6] Note that this property is stronger than non-zenoness since it also requires time divergence avoiding input-labeled transitions. It is similar to progressiveness in [22] and feasibility in [24].

[7] This requirement together with the previous divergence property (item (ii) of Def. 3.1) and non-zenoness of the underlying TA are closely related to the notion of Strong I/O Feasibility of [24].

Example 3.4 CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) is widely used protocol on LANs on the MAC sublayer. It solves the problem of sharing a single channel in a broadcast network (a multi-access channel). When a station has data to send it first listens to the channel to check whether it is idle or busy. If the bus seems idle it begins sending the message, else it waits a random amount of time and then repeats the sensing operation. When a collision occurs, the transmission is aborted simultaneously in all the stations that were transmitting and they wait a random time to start all over again. We formally model the timing aspects of the protocol using I/O timed components (see Fig.1) based on the model presented in [21]. Sender components share a bus component. We suppose that the bus is a 10Mbps Ethernet with worst case propagation delay σ of 26 ms. Messages have a fixed length of 1024 bytes, and so the time λ to send a complete messages, including the propagation delay, is 808 ms. The bus is error-free, no buffering of incoming messages is allowed. Note that $\{SendOK_i, SendBusy_i\}$ is an output selection of sender i and the selection depend on the input actually enabled in the bus state. In fact, $SendBusy_i$ is enabled when the head of a message has already propagated. It takes at most σ to propagate the collision signal to all the senders. The sender stays at most δ in the transmission location. Note also that the sender non-deterministically makes a new attempt to send before 2σ elapsed since the last attempt. In models like Timed Process [19], it would be necessary for the sender component to issue a signal standing for the sensing of the bus state, and then wait for the status answer of the bus component (which can not arrive at zero time due to a "non-immediate response" constraint in that model). That two phase modeling idiom, common in software models, can be reduced in our modeling framework using appropriate Input and Output selections.

In [8], the reader can found how several examples taken from the literature are modeled as I/O TCs.

3.1 I/O Components: Composition and Non-Zenoness

Let us state some results that help to prove that a TA-model is non-zeno. Firstly, we will see how an admissible interface can be derived for the parallel composition of two compatible I/O TCs. This is a rather strong result which implies the following fact: given two compatible I/O TCs C_1 and C_2 then the composition, which turns out to be non-zeno, is also a I/O TC (i.e., $A_1 \parallel A_2$ is non-zeno and moreover it can be given an admissible I/O interface). Briefly, the new input interface is constituted by the original input selections that do not loose "selectivity property" of item (i) of Def. 3.1. That property is preserved for any input selection whenever there is no matching output selection and it is not properly included into another input selection.

Something similar can be done to build the new output interface. Since output selections that intersect with input selections of size greater than one

