Liveness in Timed and Untimed Systems*

Rainer Gawlick† Roberto Segala† Jørgen Søgaard-Andersen‡ Nancy Lynch†

Abstract

When proving the correctness of algorithms in distributed systems, one generally considers *safety* conditions and *liveness* conditions. The Input/Output (I/O) automaton model and its timed version have been used successfully, but have focused on safety conditions and on a restricted form of liveness called fairness. In this paper we develop a new I/O automaton model, and a new timed I/O automaton model, that permit the verification of general liveness properties on the basis of existing verification techniques. Our models include a notion of *environment-freedom* which generalizes the idea of *receptiveness* of other existing formalisms, and enables the use of compositional verification techniques. The presentation includes an *embedding* of the untimed model into the timed model which preserves all the interesting attributes of the untimed model. Thus, our models constitute a *coordinated framework* for the description of concurrent and distributed systems satisfying general liveness properties.

Keywords: Automata, timed automata, I/O automata, liveness, environment-freedom, receptiveness, formal verification, simulation techniques, execution correspondence.

---

*Supported by NSF grant CCR-89-15206, by DARPA contracts N00014-89-J-1988 and N00014-92-J-4033, and by ONR contract N00014-91-J-1046. Also supported in part at the Technical University of Denmark by the Danish Technical Research Council. Appears as Technical Report MIT/LCS/TR-587, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 and as Technical Report ID-TR: 1993-128, Department of Computer Science, Technical University of Denmark, DK-2800 Lyngby, Denmark.

†Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139.

‡Department of Computer Science, Technical University of Denmark, DK-2800 Lyngby, Denmark.

Contents

1 Introduction

The increasing need for reliable software has led the scientific community to develop many formalisms for verification. Particularly important are formalisms that can model distributed and concurrent systems and those that can model real time systems, i.e., systems that rely on time constraints in order to guarantee correct behavior. Formalisms should be able to support verification of both *safety* and *liveness* properties [AS85]. Roughly speaking, a liveness property specifies that certain desirable events will eventually occur, while a safety property specifies that undesirable events will never occur.

In this paper, we present a *coordinated framework* that permits modeling and verification of safety and liveness properties for both timed and untimed systems. The framework consists of two models, one timed and one untimed, with an *embedding* of the untimed model into the timed model. Both models come equipped with notions of external behavior and of implementation, which are based simply on *traces*. The framework is intended to support a variety of verification techniques, including *simulation methods*, *compositional reasoning*, *algebraic methods*, and *temporal logic methods*.

A successful technique for the verification of safety properties and some special liveness properties is based on the *simulation method* of [AL91a, LV91, LV93a, LV93b, Jon91], applied to the *Input/Output automaton model* of [LT87] and to its generalization to the timed case [MMT91]. I/O automata are state machines with a labeled transition relation where the labels, also called *actions*, model communication. A key feature of I/O automata is the explicit distinction between their *input* and *output* actions, which characterize the events under the control of the environment and those under the control of the automaton, respectively. I/O automata can handle general safety properties and can also deal with a special kind of liveness, called *fairness*. Fairness captures the intuitive idea that each subcomponent of a composed system has fair chances to make progress. The notion of *implementation* for I/O automata, i.e., the way a concrete system is said to implement a more abstract specification, is expressed through *fair trace inclusion*, where a fair trace of an I/O automaton is a sequence of actions that can occur whenever the I/O automaton respects its fairness property. I/O automata can be composed in parallel, i.e., they can interact together so that they can be viewed as a single large system. An important property of I/O automata is that the implementation relation is *compositional* in the sense that it is always safe to replace a subcomponent in a large system with one of its implementations. Compositionality is needed for modular design techniques.

Despite its success, the I/O automaton model is not general enough to handle some recent verification work in [SLL93, LLS93]. In particular, [SLL93, LLS93] provide examples where fairness is not adequate as a liveness condition. Moreover, the work in [SLL93, LLS93] has shown the need for a connection between timed and untimed models to prove that an implementation that uses timing constraints correctly implements an untimed specification. The mutual exclusion algorithm of Fischer [Fis85, AL91b] is another instance of a timed implementation for an untimed specification.

This motivates a generalization of the I/O automaton model and its timed version to handle general liveness properties in such a way that the simulation based proof method still applies.

A simple and natural generalization is motivated by [AL93], which models a machine as a pair (A, L) consisting of an automaton and A and a subset L of its behaviors satisfying the desired liveness property. The implementation notion can then be expressed by *live trace inclusion* just as fair trace inclusion expresses implementation for I/O automata. The use of live trace inclusion as the implementation notion is motivated by the fact that the simulation based proof method is known to work for implementation notions based on some form of trace inclusion. Unfortunately, if L is not restricted, simple examples show that live trace inclusion is not compositional (cf. Examples 3.29 and 3.34).

In this paper we identify the appropriate restrictions on L, in both the untimed model and the timed model, so that live trace inclusion is compositional for the pair (A, L). A pair (A, L) satisfying these restrictions on L is called a *live I/O automaton* in the untimed model and a *live timed I/O automaton* in the timed model. The restrictions on L are given by a property called *environment-freedom*, which captures the intuitive idea that a live (timed) I/O automaton must not constrain its environment. The environment-freedom property is defined, using ideas from [Dil88], by means of a two-person game between a live (timed) I/O automaton and its environment. Specifically, the environment provides arbitrary inputs while the system tries to react so that it behaves according to its liveness condition. A live (timed) I/O automaton (A, L) has a *winning strategy* against its environment if A can respond to any environment move in such a way that it will always eventually satisfy its liveness condition L. If a live (timed) I/O automaton has a winning strategy, then it is said to be environment-free.

The definitions of the environment-freedom property in the untimed and the timed model are closely related. In particular, the environment-freedom property for the timed model is a natural extension of the environment-freedom property for the untimed model up to some technical details involving the so called *Zeno behaviors*. The close relation between the environment-freedom property in the untimed and the timed model allows the models to be tied together, thus permitting the verification of timed implementations of untimed specifications. Specifically, the paper presents a *patient* operator [NS92, VL92] that converts (untimed) live I/O automata into live timed I/O automata without timing constraints. The patient operator preserves the environment-freedom property and the live trace preorder relation of the untimed model. Thus, the patient operator provides the mechanism by which the timed and untimed models are unified into a coordinated framework.

Our models generalize several existing models. The fairness condition of I/O automata satisfies the environment-freedom property; thus, live I/O automata are a proper generalization of I/O automata. Environment-freedom also implies feasibility as defined in [LS89]. The failure free complete trace structures of [Dil88] are also properly generalized by our model. In the timed case, our model generalizes [MMT91] and the notion of *strong I/O feasibility* introduced in [VL92]. Finally, in contrast to [AL91b], our timed model does not give either the system or the environment control over the passage of time.

In order to extend the simulation based proof method to our model, we introduce an *execution correspondence theorem* which builds on a similar lemma of [LT87] by extending the result to some of the simulation relations studied in [LV93a, LV93b]. The execution correspondence theorem says that the existence of a simulation relation between two automata induces a strict correspondence between their behaviors. The paper shows how such a correspondence can be used to prove live trace inclusion.

We believe that our coordinated untimed and timed models comprise a good general framework for verification of concurrent systems. The models have already been used in [SLL93, LLS93] which deal with a non-trivial system, a communication protocol used in the Internet, and require all the new expressiveness and simulation tools provided in this paper.