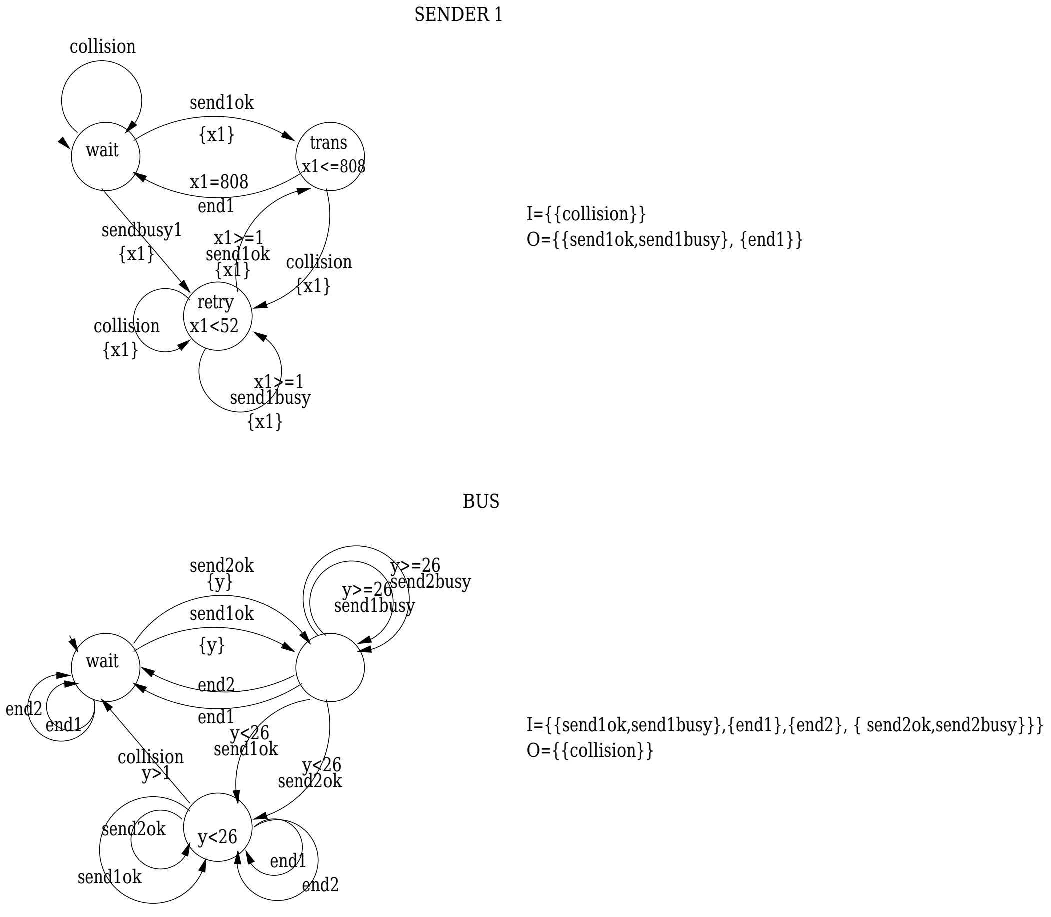

Fig. 1. I/O Components of the CSMA/CD Protocol

may loose the simultaneous availability property (item (iii) of Def. 3.1), they are not part of the new output selections. However, all the labels of those "lost" output selections can be safely added as output selections of size 1 (singletons trivially satisfy item (iii) of Def. 3.1). Thus, all exported labels of the components are exported in the composition. This fact is important to prove that this construction can be generalized to the parallel composition of n components:

Theorem 3.5 *Given an indexed set* $S = \{(A_i, (I^{A_i}, O^{A_i})\}_{1\leq i\leq n}$ *of* n *I/O TCs such that they are pair-wise compatible, we define the sets* $I_A = \bigcup_{1\leq i\leq n} I^{A_i}$, $I_{A_{>1}} = \bigcup_{1\leq i\leq n}\{I \in I^{A_i} / \#I > 1\}$, $O_A = \bigcup_{1\leq i\leq n} O^{A_i}$.

$(A, (I^A, O^A))$ *is a component where:*

$A = \|_{1\leq i\leq n} A_i$

$I^A = \{I \in I_A /\ \forall I' \in I_A : I \not\subset I' \wedge \forall O' \in O_A : I \cap O' = \emptyset\}$ *and,*

$O^A = \{O \in O_A /\ \forall I' \in I_{A_{>1}} : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in O_A \wedge \exists I' \in I_{A_{>1}} : I' \cap O \neq \emptyset\}$

Proof. See Appendix A. The basic idea is that, from the point of view of a component, its partners do not block its outputs: they just select them (items (i) and (iii) of Def. 3.1), also it does not require inputs to allow time elapse (item (ii) of Def. 3.1). On the other hand, a subset of I/O TCs can not engage themselves in an infinite activity in a finite interval of time since this is ruled out by item (iv) of Def. 3.1. □

In the example 3.4 the resulting interface of the parallel composition $A =^{def} SENDER_1 \parallel BUS$ is $I^A = \{\{Send2ok, Send2Busy\}, \{end2\}\}$, $O^A = \{\{end1\}$, $\{Send1ok\}$, $\{Send1Busy\}$, $\{collision\}\}$. Note that since simultaneous availability of output selection $\{Send1ok, Send1Busy\}$ is lost, they became singleton output-selections.

4 Applications of I/O TCs

Non Zeno Models:

Compatibility is a syntactical condition that ensures non-zenoness of the resulting parallel composition. As was already said, non-zenoness is a property required to perform some verification procedures. In [8,9] we model Real-Time System execution architectures by means of I/O TCs. We use I/O compatibility to ensure that I/O TCs modeling the connectors and the environment do not block the rest of the system (the tasks). As was already explained, I/O selections may be an useful mechanism to model in a single transition action/result on software entities.

Reduction:

Safety requirements are commonly modeled by means of virtual components (Observers) which are composed in parallel with the system under analysis (*SUA*) (e.g., [1,9]). In [8,10,11] we present a technique that, given the *SUA* and an observer, builds a smaller parallel composition equivalent to the original one up to the branching structure of the LTS. In a few words, we develop a technique that calculates the components that may be forgotten at each observer location since their future behavior do not influence the future evolution of the *SUA* up to the observer. Under some reasonable assumptions on the topology of the observers, those remaining sets (the relevant components) are proper subsets of the set of all components. The time needed for verification is drastically reduced in some cases. The core of that technique is a notion of potential "direct influence" of an automaton behavior over another automaton behavior. A naive solution would say that an automaton A potentially influences another automaton B iff they share a label. Unfortunately, this would lead to a rather large symmetrical overestimation. Then, by using the I/O interface attached to TAs, we are able to define an asymmetrical condition of behavioral influence that could be statically checked. That is, we provide a better overestimation of potential influence than simple label sharing. It is worth mentioning that the technique presented in [16] is based on a simpler notion of I/O interface than the one presented in this article.

The details of that "relevance calculus" using the definitions of this paper can be found in [8,10,11].

5 On Breaking Circularity in Assume-Guarantee Rules

The authors of [19] present a simple modularity principle for abstraction relations in Timed Processes. Assume-guarantee rule has an apparent circularity: to prove that $A \parallel B$ is a refinement of $A' \parallel B'$ it suffices to prove that (1) A is a refinement of A' assuming that the environment behaves like B', and (2) B is a refinement of B' assuming that the environment behaves like A'. For this rule to be true in our setting, we have to add a couple of conditions. Firstly, let us define when an state is non urgent from the point of view of outputs.

Definition 5.1 [Non-Urgent state] Given a I/O TC $C = (A, (I^A, O^A))$, a state q is not output urgent (denoted as $NU(q)$) iff there exists a run $r \in R_A(q)$ such that $0 < \tau_r$ and $Lab(r) \subseteq \epsilon_A$.

Definition 5.2 [Non-Blocking Extra Conditions] We say that an I/O TC satisfies the Non-Blocking Extra Conditions if and only if:

(i) Guards and Invariants are closed predicates (i.e., its binary relations are only $\leq$, $=$ or $\geq$).

(ii) Inputs do not disable nor enable urgent outputs: given a state $q \in Reach(A)$ and a label $i \in In_A$, if $q \mapsto_0^i q'$ then $NU(q)$ iff $NU(q')$[8].

It is easy to see that those properties are preserved by parallel composition $A = \parallel_{j \in J} A_j$. Firstly note that guards and invariants of A are inherited from the components A_j. For the item (ii) of Def. 5.2, if $q \mapsto_0^i q'$ then i is an unmatched input of one component, namely k, and thus q and q' just differ in the local state of k. Also, $NU(q)$ if and only if for all $j \in J$ $NU(\Pi_j(q))$ (since the set of internal labels of the composition A is the union of internal labels of components A_j). Therefore, $NU(q)$ iff $NU(q')$ since $NU(\Pi_k(q))$ iff $NU(\Pi_k(q'))$ and the rest of the components remain the same.

Theorem 5.3 (Assume/Guarantee) *Given the I/O TCs A, B, A', B' satisfying the non-blocking extra conditions such that A and B are I/O compatible, and A' and B' have the same I/O interface that A and B resp. If $(A \parallel B') \leq_{Exp_A} A'$ and $(A' \parallel B) \leq_{Exp_B} B'$ imply that $(A \parallel B) \leq_{Exp_A \cup Exp_B} (A' \parallel B')$.*

Proof. See appendix. □

[8] This property can be checked, for instance, using the verification engine of Kronos tool [13].

6 Conclusions and Future Work

We present I/O Timed Components, a simple compositional notion that extends Timed Automata "à la" Alur-Dill to get live non-zeno models [8], also providing some important methodological advantages like influence detection [10]. Assume-guarantee modular reasoning like [19] is obtained by adding a couple of constraints to I/O TCs without resorting to games. In our opinion, keeping non-zeno preservation conditions apart from the ones that break circularity in assume guarantee has practical and theoretical value.

We believe that admissible interfaces of a TA could be ordered according to the information it provides about availability of labels. That is, $(I_1, O_1) \leq (I_2, O_2)$ iff the admissibility of the interface (I_1, O_1) for a TA A implies the admisibility of (I_2, O_2) for A. We would like to study if this relationship between interfaces could be a declarative way to define the I/O interface of the composition.

We would like to study how to express and generalize our idea in term of Interface Theories [14] framework.

Conditions for assume-guarantee could be weakened, for instance: it is sufficient for A and B to satisfy that inputs do not enable urgent outputs, and for A' and B' to satisfy that inputs do not disable urgent outputs.

References

[1] Alpern, B., and F. Schneider, *Verifying Temporal Properties without Temporal Logic*, ACM Trans. Programming Languages and Systems, **11 (1)** (1989), 147–167.

[2] Alur, R., "Techniques for Automatic Verification of Real-Time Systems," Ph.D. thesis, Stanford University, 1991.

[3] Alur, R., C. Courcoubetis, and D. Dill, *Model-Checking for Real-Time Systems* In Proceedings of Logic in Computer Science, IEEE Computer Society, Los Alamitos, Calif, 414-425, 1990. Also in Information and Computation, **104 (1)** (1993) 2–34.

[4] Alur, R. and D. Dill, *A Theory of Timed Automata*, Theoretical Computer Science, **126** (1994) 183–235.

[5] Alur, R., and T. Henzinger, *Modularity for Timed and Hybrid Systems,* In Proceedings of CONCUR'97, LNCS 1243, 1997.

[6] Bengtsson, J., K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi, *UPPAAL- A Tool Suite for the Automatic Verification of Real-Time Systems,* In Proceedings of Hybrid Systems III, LNCS 1066, Springer Verlag, 1996, 232–243.

[7] Bornot, S., and J. Sifakis. *An Algebraic Framework for Urgency* To appear in Information and Computation, Academic Press.

[8] Braberman, V. “Modeling and Checking Real-Time System Designs,” Ph.D Thesis, Universidad de Buenos Aires, 2000. [Thesis]

[9] Braberman, V., and M. Felder, *Verification of Real-Time Designs: Combining Scheduling Theory with Automatic Formal Verification,* In Proceedings of 7th European Conf. on Software Eng./ 7th ACM SIGSOFT Symposium on the Foundations of Software Eng., (ESEC/FSE 99), LNCS 1687, Springer Verlag, Sept. 1999, 494–510.

[10] Braberman, V., D. Garbervetski, and A. Olivero, *Improving the Verification of Timed Systems using Influence Information* Submitted to TACAS 2002.

[11] Braberman, V., D. Garbervetski, and A. Olivero, “Influence Information to Improve the Verification of Timed Systems,” *Tech. Report DC-UBA 2001-003.*

[12] Clarke, E., O. Grumberg and D. Peled, “Model Checking”, MIT Press, January (2000), 330pp..

[13] Daws, C., A. Olivero, S. Tripakis, and S. Yovine, *The Tool KRONOS,* In Proceedings of Hybrid Systems III, LNCS 1066, Springer Verlag, 1996, 208–219.