After some preliminary definitions, given in Section 2, the paper is divided into four main sections. Section 3 presents the untimed model, Section 4 presents the timed model, Section 5 embeds the untimed model into the timed model by means of the *patient* operator, and Section 6 extends the simulation method to live (timed) I/O automata. The presentation of both the untimed and timed models starts with a general automaton model with liveness conditions in the style of [AL91b]; then the I/O distinction is introduced together with the environment-freedom property. The presentation of the untimed model also includes several examples that motivate the definition of environment-freedom and show that there does not seem to be any trivial generalization of our environment-freedom property that still leads to the compositionality of the live trace preorder. Once live (timed) I/O automata are defined for each model, the paper introduces the corresponding notions of implementation and compares our model with other existing models.

2 Preliminaries

Notation for Natural Numbers

Unless otherwise stated, indices like i, j, and k as well as the constant N range over the natural numbers $\mathbb{N}_0$. The notation $0 < i \leq \infty$, as well as the notation $0 < i < \infty$, states that i is a positive natural number. Similarly, the notation $\{0, 1, \ldots, \infty\}$ denotes the set of natural numbers.

Sequences

We use "list" and "sequence" synonymously. The empty sequence is denoted by ε. A finite sequence $l_1 = e_1 \ldots e_n$ and a sequence $l_2 = e_{n+1}e_{n+2}\ldots$ can be *concatenated*. The concatenation, written $l_1 \,\hat{}\, l_2$, or sometimes just $l_1 l_2$, is the sequence $e_1 \ldots e_n e_{n+1} e_{n+2} \ldots$.

A sequence l_1 is a *prefix* of a sequence l_2, written $l_1 \leq l_2$, if either $l_1 = l_2$, or l_1 is finite and there exists a sequence l_1' such that $l_2 = l_1 \,\hat{}\, l_1'$.

For any sequence l_2 and any finite sequence l_1 with $l_1 \leq l_2$, we denote by $l_2 - l_1$ the unique sequence l_1' such that $l_2 = l_1 \,\hat{}\, l_1'$.

For any non-empty sequence $l = e_1 e_2 e_3 \ldots$, define $head(l)$ to be e_1, the first element of l, and $tail(l)$ to be the sequence $e_2 e_3 \ldots$, the rest of l.

König's Lemma

The following lemma about digraphs is a generalization of König's Lemma. This generalization also appears in [LV93a]. A *root* in a digraph is a node with no incoming edges.

Lemma 2.1 (Generalization of König's Lemma)
Let G be an infinite digraph that satisfies the following properties:

1. *G has finitely many roots.*
2. *Each node of G has finite outdegree.*
3. *Each node of G is reachable from some root of G.*

Then there is an infinite path in G starting from some root.

Proof. The usual proof of König's Lemma [Kön26] extends to this case. ■

3 Untimed Systems

The discussion of untimed systems is organized as follows. Section 3.1 defines *automata*. Section 3.2 introduces *live automata* without I/O distinction. Section 3.3 defines *safe I/O automata* by adding an Input/Output distinction to safe automata, and introduces the standard parallel composition, action hiding, and action renaming operators found in the literature. Section 3.4 introduces *environment-freedom*, defines *live I/O automata*, and extends the operators of Section 3.3. Thus, the presentation separates the issue of liveness from that of I/O distinction and environment-freedom. Section 3.5 defines two preorder relations, the safe preorder and the live preorder, and shows in what sense the live preorder can express a notion of implementation. Section 3.6 compares our model with existing work.

3.1 Automata

The following definition of an automaton is given in the style of [LT87] and essentially describes a transition system.

Definition 3.1 (Automaton)
An *automaton* A consists of four components:

- a set $states(A)$ of states.
- a nonempty set $start(A) \subseteq states(A)$ of start states.
- an action signature $sig(A) = (ext(A), int(A))$ where $ext(A)$ and $int(A)$ are disjoint sets of external and internal actions, respectively. Denote by $acts(A)$ the set $ext(A) \cup int(A)$.

- a transition relation $steps(A) \subseteq states(A) \times acts(A) \times states(A)$. ■

Thus, an automaton is a state machine with labeled steps. Its action signature describes the interface with the environment. It specifies which actions model events that are visible from the environment and which ones model internal events.

An action a of automaton A is said to be *enabled* in state s if there exists a state s' such that the step (s, a, s') is an element of $steps(A)$.

An *execution fragment* α of an automaton A is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the execution fragment is finite, ending in a state,

$$\alpha = s_0 a_1 s_1 a_2 s_2 \cdots,$$

where each triplet (s_i, a_{i+1}, s_{i+1}) is an element $steps(A)$. Denote by $fstate(\alpha)$ the first state of α and, if α is finite, denote by $lstate(\alpha)$ the last state of α. Furthermore, denote by $frag^*(A), frag^\omega(A)$ and $frag(A)$ the sets of finite, infinite and all execution fragments of A, respectively. An *execution* is an execution fragment whose first state is a start state. Denote by $exec^*(A)$, $exec^\omega(A)$ and $exec(A)$ the sets of finite, infinite and all execution of A, respectively. A state s of A is *reachable* if there exists a finite execution of A that ends in s.

A finite execution fragment $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of A and an execution fragment $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of A can be *concatenated*. In this case the concatenation, written $\alpha_1 \frown \alpha_2$, is the execution fragment $s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$.

An execution fragment α_1 of A is a *prefix* of an execution fragment α_2 of A, written $\alpha_1 \leq \alpha_2$, if either $\alpha_1 = \alpha_2$, or α_1 is finite and there exists an execution fragment α_1' of A such that $\alpha_2 = \alpha_1 \frown \alpha_1'$.

Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ be an execution fragment. The *length* of α is the number of actions occurring in α. The length is infinite for infinite execution fragments. Define the ith *prefix*, ith *suffix*, and (i, j)*-segment* of α, for $0 \leq i \leq j \leq |\alpha|$, as

$$\begin{array}{lcl}
\alpha|_i & \triangleq & s_0 a_1 s_1 \cdots a_i s_i \\
{}_i|\alpha & \triangleq & \left\{ \begin{array}{ll} s_i a_{i+1} s_{i+1} \cdots & \text{if } i < |\alpha| \\ s_{|\alpha|} & \text{if } \alpha \text{ is finite and } i = |\alpha| \end{array} \right. \\
{}_i|\alpha|_j & \triangleq & s_i a_{i+1} s_{i+1} \cdots a_j s_j
\end{array}$$

The *trace* of an execution fragment α of an automaton A, written $trace_A(\alpha)$, or just $trace(\alpha)$ when A is clear from context, is the list obtained by restricting α to the set of external actions of A, i.e., $trace(\alpha) = \alpha \restriction ext(A)$, where $\restriction$ is the standard restriction operator on lists. Let β be a sequence of actions from $acts(A)$. Then, $trace_A(\beta)$, or just $trace(\beta)$ when A is clear from context, denotes the list obtained by restricting β to the set of external actions of A. For a set S of executions of an automaton A, denote by $traces_A(S)$, or just $traces(S)$ when A is clear from context, the set of traces of the executions in S. We say that β is a trace of an automaton A if there exists an execution α of A with $trace(\alpha) = \beta$. Denote by $traces^*(A), traces^\omega(A)$ and $traces(A)$ the sets of finite, infinite and all traces of A, respectively. Note, that a finite trace might be the trace of an infinite execution.