[14] de Alfaro, L., and T.A. Henzinger *Interface Theories for Component-based Design,* In Proceedings of EMSOFT 01: Embedded Software, LNCS 2211, Springer Verlag, 148-165, 2001.

[15] Gawlick, R., R. Segala, J. Sogaard-Andersen, N. Lynch *Liveness in Timed and Untimed Systems,* In Proceedings of ICALP , LNCS 820, Springer Verlag, 166-177, 1994. Also in Information and Computation (1998).

[16] Garbervetsky, G. “Un Método de Reducción para la Composición de Sistemas Temporizados” Master Thesis, Universidad de Buenos Aires, 2000.

[17] Henzinger, T.A., X. Nicollin, J. Sifakis, and S. Yovine, *Symbolic model checking for real-time systems.* Information and Computation, **111(2)** (1994), 193–244.

[18] Larsen, K.G., F. Laroussinie, *CMC: A Tool for Compositional Model-Checking of Real-Time Systems,* In Proceedings of. FORTE-PSTV’98, 439-456, Kluwer Academic Publishers, 1998.

[19] Kurshan, R. P., S. Tasiran, R. Alur, and R. K. Brayton, *Verifying Abstractions of Timed Systems,* In Proceedings CONCUR 96, LNCS 1119, Springer Verlag, 1996.

[20] Merritt, M., F. Modugno, and M. Tuttle, *Time Constrained Automata,* In Proceedings of CONCUR’91, LNCS 527, Springer Verlag, 1991.

[21] Nicollin, X., J. Sifakis, and S. Yovine, *Compiling Real-Time Specification into Extended Automata,* IEEE Trans. on Soft. Eng.,Vol. **18 (9)** (1992), 794–804.

[22] Springintveld, J., F. Vaandrager, P. D’Argenio , *Testing Timed Automata*, To appear in Theoretical Computer Science, **254 (1-2)** (2001), 225–257.

[23] Tripakis, S. "L'Analyse Formelle des Systemès Temporisés en Practique", Phd. Thesis, Univesité Joseph Fourier, December 1998.

[24] Vaandrager, F., N. Lynch, *Action Transducers and Timed Automata,* In Proceedings of CONCUR'92, LNCS 630, 436-455, 1992.

[25] Yi, Wang, *Real-Time Behavior of Asynchronous Agents,* In Proceedings of CONCUR'90, LNCS 458, Springer Verlag, 1990.

[26] Yovine, S., *Model-Checking Timed Automata,* Embedded Systems, G. Rozemberg and F. Vaandrager eds., LNCS 1494, Springer Verlag, 1998.

Appendix

A On I/O Timed Components

Lemma A.1 *Given two I/O-compatible components* $C_1 = (A_1, (I^{A_1}, O^{A_1}))$ *and* $C_2 = (A_2, (I^{A_2}, O^{A_2}))$*, we define the sets* $I_A = I^{A_1} \cup I^{A_2}$*,* $I_{A_{>1}} = \{I \in I_A / \#I > 1\}$*,* $O_A = O^{A_1} \cup O^{A_2}$*.*

$C = (A, (I^A, O^A))$ *is a component where:*

$A = A_1 \parallel A_2$

$I^A = \{I \in I_A / \ \forall I' \in I_A : I \not\subset I' \wedge \forall O' \in O_A : I \cap O' = \emptyset\}$ *and,*

$O^A = \{O \in O_A / \ \forall I' \in I_{A_{>1}} : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in O_A \wedge \exists I' \in I_{A_{>1}} : I' \cap O \neq \emptyset\}$

Proof. The most difficult point is the proof that $A_1 \parallel A_2$ is indeed non-zeno regardless input transitions (item (ii), Def. 3.1). We will see that any state reachable *by a finite run* is not a timelock. Moreover, time can elapse avoiding input transitions. Let q be a reachable state by a finite run of $A_1 \parallel A_2$ then $q^1 = \Pi_{A_1}(q)$ and $q^2 = \Pi_{A_2}(q)$ are reachable states (by finite runs) of A_1 and A_2 resp. Let $k \in \mathbb{R}_{\geq 0}$ be a constant. From the definition of component, there must be runs r_1 and r_2 starting in q^1 and q^2 resp. of time length equal to k such that r_1 does not contain any transition with label in In_{A_1} and r_2 does not contain any transition with label in In_{A_2} (thus they do not contain any label in I^A). Now, we show a procedure to obtain a run r of $A_1 \parallel A_2$ from r_1 and r_2. To obtain such a run, we would need to merge r_1 and r_2. If the discrete transitions of r_1 and r_2 are sorted according to the time of occurrence, it is easy to combine them obtaining r till the first output-labeled transition which shared by the other automaton is found. To outline the merge, lets $r_1 = q^1 \mapsto_{t_1}^{l_1} q_1^1 \mapsto_{t_2}^{l_2} ... q_n^1$, and $r_2 = q^2 \mapsto_{t'_1}^{l'_1} q_1^2 \mapsto_{t'_2}^{l'_2} ... q_{n'}^2$. Now, suppose that $t_1 \leq t'_1$ (the other case is symmetrical) and l_1 is not shared by A_2 (or it is λ). Then - thanks to the parallel composition interleaving semantics - the resulting run r can be build as follows: $r = q \mapsto_{t_1}^{l_1} (q_1^1, q^2 + t_1)$ concatenated with the run obtained using the same procedure from $(q_1^1, q^2 + t_1)$ with $r_1 = q_1^1 \mapsto_{t_2}^{l_2} ...$, and $r_2 = q^2 + t_1 \mapsto^{l'_1}_{t'_1 - t_1} q^2_1 \mapsto^{l'_2}_{t'_2} ...q'_{n'}$. Clearly this procedure can be iterated finitely till we reach the end of both runs (the variant is sum of the number of transitions of both runs), thus obtaining a run of A of time length k, or till a shared label is found. [9]