3.2 Live Automata

The automaton A of Definition 3.1 can be thought of as expressing the *safety* properties of a system, i.e, what always holds, or equivalently what is never supposed to happen. The *liveness* properties of a system, i.e., what must eventually happen, can be expressed by a subset L of the executions of its safe part A, as proposed in [AL93]. Thus, informally, a *live automaton* is a pair (A, L) where A is an automaton and L is a subset of its executions. The executions of L, which satisfy both the safety and liveness requirements of (A, L), are the only ones that can occur in the described system. However, in order to ensure that the set L of executions does not introduce any more safety than is already given by A, it should not be possible to violate L in a finite number of steps. As a consequence, any finite execution of A must be extendible to an execution in L. In fact, if the safe part A of live automaton (A, L) has a finite execution α that cannot be extended to an execution in L, then α cannot occur in the system described by (A, L), and thus L introduces the additional safety property that α cannot occur. Our restriction on the pair (A, L) implies that the pair $(exec(A), L)$ is *machine-closed* as defined in [AL93].

Definition 3.2 (Live automaton)

A *liveness condition* L for an automaton A is a subset of the executions of A such that any finite execution of A has an extension in L, i.e., for each $\alpha \in exec^*(A)$ there exists an $\alpha' \in frag(A)$ such that $\alpha \frown \alpha' \in L$.

A *live automaton* is a pair (A, L), where A is an automaton and L is a liveness condition for A. The executions of L are called the *live executions* of (A, L). ■

Informally, a liveness condition can be used to express (at least) two intuitively different requirements. First, a liveness condition can be used to specify assumptions about the long-term behavior of a system that are based on its physical structure. For example, it is reasonable to assume that two independent processes running in parallel are both allowed to make progress infinitely often. In a physical system this is ensured by executing the two processes on separate processors or by using a fair scheduler in a multiprogramming environment. The notion of fairness of I/O automata [LT87] exactly captures this particular physical assumption. Second, a liveness condition can be used to specify additional properties that a system is required to satisfy. For example, in a mutual exclusion problem we may require a process to eventually exit the critical region whenever it enters it.

Even though a liveness condition can express many specific intuitive ideas, for the purpose of this paper a liveness condition simply represents the set of executions that a system can exhibit whenever it is "working properly".

3.3 Safe I/O Automata

Our notion of safe I/O automaton is the same as the "unfair" I/O automaton of [LT87], i.e., the automaton obtained by removing the partition of the locally-controlled actions from an I/O automaton of [LT87].

Definition 3.3 (Safe I/O automaton)

A *safe I/O automaton* A is an automaton augmented with an *external action signature*, $esig(A) = (in(A), out(A))$, which partitions $ext(A)$ into input and output actions. In each state, each input action must be enabled. A is said to be *input-enabled.*

The internal and output actions of a safe I/O automaton A are referred to as the *locally-controlled* actions of A, written $local(A)$. Thus, $local(A) = int(A) \cup out(A)$. ■

The interaction between safe I/O automata is specified by the parallel composition operator. We use the synchronization style of [Hoa85, LT87], where automata synchronize on their common actions and evolve independently on the others. We also retain the constraint of [LT87] that each action is under the control of at most one automaton by defining parallel composition only for *compatible* safe I/O automata. Compatibility requires that each action be an output action of at most one safe I/O automaton. Furthermore, to avoid action name clashes, compatibility requires that internal action names be unique.

Definition 3.4 (Parallel composition)

Safe I/O automata $A_1, \ldots, A_N$ are *compatible* if for all $1 \leq i, j \leq N$ with $i \neq j$, the following conditions hold:

1. $out(A_i) \cap out(A_j) = \emptyset$
2. $int(A_i) \cap acts(A_j) = \emptyset$

The *parallel composition* $A_1 \parallel \cdots \parallel A_N$ of compatible safe I/O automata $A_1, \ldots, A_N$ is the safe I/O automaton A such that

1. $states(A) = states(A_1) \times \cdots \times states(A_N)$
2. $start(A) = start(A_1) \times \cdots \times start(A_N)$
3. $out(A) = out(A_1) \cup \cdots \cup out(A_N)$
4. $in(A) = (in(A_1) \cup \cdots \cup in(A_N)) \setminus out(A)$
5. $int(A) = int(A_1) \cup \cdots \cup int(A_N)$
6. $((s_1, \ldots, s_N), a, (s'_1, \ldots, s'_N)) \in steps(A)$ iff for all $1 \leq i \leq N$
   (a) if $a \in acts(A_i)$ then $(s_i, a, s'_i) \in steps(A_i)$
   (b) if $a \notin acts(A_i)$ then $s_i = s'_i$ ■

The executions of the parallel composition of compatible safe I/O automata $A_1, \ldots, A_N$ can alternatively be characterized as those executions that, when projected onto any component A_i, yield an execution of A_i. In particular, let $A = A_1 \parallel \cdots \parallel A_N$. First let s be a state of

A. Then, for any $1 \leq i \leq N$, define $s\lceil A_i$ to be s projected onto the i^{th} component. Now, let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ be an alternating sequence of states and actions such that $s_k \in \mathit{states}(A)$ and $a_k \in \mathit{acts}(A)$, for all k, and α ends in a state if it is a finite sequence. Define $\alpha\lceil A_i$, where $1 \leq i \leq N$, to be the sequence obtained from α by projecting its states onto their i^{th} component and by removing each action not in $\mathit{acts}(A_i)$ together with its following state.

Lemma 3.5

Let $A = A_1 \parallel \cdots \parallel A_N$. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ be an alternating sequence of states and actions such that $s_k \in \mathit{states}(A)$ and $a_k \in \mathit{acts}(A)$, for all k, and α ends in a state if it is a finite sequence. Then $\alpha \in \mathit{exec}(A)$ iff, for each i, $\alpha\lceil A_i \in \mathit{exec}(A_i)$ and $s_{j-1}\lceil A_i = s_j\lceil A_i$ whenever $a_j \notin \mathit{acts}(A_i)$.

Proof. The lemma is a direct consequence of Corollary 8 of [LT87]. ■

The parallel composition operator could alternatively be defined as a commutative and associative (up to isomorphism) binary operator. Thus, the parallel composition of N I/O automata could be obtained by applying the binary composition operator $N-1$ times. We use the N-ary parallel composition operator since it provides a simpler and more direct notation. Finally, the parallel composition operator is restricted to the composition of finitely many I/O automata in order to preserve compatibility with the timed model, where composition of infinitely many live timed I/O automata is not possible.

Parallel composition is typically used to build complex systems based on simpler components. However, some actions are meant to represent internal communications between the subcomponents of the complex system. The hiding operator of [LT87] changes some external actions into internal actions.

Definition 3.6 (Action hiding)

Let A be a safe I/O automaton and let Λ be a set of actions such that $\Lambda \subseteq \mathit{local}(A)$. Then define $A \setminus \Lambda$ to be the safe I/O automaton such that

1. $\mathit{states}(A \setminus \Lambda) = \mathit{states}(A)$
2. $\mathit{start}(A \setminus \Lambda) = \mathit{start}(A)$
3. $\mathit{in}(A \setminus \Lambda) = \mathit{in}(A)$
4. $\mathit{out}(A \setminus \Lambda) = \mathit{out}(A) \setminus \Lambda$
5. $\mathit{int}(A \setminus \Lambda) = \mathit{int}(A) \cup \Lambda$
6. $\mathit{steps}(A \setminus \Lambda) = \mathit{steps}(A)$ ■

Lemma 3.7

Let A be a safe I/O automaton and $\Lambda \subseteq \mathit{local}(A)$. Then $\mathit{exec}(A \setminus \Lambda) = \mathit{exec}(A)$.

Proof. The lemma is a direct consequence of Corollary 13 of [LT87]. ∎

Several processes might be identical except for their actions' names. The processes of a token ring communication network provide a classical example. Such processes can be specified by first defining a generic automaton representing the functionality of a generic token ring process, and then creating an instance for each process by renaming the actions of the generic automaton via an action renaming operation. Action renaming can also be used to resolve name clashes that lead to incompatibilities in Definition 3.4.

Definition 3.8 (Action renaming)

A mapping ρ from actions to actions is *applicable* to a safe I/O automaton A if it is injective and $acts(A) \subseteq dom(\rho)$. Given a safe I/O automaton A and a mapping ρ applicable to A, define $\rho(A)$ to be the safe I/O automaton such that

1. $states(\rho(A)) = states(A)$
2. $start(\rho(A)) = start(A)$
3. $in(\rho(A)) = \rho(in(A))$
4. $out(\rho(A)) = \rho(out(A))$
5. $int(\rho(A)) = \rho(int(A))$
6. $steps(\rho(A)) = \{(s, \rho(a), s') \mid (s, a, s') \in steps(A)\}$ ∎

Lemma 3.9

Let A be a safe I/O automaton and let ρ be a mapping applicable to A. For each execution $\alpha \in exec(A)$, let $\rho(\alpha)$ be the sequence that results from replacing each occurrence of every action a in α by $\rho(a)$. Then $exec(\rho(A)) = \{\rho(\alpha) \mid \alpha \in exec(A)\}$.

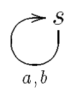
Proof. The lemma is a direct consequence of Lemma 15 of [LT87]. ∎

3.4 Live I/O Automata

In defining live I/O automata one could follow the approach of Definition 3.2 and define a live I/O automaton to be a pair (A, L) where A is a safe I/O automaton and L is a liveness condition for A. However, such a naive definition would not capture the fact that a live I/O automaton should behave properly independently of the inputs provided by its environment. Given the structure of our liveness conditions, such independence from the environment will prove to play a fundamental role in the proofs for the closure of live I/O automata under parallel composition and the substitutivity of our trace based preorders.

Example 3.10

Let A be a the safe I/O automaton described by the diagram,

$A:$ 

where a is an input action and b is an output action. Let L be the set of executions of A containing at least five occurrences of action a. L is trivially a liveness condition for A; however, the pair (A, L) would not behave properly if the environment does not provide more than four a actions (recall that behaving properly means being an execution of L). ■

Some of the problems arising from the requirement that a live I/O automaton should behave properly independently of the inputs provided by its environment are addressed in [Dil88, AL93]. Their solutions lead to the notion of *receptiveness*. Intuitively a system is receptive if it behaves properly independently of the inputs provided by its environment, or equivalently, if it does not constrain its environment. The interaction between a system and its environment is represented as a two person game where the environment moves consist of providing an arbitrary finite number of inputs, i.e., in our model, a finite number of input actions, and the system moves consist of performing at most one local step, i.e., in our model, at most one locally-controlled step. A system is *receptive* if it has a way to win the game (i.e., to behave properly) independently of the moves of its environment. The fact that an environment move can include at most a finite number of actions represents the natural requirement that the environment cannot be infinitely faster than the system.

The behavior of the system during the game is determined by a *strategy*. In our model a strategy consists of a pair of functions (g, f). The function g decides which of the possible states the system reaches in response to any given input action; the function f determines the next move of the system. The move can be a local step or no step ($\perp$ move).

Definition 3.11 (Strategy)

Consider any safe I/O automaton A. A *strategy* defined on A is a pair of functions (g, f) where $g : exec^*(A) \times in(A) \rightarrow states(A)$ and $f : exec^*(A) \rightarrow (local(A) \times states(A)) \cup \{\perp\}$ such that

1. $g(\alpha, a) = s$ implies $\alpha a s \in exec^*(A)$
2. $f(\alpha) = (a, s)$ implies $\alpha a s \in exec^*(A)$ ■

In the game between the environment and the system the moves of the environment are represented as an infinite sequence $\mathcal{I}$, called an *environment sequence*, of input actions interleaved with infinitely many λ symbols. The symbol λ represents the points at which the system is allowed to move. The occurrence of infinitely many λ symbols in an environment sequence guarantees that each environment move consists of only finitely many input actions.

Suppose the game starts after a finite execution α. Then the *outcome* of a strategy (g, f), given α and an environment sequence $\mathcal{I}$, is the extension of α obtained by applying g at each input action in $\mathcal{I}$ and f at each λ in $\mathcal{I}$.

Definition 3.12 (Outcome of a strategy)

Let A be a safe I/O automaton and (g, f) a strategy defined on A. Define an *environment sequence* for A to be any infinite sequence of symbols from $in(A) \cup \{\lambda\}$ with infinitely many occurrences of λ. Then define $R_{(g,f)}$, the *next-function induced by* (g, f) as follows: for any finite execution α of A and any environment sequence $\mathcal{I}$ for A,

$$R_{(g,f)}(\alpha, \mathcal{I}) = \begin{cases} (\alpha a s, \mathcal{I}') & \text{if } \mathcal{I} = \lambda\mathcal{I}',\ f(\alpha) = (a, s) \\ (\alpha, \mathcal{I}') & \text{if } \mathcal{I} = \lambda\mathcal{I}',\ f(\alpha) = \bot \\ (\alpha a s, \mathcal{I}') & \text{if } \mathcal{I} = a\mathcal{I}',\ g(\alpha, a) = s \end{cases}$$

Let α be any finite execution of A and $\mathcal{I}$ any environment sequence for A. The *outcome sequence of* (g, f) *given* α *and* $\mathcal{I}$ is the unique infinite sequence $(\alpha^n, \mathcal{I}^n)_{n\geq 0}$ that satisfies:

- $(\alpha^0, \mathcal{I}^0) = (\alpha, \mathcal{I})$ and
- for all $n > 0$, $(\alpha^n, \mathcal{I}^n) = R_{(g,f)}(\alpha^{n-1}, \mathcal{I}^{n-1})$.

Note, that $(\alpha^n)_{n\geq 0}$ forms a chain ordered by *prefix*.

The *outcome* $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})$ of the strategy (g, f) given α and $\mathcal{I}$ is the execution $\lim_{n\to\infty} \alpha^n$, where $(\alpha^n, \mathcal{I}^n)_{n\geq 0}$ is the outcome sequence of (g, f) given α and $\mathcal{I}$ and the limit is taken under prefix ordering. ■

Lemma 3.13

Let A be a safe I/O automaton and (g, f) a strategy defined on A. Then for any finite execution α of A and any environment sequence $\mathcal{I}$ for A, the outcome $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})$ is an execution of A such that $\alpha \leq \mathcal{O}_{(g,f)}(\alpha, \mathcal{I})$. ■

The concepts of strategies and outcomes are used to define formally the property that a system does not constrain its environment. This property is called *environment-freedom*. Informally, environment-freedom requires that there exists a strategy, called an *environment-free strategy*, that allows the system to win every game against its environment. In other words, every outcome of the environment-free strategy should be an element of L. An important feature of the definition of environment-freedom is that it considers outcomes where the environment-free strategy for (A, L) is applied after any finite execution of A. The discussion following the definition shows that this feature leads to a clean separation of safety and liveness properties.