Without loss of generality, let us suppose that the earliest still non synchronized shared output-transition $q_j \mapsto^o_0 q_{j+1}$ belongs to r_1 and $o \in O \in O^{A_1}$. Let $I \in I^{A_2}$, $I \subseteq O$ be the corresponding matching input selection (i.e., $o \in I$) by compatibility (item (iv), Def. 3.3). By definition of input selection, there is a transition labeled $i' \in I$ enabled in A_2 at the time of occurrence of that j^{th} transition. By definition of output selection, at q_j there must be also a discrete transition $q_j \mapsto^{i'}_0 s$. By applying this procedure, we can fix up both runs to get a finite run starting at q such that either it has time length k or it ends with an output transition into an intermediate state q'. Therefore, since both TA are non-transient for output labeled transitions (item (iv) of I/O interface admissibility), by repeating the whole procedure from those intermediate states (i.e., obtaining new r_1, r_2, etc.), a run of time length k is eventually built (if not, either the projection of that infinite run on A_1 or A_2 would show an infinite number of output-labeled transitions, and since there is a finite number of labels at least one output label would be repeated infinitely often thus violating item (iv) of I/O interface admissibility). The rest of the items of I/O interface are proven as follows:

- the new input and output labels are disjoint (input selections intersecting with an output selections are not part of the new interface).
- Input Selection Property (item (i)): given an state q of A and an input selection I of I^A, we know that I belongs either to I^{A_1} or to I^{A_2}. Without loose of generality, lets suppose that it belongs to I^{A_1}. Then, there exists $i \in I$ such that $\Pi_{A_1}(q) \mapsto^i_0 r$. We also know that if $i \in Labels(A_2)$ then $\{i\} \in I^{A_2}$ (input selection of size 1) and thus there exists s such that $\Pi_{A_2}(q) \mapsto^i_0 s$ and then $q \mapsto^i_0 (r, s)$.
- Output Selection Property (item (iii)): Similar to the previous one.
- finally, a run containing an infinite number of internal or output-labeled transitions is necessarily time-divergent (item (iv)). Indeed, since any run of A can projected into a run of A_1 and a run of A_2 and one of those runs must exhibit an infinite number of outputs or internal transitions and therefore diverge.

□

[9] Note that if one of the runs is empty then it just remains a set of discrete (0 time) transitions in the other run (both have originally the same time length) and therefore we can omit that suffix since we have already built a run of time length k.

Theorem 3.5

Given an indexed set $S = \{(A_i, (I^{A_i}, O^{A_i})\}_{1 \leq i \leq N}$ of N I/O TCs such that they are pair-wise compatible, we define the sets ${I_A}^n = \bigcup_{1 \leq i \leq n} I^{A_i}$, ${I_{A_{>1}}}^n = \bigcup_{1 \leq i \leq n} \{I \in I^{A_i} / \#I > 1\}$, ${O_A}^n = \bigcup_{1 \leq i \leq n} O^{A_i}$.

$C = (A, (I^A, O^A))$ is a component where:

$A = \|_{1 \leq i \leq N} A_i$

$I^A = \{I \in {I_A}^N / \ \forall I' \in {I_A}^N : I \not\subset I' \wedge \forall O' \in {O_A}^N : I \cap O' = \emptyset\}$ and,

$O^A = \{O \in {O_A}^N / \ \forall I' \in {I_{A_{>1}}}^N : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in {O_A}^N \wedge \exists I' \in {I_{A_{>1}}}^N : I' \cap O \neq \emptyset\}$

Proof. By induction. Base case is solved by the last lemma. Case n+1. By inductive hypothesis we know that

$C^n = (A^n, (I^n, O^n))$ is a component where:

$A^n = \|_{1 \leq i \leq n} A_i$

$I^n = \{I \in {I_A}^n / \ \forall I' \in {I_A}^n : I \not\subset I' \wedge \forall O' \in {O_A}^n : I \cap O' = \emptyset\}$ and,

$O^n = \{O \in {O_A}^n / \ \forall I' \in {I_{A_{>1}}}^n : I' \cap O = \emptyset\} \cup \{\{o\} / o \in O \in {O_A}^n \wedge \exists I' \in {I_{A_{>1}}}^n : I' \cap O \neq \emptyset\}$

We know that $C_{n+1} = (A_{n+1}, (I^{A_{n+1}}, O^{A_{n+1}})$ $(A_{n+1}, (I_{n+1}, O_{n+1}))$ is compatible with all $C_i = (A_i, (I^{A_i}, O^{A_i}))$ for $1 \leq i \leq n$. Let us show that is compatible with the interface for the n components but firstly let pinpoint some facts about the interface (I^n, O^n) of C^n.