Definition 3.14 (Environment-freedom)

A pair (A, L), where A is a safe I/O automaton and $L \subseteq exec(A)$, is *environment-free* if there exists a strategy (g, f) defined on A such that for any finite execution α of A and any environment sequence $\mathcal{I}$ for A, the outcome $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})$ is an element of L. The strategy (g, f) is called an *environment-free strategy* for (A, L). ∎

Lemma 3.15

Consider the pair (A, L), where A is a safe I/O automaton and $L \subseteq exec(A)$. If (A, L) is environment-free, then L is a liveness condition for A.

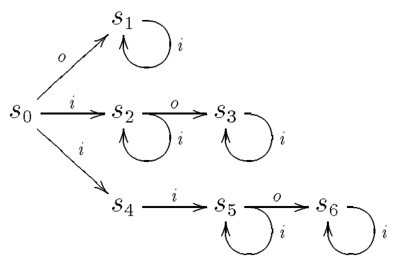
Proof. Consider any environment-free strategy (g, f) for (A, L), any finite execution α of A, and any environment sequence $\mathcal{I}$ for A. Then, since (g, f) is an environment-free strategy for (A, L), the outcome $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})$ is an element of L. Furthermore, by Lemma 3.13, $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})$ is an extension of α. Hence, any finite execution of A has an extension in L. ∎

Definition 3.16 (Live I/O automaton)

A *live I/O automaton* is a pair (A, L), where A is a safe I/O automaton and $L \subseteq exec(A)$, such that (A, L) is environment-free. ∎

Example 3.17

Consider the safe I/O automaton A described by the transition diagram below.



The unique start state of A is s_0. Action i is an input action and action o is an output action. Let L be the liveness condition for A consisting of the set of executions of A with at least one occurrence of action o. The pair (A, L) is not environment-free. Specifically, consider the finite execution $\alpha = s_0 i s_4$ and the environment sequence $\mathcal{I} = \lambda\lambda\lambda\cdots$. Performing action o after reaching state s_4 requires receiving an input i. Therefore, there is no strategy whose outcome given α and $\mathcal{I}$ is an execution in L.

Define a new automaton A' from A by removing states s_4, s_5, s_6, and let L' be the set of executions of A' containing at least one occurrence of action o. Then the pair (A', L') is environment-free. Function f chooses to perform action o whenever applied to an execution ending in s_0 or s_2 and chooses $\perp$ otherwise; function g always moves to the only possible next state. In [AL93] the pair (A, L) is said to be *realizable* and is identified with its *realizable part* (A', L'). Realizability can be defined in our model by considering only those outcomes $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})$ where α consists of a start state. However, the approach of [AL93] implies that state s_4 should never be reached in (A, L), thus adding new safety requirements to A via L. It is the requirement of our environment-freedom condition that $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I}) \subseteq L$ for all $\alpha \in \mathit{exec}^*(A)$ which ensures that such new safety properties are not introduced.

Let B be a safe I/O automaton that performs its unique output action i just once, and let L_B be the set of executions of B. The pair (B, L_B) is trivially a live I/O automaton. It is easy to see that the parallel composition $(A, L) \parallel (B, L_B)$ is not even a live automaton. Thus, realizable pairs are not closed under parallel composition. The reader is referred to Section 3.6 for more details. ■

Remark 3.18

Note that for a pair (A, L) to be environment-free, all input actions must be enabled in all reachable states. Consider any reachable state s of A and any finite execution α of A leading to state s. Since α must be extendible for all input actions that the environment might provide, each input action must be enabled in s. For this reason safe I/O automata are required to be input-enabled by definition. ■

The parallel composition, hiding and renaming operators can now be extended to live I/O automata by using the results of Lemmas 3.5, 3.7, and 3.9.

Definition 3.19 (Parallel composition)

Live I/O automata $(A_1, L_1), \ldots, (A_N, L_N)$ are *compatible* iff the safe I/O automata $A_1, \ldots, A_N$ are compatible.

The *parallel composition* $(A_1, L_1) \parallel \cdots \parallel (A_N, L_N)$ of compatible live I/O automata $(A_1, L_1), \ldots, (A_N, L_N)$ is defined to be the pair (A, L) where $A = A_1 \parallel \cdots \parallel A_N$ and $L = \{\alpha \in \mathit{exec}(A) \mid \alpha\lceil A_1 \in L_1, \ldots, \alpha\lceil A_N \in L_N\}$. ■

Definition 3.20 (Action hiding)

Let (A, L) be a live I/O automaton and let Λ be a set of actions such that $\Lambda \subseteq \mathit{local}(A)$. Then define $(A, L) \setminus \Lambda$ to be the pair $(A \setminus \Lambda, L)$. ■

Definition 3.21 (Action renaming)

A mapping ρ from actions to actions is *applicable* to a live I/O automaton (A, L) if it is applicable to A. Let α be an execution of (A, L). Define $\rho(\alpha)$ to be the sequence that results from replacing each occurrence of every action a in α by $\rho(a)$. Given a live I/O automaton (A, L) and a mapping ρ applicable to (A, L), define $\rho((A, L))$ to be the pair $(\rho(A), \{\rho(\alpha) \mid \alpha \in L\})$. ■

All the operators above are closed for live I/O automata in the sense that they produce a new live I/O automaton whenever applied to live I/O automata.

Proposition 3.22 (Closure of action hiding)

Let (A, L) be a live I/O automaton and let $\Lambda \subseteq local(A)$. Then $(A, L) \setminus \Lambda$ is a live I/O automaton.

Proof. To see that $(A, L) \setminus \Lambda$ is a live I/O automaton it is sufficient to note that $A \setminus \Lambda$ is a safe I/O automaton, $L \subseteq exec(A \setminus \Lambda)$ (by Lemma 3.7), and that an environment-free strategy for (A, L) is also an environment-free strategy for $(A, L) \setminus \Lambda$. ■

Proposition 3.23 (Closure of action renaming)

Let (A, L) be a live I/O automaton and let ρ be a mapping applicable to (A, L). Then $\rho((A, L))$ is a live I/O automaton.

Proof. To see that $\rho((A, L))$ is a live I/O automaton it is sufficient to note that $\rho(A)$ is a safe I/O automaton, $\{\rho(\alpha) \mid \alpha \in L\} \subseteq exec(\rho(A))$ (by Lemma 3.9), and that an environment-free strategy for (A, L) can easily be modified to be an environment-free strategy for $\rho((A, L))$. Specifically, since ρ is injective, any environment-free strategy (g, f) for (A, L) can be transformed into a new environment-free strategy (g_ρ, f_ρ) for $\rho((A, L))$ where

$$
\begin{aligned}
g_\rho(\rho(\alpha), \rho(a)) &= g(\alpha, a) \\
f_\rho(\rho(\alpha)) &= \begin{cases} (\rho(a), s) & \text{if } f(\alpha) = (a, s) \\ \perp & \text{if } f(\alpha) = \perp \end{cases}
\end{aligned}
$$

■

The analysis for the parallel composition operator is more complicated and needs some technical lemmas. Given $(A, L) = (A_1, L_1) \parallel \cdots \parallel (A_N, L_N)$, it is easy to see that A is a safe I/O automaton since its definition is based on the parallel composition of safe I/O automata. However, it is not as easy to see that the pair (A, L) is environment-free, and hence a live I/O automaton. The proof that (A, L) is environment-free uses a strategy (g, f) for (A, L) based on environment-free strategies (g_i, f_i) for each of the (A_i, L_i), and shows that (g, f) is an environment-free strategy for (A, L).