(i) An exported label of C_i $(1 \leq i \leq n)$ is also exported by C^n. This comes from the following facts: (a) Input labels remain as input labels in the biggest input selection containing it except in the case that the input selection matches with an output selection (in that case, $I \subseteq O$), and (b) Output labels remain in the interface.

(ii) Input selections of I^n are input selections of some of its constituent components (i.e., if $I \in I^n$ then there exists $k \in \mathbb{N} : 1 \leq k \leq n$ such that $I \in I^{A_k}$)

(iii) If O is an output selection of O^n, if there exist $k \in \mathbb{N} : 1 \leq k \leq n$ such that $O \in O^{A_k}$ and no Input selection of size greater than one intersects it or there exists $O' \in O^{A_k}$ and $O = \{a\} \subseteq O'$, and there exists $m : 1 \leq m \leq n$ such that $I' \in I^{A_m}$ and $I' \subseteq O'$.

Therefore, suppose that A_{n+1} has a common label with $\|_{1 \leq i \leq n} A_i$ then, for instance, that label belongs to a k^{th} automata and therefore belongs to the interface of the components C_k and C_{n+1}. If that label is an output label of the C_{n+1} component, that label is exported by C^n due to the first observation.

The compatibility (item (ii) of Def. 3.3) $I \cap I' \neq \emptyset$ then either $\#I = 1$ or $\#I' = 1$ is trivially true due to the observation that input selections of C^n are input selections of the original components and the pairwise compatibility. Similarly, if an output selection O of $O^{A_{n+1}}$ intersects with some input selection

I of I^n then that input selection must be an input selection of some component and therefore that input selection must be included in the output selection (i.e., $I \subseteq O$). If an input selection I of $I^{A_{n+1}}$ intersects with some output selection of O^n namely O, then either it is an input selection of size one and it is trivially included in O, or, by the last observation, we know that there exists k such that $O \in O^{A_k}$ and thus $I \subseteq O$ (that is, due to pairwise compatibility, I must be the only input selection of size greater than 1 intersecting with O and then by the last observation O must belong to O^n).

Therefore, C^n and C_{n+1} are compatible components and by Lemma A.1 the pair (I'^{n+1}, O'^{n+1}) is a compatible interface, where:
$I'^{n+1} = \{I \in I^n \cup I^{A_{n+1}} / \; \forall I' \in I^n \cup I^{A_{n+1}} : I \not\subset I' \wedge \forall O' \in O^n \cup O^{A_{n+1}} : I \cap O' = \emptyset\}$ and,
$O'^{n+1} = \{O \in O^n \cup O^{A_{n+1}} / \; \forall I' \in I^n \cup I^{A_{n+1}} \wedge \#I > 1 : I' \cap O = \emptyset\} \cup$
$\{\{o\} / o \in O \in O^n \cup O^{A_{n+1}} \wedge \exists I' \in I^n \cup I^{A_{n+1}} \wedge \#I > 1 : I' \cap O \neq \emptyset\}$

It is not difficult to see that this interface is equivalent to (I^{n+1}, O^{n+1}) where
$I^{n+1} = \{I \in {I_A}^{n+1} / \; \forall I' \in {I_A}^{n+1} : I \not\subset I' \wedge \forall O' \in {O_A}^{n+1} : I \cap O' = \emptyset\}$ and,
$O^{n+1} = \{O \in {O_A}^{n+1} / \; \forall I' \in {I_{A_{>1}}}^{n+1} : I' \cap O = \emptyset\} \cup$
$\{\{o\} / o \in O \in {O_A}^{n+1} \wedge \exists I' \in {I_{A_{>1}}}^{n+1} : I' \cap O \neq \emptyset\}$

In fact, if we write I^{n+1} in terms of I^n we need to add the input selections of $I^{A_{n+1}}$ which are not strictly included in an Input Selection of other I^{A_k} and do not match with an output selection. On the other hand, we have to eliminate from I^n the input selections strictly included in an input selection of $I^{A_{n+1}}$ and the ones that match with an Output Selection of $O^{A_{n+1}}$. That is, $I^{n+1} = (I^n - \{I \in {I_A}^n / \; \exists I' \in I^{A_{n+1}} : I \subset I' \vee \exists O \in O^{A_{n+1}} : I \cap O \neq \emptyset\} \cup \{I' \in I^{A_{n+1}} / \; \forall I \in {I_A}^n : I' \not\subset I \wedge \forall O \in {O_A}^n : I' \cap O = \emptyset\}$

Let us show that the definition of I'^{n+1} specifies that manipulation: note that, though I^n may contain less Input Selections than the union of them $(\bigcup_{1 \leq i \leq n} I^{A_i})$, it is easy to see that (a) If an input selection of the union is not present in I^n then, either it is included on another input selection of I^n, or it intersects an output selection of O^n, and (b) $\exists O \in O^n : I \cap O \neq \emptyset$ iff $\exists k \leq n, O \in O^{A_k} : I \cap O \neq \emptyset$ (all output label remains). Therefore, the set $\{I' \in I^{A_{n+1}} / \; \forall I \in {I_A}^n : I' \not\subset I \wedge \forall O \in O_A^n : I' \cap O = \emptyset\}$ is equivalent to $\{I' \in I^{A_{n+1}} / \; \forall I \in I^n : I' \not\subset I \wedge \forall O \in O^n : I' \cap O = \emptyset\}$. This proves that in I'^{n+1}, the same input selections of $I^{A_{n+1}}$ filtered by the I^{n+1} are present. Finally, $I^n - \{I \in {I_A}^n / \; \exists I' \in I^{A_{n+1}} : I \subset I' \vee \exists O \in O^{A_{n+1}} : I \cap O \neq \emptyset\}$ is equivalent to $\{I \in I^n / \; \forall I' \in I^{A_{n+1}} : I \not\subset I' \wedge \forall O \in O^{A_{n+1}} : I \cap O = \emptyset\}$ and we can conclude that $I'^{n+1} = I^{n+1}$.