Function g should compute, given input a, the next state according to the g_i functions of those components of A for which a is an input action, and simply leave the state unchanged for those components where a is not an action.

Function f must ensure that every component of A gets a chance to control a step of A infinitely often. This fact accounts for much of the complexity in the definition of (g, f). Ensuring that every component of A gets a chance to control a step infinitely often would most naturally be done by assigning the control of steps to components in a round robin fashion. The round robin based approach, however, would give rise to a technical problem in the definition of f: since the only argument to f is a finite execution α, the component whose turn it is to control the step in the round robin schedule must be determined from α. Unfortunately, the finite execution α does not include enough information to make this determination. Consider the following scenario. Assume that it is component A_i's turn to control the step after a finite execution α. Assume further that A_i decides to perform a $\perp$ move and that the next input is a λ symbol. In this case α will not change and, thus, it will again be A_i's turn to control the next step. Therefore, the round robin protocol is violated. The problem is, of course, that $\perp$ and λ moves are "invisible" in α. One solution to this problem would be to let f be a function of "extended" executions that contain information about $\perp$ and λ moves. The problem with this solution, however, is that it becomes messy due to the fact that this new notion of execution must keep track of $\perp$ and λ moves of subcomponents of components, and so on. An alternative solution, adopted in our definition of f, uses the number of locally-controlled actions in α to determine which component controls a step. If the component controlling a step wants to perform a $\perp$ move but another component wants to perform a local step, a component wanting to perform the local step is given control. Thus a new locally-controlled action is added ensuring that another component will be given the opportunity to control the next step. Only if all components want to perform $\perp$ moves, does f yield a $\perp$ move.

One final technicality in the definition of f is that it uses the g_i functions. In particular, if a component performs a local step with action a, action a might be an input action of other components. In this case, the definition of f will need the g_i functions of all those components for which action a is an input action.

Definition 3.24 (Parallel composition of strategies)

Let $A = A_1 \parallel \cdots \parallel A_N$ be the parallel composition of compatible safe I/O automata $A_1, \ldots, A_N$. For each finite execution $\alpha \in \mathit{exec}^*(A)$, let $l(\alpha)$ be the number of occurrences of locally-controlled actions of A in α, i.e., $l(\alpha) = |\alpha \restriction \mathit{local}(A)|$, and let $p(\alpha) = (l(\alpha) \bmod N) + 1$. Let, for each $1 \leq i \leq N$, (g_i, f_i) be a strategy defined on A_i.

The *parallel composition* $(g_1, f_1) \parallel \cdots \parallel (g_N, f_N)$ of the strategies $(g_1, f_1), \ldots, (g_N, f_N)$ is the pair of functions (g, f) defined as follows.

Function $g : \mathit{exec}^*(A) \times \mathit{in}(A) \to \mathit{states}(A)$ is defined such that $g(\alpha, a) = s$ where, for each component A_i,

$$s\lceil A_i = \begin{cases} g_i(\alpha\lceil A_i, a) & \text{if } a \in \mathit{in}(A_i) \\ \mathit{lstate}(\alpha)\lceil A_i & \text{otherwise} \end{cases}$$

Function $f : \mathit{exec}^*(A) \to (\mathit{local}(A) \times \mathit{states}(A)) \cup \{\perp\}$ is defined for α based on the following cases:

1. If there exists A_i such that $f_i(\alpha\lceil A_i) \neq \perp$, then define k as follows. If $f_{p(\alpha)}(\alpha\lceil A_{p(\alpha)}) \neq \perp$, then $k = p(\alpha)$. Otherwise, k is the minimum index i such that $f_i(\alpha\lceil A_i) \neq \perp$. Now let

$f_k(\alpha\lceil A_k) = (a, s_k)$ and define $f(\alpha) = (a, s)$ where, for each component A_i,

$$s\lceil A_i = \begin{cases} s_k & \text{if } i = k \\ g_i(\alpha\lceil A_i, a) & \text{if } a \in in(A_i) \\ lstate(\alpha)\lceil A_i & \text{otherwise} \end{cases}$$

2. If $f_i(\alpha\lceil A_i) = \bot$ for all A_i
   then $f(\alpha) = \bot$. ■

It is easy to see that the strategy of Definition 3.24 is indeed a strategy defined on A.

Lemma 3.25

Let $A_1, \ldots, A_N$ be compatible safe I/O automata and let, for each $1 \leq i \leq N$, (g_i, f_i) be a strategy defined on A_i. Then $(g_1, f_1) \parallel \cdots \parallel (g_N, f_N)$ is a strategy defined on $A_1 \parallel \cdots \parallel A_N$.

Proof. Let $A = A_1 \parallel \cdots \parallel A_N$ and $(g, f) = (g_1, f_1) \parallel \cdots \parallel (g_N, f_N)$. From Definition 3.4, we know that A is a safe I/O automaton. Now the proof is a simple cases analysis on the different cases of Definition 3.24. In fact, for each one of those cases, it is sufficient to show that f and g give legal steps of A. ■

The following lemma is the key lemma for proving that the strategy of Definition 3.24 is environment-free if the component strategies are environment-free. The lemma shows that the projection of an outcome of the composed strategy onto any A_i is an outcome of the strategy (g_i, f_i). Intuitively, this means that, even though the composed system uses its composed strategy to find its outcome, it still looks to each component as if it was using its own component strategy.

Lemma 3.26

Let $A_1, \ldots, A_N$ be compatible safe I/O automata and, for each $1 \leq i \leq N$, let (g_i, f_i) be a strategy defined on A_i. Let $A = A_1 \parallel \cdots \parallel A_N$ and let $(g, f) = (g_1, f_1) \parallel \cdots \parallel (g_N, f_N)$.

Furthermore, let α be an arbitrary finite execution of A, $\mathcal{I}$ be an arbitrary environment sequence for A, and i, with $1 \leq i \leq N$, be an arbitrary index. Then, there exists an environment sequence $\mathcal{I}_i$ for A_i such that $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})\lceil A_i = \mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i, \mathcal{I}_i)$.

Proof. From Definition 3.4 we know that A is a safe I/O automaton. Furthermore, by Lemma 3.25, (g, f) is a strategy defined on A.

Let $R_{(g,f)}$ and $R_{(g_i,f_i)}$ be the next-functions induced by (g, f) and (g_i, f_i), respectively. Also, let $(\alpha^n, \mathcal{I}^n)_{n\geq 0}$ be the outcome sequence of (g, f) given α and $\mathcal{I}$. Then $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I}) = \lim_{n\to\infty} \alpha^n$. Finally, for any finite execution $\alpha' \in exec^*(A)$, let $l(\alpha')$ be the number of occurrences of locally-controlled actions of A in α', i.e., $l(\alpha') = |\alpha' \upharpoonright local(A)|$, and let $p(\alpha') = (l(\alpha') \bmod N) + 1$. (See Definition 3.24.)