On the other hand, to write O^{n+1} in terms of O^n, the output selections of $O^{A_{n+1}}$ that do not match with input selections of size greater than one must be added as well as the singletons for the ones that match. Besides, the output selections of O^n must be checked against the input selections of I_{n+1} to eliminate and convert into singleton output selections the ones that match

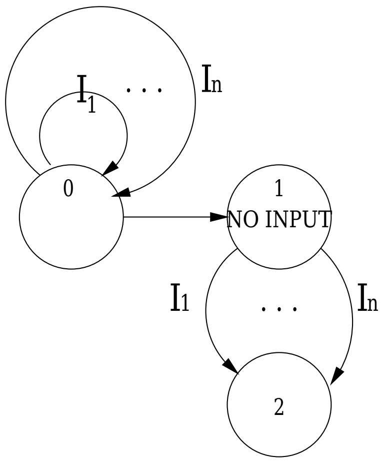

Fig. A.1. Observer for Checking Non-Zeno Regardless Input

with input selections of size greater than one. Again, this is specified by the definition of O'^{n+1}. $\square$

A.1 Guaranteeing I/O Admissibility

For the sake of self containment we provide sufficient syntactic constraints and checking-algorithms to guarantee that $(A, (I_A, O_A))$ is indeed a component.

For example, to satisfy the property of input being non-blocking, we can resort to the following syntactic property: $\forall I' \in I_A : \forall l \in Locs(A) : Inv(l) = \bigvee_{\{e:Label(e)\in I' \wedge src(e)=l\}} Guard(e)$. That is, while the invariant is valid at least one I'-labeled transition is enable.

To check that any output selection is simultaneously enabled one of the possible syntactic property is the following: $\forall l \in Locs(A), \forall o, o' \in O \in O^A$:

$$\bigvee_{\{e\in Edges(A):src(e)=l\wedge Label(e)=o\}} Guard(e) = \bigvee_{\{e'\in Edges(A):src(e')=l\wedge Label(e')=o'\}} Guard(e')$$

To check non-zenoness we use an observer automaton with three locations: location 1 is entered non-deterministically from initial location 0 and it is left to go to a trap location 2 whenever input occurs. Then, we ask whether $A \parallel Observer$ satisfies the following TCTL [17] formula : $\forall\Box(@ = 1 \rightarrow \exists \diamond_{\geq 1} @ = 1)$, i.e., whether time can elapse without traversing an input edge (See Fig. A.1).

For non-transientness of outputs, it suffices to require that no pair of outputs or internal events can occur closer than one time unit. This can be done by resorting to an observer TA or, alternatively, adding and checking some syntactic constraints on output edges, for instance, having a minimum delay guard on a clock reset in the potential previous events. Another alternative is checking strong non-zenoness [23] for sequences containing an infinite number of output labels.

B Assume Guarantee

Lemma B.1 (Extending event sequences) *Given a TA A with closed predicates (item (i), Def. 5.2) and a set L of labels, if $r \in R_A(q_0)$ then there exists $r' \in R_A(q_0)$ such that,*

(i) $r_L = r'_L$,

(ii) $\tau_r \leq \tau_{r'}$, *and*

(iii) $\exists k : 0 \leq k < \#r_L : \tau_{r'} - time(r'_L[k]) \in \mathbb{N}$.

Proof. This can be done by following a procedure on r that, step by step, shifts forward not visible transitions (i.e., not L-labeled transitions) to be at integer distance of a visible transition. □

Theorem 5.3

Given the I/O TCs A, B, A', B' satisfying the non-blocking extra conditions such that A and B are I/O compatible, and A' and B' have the same I/O interface that A and B resp. If $(A \parallel B') \leq_{Exp_A} A'$ and $(A' \parallel B) \leq_{Exp_B} B'$ imply that $(A \parallel B) \leq_{Exp_A \cup Exp_B} (A' \parallel B')$.

Proof. This is the sketch of the proof. Let $L = Exp_A \cup Exp_B$. Let $r \in R_{A\parallel B}(q_0)$ be a finite run such that there is no run in $A' \parallel B'$ of the same time length exhibiting the sequence of timed events r_L. First note there exists a maximum $k < \#r_L$ such that there exits $r' \in R_{A'\parallel B'}(q'_0)$ with $r_L[0...k] = r'_L$. There are two cases:

(i) There exists $r' \in R_{A'\parallel B'}(q'_0)$ such that $r_L[0...k] = r'_L$, and $time(r'_L[k+1]) \geq time(r_L[k+1])$.

(ii) For all $r' \in R_{A'\parallel B'}(q'_0)$, $\tau_{r'} \geq time(r_L[k+1])$ and $r_L[0...k] = r'_L$ implies $time(r'_L[k+1]) < time(r_L[k+1])$ (something urgent must happen before).

Before treating the cases, let r be a run in $A \parallel B$, and r' a run in $A' \parallel B'$ such that $r_L[0...k] = r'_L$. It is easy to see that we can project the run r into a run of A and a run of B. On the other hand, run r' can be projected into a run of A' and a run of B'. Due to the hypothesis on exported labels and labeling, we can safely recombine those runs to get a run $r_{AB'}$ of $A \parallel B'$ and a run $r_{A'B}$ of $A' \parallel B$ with the same time length of r'. Let q_A be the last state of $r_{AB'}$ projected into A, q_B the last state of $r_{A'B}$ projected into B, $q_{A'}$ the last state of $r_{A'B}$ projected into A', and $q_{B'}$ the last state of $r_{AB'}$ projected into B'.

Case i:

Suppose that $lab(r_L[k+1]) = o \in O^A_m$ (the other case is analogous). Then, at the last state of $r_{AB'}$ it is possible to execute some $o' \in O^A_m$ (at q_A A can perform any output in O^A_m and B' is receptive to at least one of them). From the fact that $(A \parallel B') \leq_{Exp_A} A'$, we can show the existence of a run $r_{A'}$ of A' such that $r_{A' Exp_{A'}}$ is equal to $r'_L[0...k+1]$ projected into $Exp_{A'}$ $(= Exp_A)$. Due to the fact that $O^A_m = O^{A'}_s$ is simultaneously enabled, there is a run $r'_{A'}$ of A' exhibiting $r_L[0...k+1]$ projected into $Exp_{A'}$. On the other hand, r shows that o is enabled at q_B. We can replace the original projection of $r_{A'B}$ on A' by the run $r'_{A'}$. Then, from the fact that $(A' \parallel B) \leq_{Exp_B} B'$ we can conclude that there is a run $r_{B'}$ in B' exhibiting $r_L[0...k+1]$ projected into $Exp_{B'}$. Combining the new runs $r'_{A'}$ and $r_{B'}$ for A' and B' resp., we conclude that there is a run r^* in $R_{A'\parallel B'}(q'_0)$ such that its exhibited sequence over L $r^*_L[0...k+1]$ is equal to $r_L[0...k+1]$, a contradiction.

Case ii:

Let $r' \in R_{A'\parallel B'}(q'_0)$ such that $\tau_{r'} \geq time(r_L[k+1])$ and $r'_L[0...k] = r_L[0...k]$, let r'' be the prefix run of r' such that $r''_L = r_L[0...k]$. By the previous lemma and the assumptions of this case, there exists another run ρ in $A' \parallel B'$ such that $\rho_L = r''_L = r_L[0...k]$, $\tau_{r''} \leq \tau_\rho < time(r_L[k+1])$, and $\exists s : 0 \leq s \leq k : \tau_\rho - time(r_L[s]) \in \mathbb{N}$. This means that, when runs show that in $A' \parallel B'$ something urgent must happen before time $time(r_L[k+1])$, there must exist a maximum value for its ocurrence. As shown, this follows from the fact that, for any r'' such that $r''_L = r_L[0...k]$ there exists a longer ρ such that $\rho_L = r''_L = r_L[0...k]$ and there are a finite number of those ρ (ρ ends at integer time distance of some event and before $time(r_L[k+1])$). We will show that any such ρ ends at a state where time can ellapse arriving to an absurd.

We know that $NU(q_A)$ and $NU(q_B)$. From the fact that $q \mapsto^i p$ implies $NU(q) \Rightarrow NU(p)$ (item (ii), Def. 5.2), A can wait, and any finite number of inputs of A (outputs of B') can not change this fact (an infinite number of outputs of B' would also imply time-divergence). Then, there exists $w \in R_{A\parallel B'}((q_A, q_{B'}))$ such that $\tau_w > 0$ and $Lab(w) \cap Out_A = \emptyset$. Since $\rho' = r_{AB'} \circ w$ ($r_{AB'}$ prolonged with w) is a run of $A \parallel B'$ there exists a run $r_{A'}$ of A' such that $\tau_{r_{A'}} = \tau_{\rho'}$ and $r_{A' Exp_{A'}} = \rho'_{Exp_{A'}}$. Thus, $r_{A'}$ can be split as $r'_{A'} \circ r''_{A'}$ such that $\tau_{r''_{A'}} = \tau_w$ and $r''_{A' Exp_{A'}} = w_{Exp_{A'}}$. Then from the last state of $r'_{A'}$ (denoted $q'_{A'}$) there exists a non-transient run ($r''_{A'}$) such that it exhibits no Output label. Then, there exists a state s in the run $r''_{A'}$ such that $NU(s)$. Using Def. 5.1 and item (ii) of Def. 5.2 ($q \mapsto^i p$ implies $NU(p) \Rightarrow NU(q)$) we can conclude $NU(q'_{A'})$. Analogously, $NU(q'_{B'})$. Therefore, the combination of those runs shows the possibility of $A' \parallel B'$ to exhibit σ' plus a positive time increment (the minimum possible increment between A' and B'). Thus, we arrive to an absurd. □