The first step of the proof consists of constructing an environment sequence $\mathcal{I}_i$ such that $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})\lceil A_i = \mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i, \mathcal{I}_i)$. The construction of $\mathcal{I}_i$ is inductive on n. Along with $\mathcal{I}_i$, the inductive definition constructs an outcome sequence $(\alpha_i^j, \mathcal{I}_i^j)_{j\geq 0}$ of (g_i, f_i) given $\alpha\lceil A_i$ and $\mathcal{I}_i$, and a total nondecreasing mapping m with signature $\mathbb{N}_0 \to \mathbb{N}_0$, which, informally, maps elements of the outcome sequence $(\alpha^n, \mathcal{I}^n)_{n\geq 0}$ to their corresponding elements of the outcome sequence $(\alpha_i^j, \mathcal{I}_i^j)_{j\geq 0}$. The n^{th} step of the inductive construction of $\mathcal{I}_i$ defines the $(m(n-1)+1)^{\text{th}}, \ldots, m(n)^{\text{th}}$ elements of $\mathcal{I}_i$, which are denoted by $\mathcal{I}_{i,m(n-1)+1}, \ldots, \mathcal{I}_{i,m(n)}$.

Along with the inductive definitions, three properties are proven: the first property shows the correspondence between α^n and $\alpha_i^{m(n)}$; the second and third property are used to show that $(\alpha_i^j, \mathcal{I}_i^j)_{j\geq 0}$ is indeed an outcome sequence of (g_i, f_i) given $\alpha\lceil A_i$ and $\mathcal{I}_i$. Formally, the properties are written as follows.

P1 $\alpha^n\lceil A_i = \alpha_i^{m(n)}$.

P2 If $n > 0$ and $m(n) = m(n-1)+1$ then $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)})$.

P3 If $n > 0$ and $m(n) = m(n-1)+2$ then $(\alpha_i^{m(n)-1}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)-1})$ and $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n)-1}, \mathcal{I}_{i,m(n)})$.

The base part of the proof is trivial. The inductive part of the proof is divided into cases based on the definition of $R_{(g,f)}$ (c.f. Definition 3.12) and then subcases based on the definition of (g, f) (c.f. Definition 3.24).

Base case $n = 0$**:**

Define: $m(0) = 0$
$\alpha_i^{m(0)} = \alpha^0\lceil A_i$

P1 By definition.

P2 Vacuously satisfied.

P3 Vacuously satisfied.

Inductive step $n > 0$**:**

Assume P1–P3 hold for all $k < n$. The definition of $R_{(g,f)}$ suggests three cases which are considered in order.

Case 1 $(\alpha^n, \mathcal{I}^n) = (\alpha^{n-1}as, \mathit{tail}(\mathcal{I}^{n-1}))$ where $f(\alpha^{n-1}) = (a, s)$ and $\mathit{head}(\mathcal{I}^{n-1}) = \lambda$.

The definition of f in Definition 3.24 suggests the following sub cases:

Case 1.1 $p(\alpha^{n-1}) = i$ and $a \notin \mathit{acts}(A_i)$.

Define: $m(n) = m(n-1)+1$
$\alpha_i^{m(n)} = \alpha_i^{m(n-1)}$
$\mathcal{I}_{i,m(n)} = \lambda$

P1 Since $p(\alpha^{n-1}) = i$ and $a \notin acts(A_i)$, case 1 of the definition of f shows that $s\lceil A_i = lstate(\alpha^{n-1})\lceil A_i$. Now,

$$\begin{aligned}\alpha^n\lceil A_i &\stackrel{1}{=} (\alpha^{n-1}as)\lceil A_i\\ &\stackrel{2}{=} \alpha^{n-1}\lceil A_i\\ &\stackrel{3}{=} \alpha_i^{m(n-1)}\\ &\stackrel{4}{=} \alpha_i^{m(n)}\end{aligned}$$

where steps 1 and 4 follow from the definitions made in this case, step 2 follows from the fact that $s\lceil A_i = lstate(\alpha^{n-1})\lceil A_i$ and $s \notin acts(A_i)$, and step 3 follows from the induction hypothesis.

P2 Since $p(\alpha^{n-1}) = i$ and $a \notin acts(A_i)$, case 1 of the definition of f shows that $f_i(\alpha^{n-1}\lceil A_i) = \bot$. Based on the induction hypothesis $f_i(\alpha_i^{m(n-1)}) = f_i(\alpha^{n-1}\lceil A_i)$, so $f_i(\alpha_i^{m(n-1)}) = \bot$. Now case two of the definition of $R_{(g_i,f_i)}$ confirms that $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)})$.

P3 Vacuously satisfied.

Case 1.2 $p(\alpha^{n-1}) = i$ and $a \in in(A_i)$.

Define: $m(n) = m(n-1) + 2$

$\alpha_i^{m(n)-1} = \alpha_i^{m(n-1)}$

$\alpha_i^{m(n)} = \alpha_i^{m(n)-1}as\lceil A_i$

$\mathcal{I}_{i,m(n)-1} = \lambda$

$\mathcal{I}_{i,m(n)} = a$

P1 In this case,

$$\begin{aligned}\alpha^n\lceil A_i &\stackrel{1}{=} (\alpha^{n-1}as)\lceil A_i\\ &\stackrel{2}{=} \alpha^{n-1}\lceil A_i as\lceil A_i\\ &\stackrel{3}{=} \alpha_i^{m(n-1)}as\lceil A_i\\ &\stackrel{4}{=} \alpha_i^{m(n)-1}as\lceil A_i\\ &\stackrel{5}{=} \alpha_i^{m(n)}\end{aligned}$$

where steps 1, 4 and 5 follow from the definitions made in this case, step 2 follows from the fact that $a \in acts(A_i)$, and step 3 follows from the induction hypothesis.

P2 Vacuously satisfied.

P3 Since $p(\alpha^{n-1}) = i$ and $a \in in(A_i)$, case 1 of the definition of f shows that $f_i(\alpha^{n-1}\lceil A_i) = \bot$. Based on the induction hypothesis $f_i(\alpha_i^{m(n-1)}) = f_i(\alpha^{n-1}\lceil A_i)$, so $f_i(\alpha_i^{m(n-1)}) = \bot$. Now case two of the definition of $R_{(g_i,f_i)}$ confirms that $(\alpha_i^{m(n)-1}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)-1})$.
Since $a \in in(A_i)$, case 1 of the definition of f shows that $g_i(\alpha^{n-1}\lceil A_i, a) = s\lceil A_i$. Based on the induction hypothesis $g_i(\alpha_i^{m(n-1)}, a) = g_i(\alpha^{n-1}\lceil A_i, a)$. By definition $g_i(\alpha_i^{m(n)-1}, a) = g_i(\alpha_i^{m(n-1)}, a)$, so $g_i(\alpha_i^{m(n)-1}, a) = s\lceil A_i$. Now case three of the definition of $R_{(g_i,f_i)}$ shows that $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n)-1}, \mathcal{I}_{i,m(n)})$.

Case 1.3 $p(\alpha^n) = i$ and $a \in local(A_i)$ or $p(\alpha^n) \neq i$ and $a \in local(A_i)$.

Define: $m(n) = m(n-1) + 1$
$\alpha_i^{m(n)} = \alpha_i^{m(n-1)} as\lceil A_i$
$\mathcal{I}_{i,m(n)} = \lambda$

P1 In this case,

$$\begin{aligned} \alpha^n\lceil A_i &\stackrel{1}{=} (\alpha^{n-1}as)\lceil A_i \\ &\stackrel{2}{=} \alpha^{n-1}\lceil A_i as\lceil A_i \\ &\stackrel{3}{=} \alpha_i^{m(n-1)} as\lceil A_i \\ &\stackrel{4}{=} \alpha_i^{m(n)} \end{aligned}$$

where steps 1 and 4 follow from the definitions made in this case, step 2 follows from the fact that $a \in acts(A_i)$ and step 3 follows from the induction hypothesis.

P2 Since $a \in local(A_i)$, case 1 of the definition of f shows that $f_i(\alpha^{n-1}\lceil A_i) = (a, s\lceil A_i)$. Based on the induction hypothesis $f_i(\alpha_i^{m(n-1)}) = f_i(\alpha^{n-1}\lceil A_i)$, so $f_i(\alpha_i^{m(n-1)}) = (a, s\lceil A_i)$. Now case one of the definition of $R_{(g_i,f_i)}$ confirms that $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)})$.

P3 Vacuously satisfied.

Case 1.4 $p(\alpha^n) \neq i$ and $a \in in(A_i)$.

Define: $m(n) = m(n-1) + 1$
$\alpha_i^{m(n)} = \alpha_i^{m(n-1)} as\lceil A_i$
$\mathcal{I}_{i,m(n)} = a$

P1 In this case,

$$\begin{aligned} \alpha^n\lceil A_i &\stackrel{1}{=} (\alpha^{n-1}as)\lceil A_i \\ &\stackrel{2}{=} \alpha^{n-1}\lceil A_i as\lceil A_i \\ &\stackrel{3}{=} \alpha_i^{m(n-1)} as\lceil A_i \\ &\stackrel{4}{=} \alpha_i^{m(n)} \end{aligned}$$

where steps 1 and 4 follow from the definitions made in this case, step 2 follows from the fact that $a \in acts(A_i)$ and step 3 follows from the induction hypothesis.

P2 Since $a \in in(A_i)$ case 1 of the definition of f shows that $g_i(\alpha^{n-1}\lceil A_i, a) = s\lceil A_i$. Based on the induction hypothesis $g_i(\alpha_i^{m(n-1)}, a) = g_i(\alpha^{n-1}\lceil A_i, a)$, so $g_i(\alpha_i^{m(n-1)}, a) = s\lceil A_i$. Now case three of the definition $R_{(g_i,f_i)}$ confirms that $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)})$.

P3 Vacuously satisfied.

Case 1.5 $p(\alpha^n) \neq i$ and $a \notin acts(A_i)$.

Define: $m(n) = m(n-1)$

P1 Since $p(\alpha^{n-1}) \neq i$ and $a \notin acts(A_i)$, case 1 of the definition of f shows that $s\lceil A_i = lstate(\alpha^{n-1})\lceil A_i$. Now,

$$\begin{aligned}
\alpha^n\lceil A_i &\overset{1}{=} (\alpha^{n-1}as)\lceil A_i \\
&\overset{2}{=} \alpha^{n-1}\lceil A_i \\
&\overset{3}{=} \alpha_i^{m(n-1)} \\
&\overset{4}{=} \alpha_i^{m(n)}
\end{aligned}$$

where steps 1 and 4 follow from the definitions made in this case, step 2 follows from the fact that $s\lceil A_i = lstate(\alpha^{n-1})\lceil A_i$ and $a \notin acts(A_i)$, and step 3 follows from the induction hypothesis.

P2 Vacuously satisfied.

P3 Vacuously satisfied.

Case 2 $(\alpha^n, \mathcal{I}^n) = (\alpha^{n-1}, tail(\mathcal{I}^{n-1}))$ where $f(\alpha^{n-1}) = \perp$ and $head(\mathcal{I}^{n-1}) = \lambda$.

Define: $m(n) = m(n-1) + 1$
$\alpha_i^{m(n)} = \alpha_i^{m(n-1)}$
$\mathcal{I}_{i,m(n)} = \lambda$

P1 In this case,

$$\begin{aligned}
\alpha^n\lceil A_i &\overset{1}{=} \alpha^{n-1}\lceil A_i \\
&\overset{3}{=} \alpha_i^{m(n-1)} \\
&\overset{4}{=} \alpha_i^{m(n)}
\end{aligned}$$

where steps 1 and 3 follow from the definitions made in this case, step 2 follows from the induction hypothesis.

P2 Since $f(\alpha^{n-1}) = \perp$, case 2 of the definition of f shows that $f_i(\alpha^{n-1}\lceil A_i) = \perp$. Based on the induction hypothesis $f_i(\alpha_i^{m(n-1)}) = f_i(\alpha^{n-1}\lceil A_i)$, so $f_i(\alpha_i^{m(n-1)}) = \perp$. Now case two of the definition of $R_{(g_i,f_i)}$ confirms that $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)})$.

P3 Vacuously satisfied.

Case 3 $(\alpha^n, \mathcal{I}^n) = (\alpha^{n-1}as, tail(\mathcal{I}^{n-1}))$ where $g(\alpha^{n-1}, a) = s$ and $head(\mathcal{I}^{n-1}) = a$.

The definition of g in Definition 3.24 suggests the following sub cases:

Case 3.1 $a \in in(A_i)$.

Define: $m(n) = m(n-1) + 1$
$\alpha_i^{m(n)} = \alpha_i^{m(n-1)}as\lceil A_i$
$\mathcal{I}_{i,m(n)} = a$

P1 In this case,

$$\begin{aligned}\alpha^n\lceil A_i &\stackrel{1}{=} (\alpha^{n-1}as)\lceil A_i\\ &\stackrel{2}{=} \alpha^{n-1}\lceil A_i as\lceil A_i\\ &\stackrel{3}{=} \alpha_i^{m(n-1)}as\lceil A_i\\ &\stackrel{4}{=} \alpha_i^{m(n)}\end{aligned}$$

where steps 1 and 4 follow from the definitions made in this case, step 2 follows from the fact that $a \in acts(A_i)$ and step 3 follows from the induction hypothesis.

P2 The definition of g shows that $g_i(\alpha^{n-1}\lceil A_i, a) = s\lceil A_i$. Based on the induction hypothesis, $g_i(\alpha_i^{m(n-1)}, a) = g_i(\alpha^{n-1}\lceil A_i, a)$, so $g_i(\alpha_i^{m(n-1)}, a) = s\lceil A_i$. Now case three of the definition of $R_{(g_i,f_i)}$ shows that $(\alpha_i^{m(n)}, \varepsilon) = R_{(g_i,f_i)}(\alpha_i^{m(n-1)}, \mathcal{I}_{i,m(n)})$.

P3 Vacuously satisfied.

Case 3.2 $a \notin in(A_i)$.

Define: $m(n) = m(n-1)$

P1 The definition of g shows that $s\lceil A_i = lstate(\alpha^{n-1})\lceil A_i$. Then,

$$\begin{aligned}\alpha^n\lceil A_i &\stackrel{1}{=} (\alpha^{n-1}as)\lceil A_i\\ &\stackrel{2}{=} \alpha^{n-1}\lceil A_i\\ &\stackrel{3}{=} \alpha_i^{m(n-1)}\\ &\stackrel{4}{=} \alpha_i^{m(n)}\end{aligned}$$

where steps 1 and 4 follow from the definitions made in this case, step 2 follows from the fact that $s\lceil A_i = lstate(\alpha^{n-1})\lceil A_i$ and $a \notin acts(A_i)$, and step 3 follows from the induction hypothesis.

P2 Vacuously satisfied.

P3 Vacuously satisfied.

This concludes the inductive definition and induction proof.

The second part of the proof consists of showing that $\mathcal{I}_i$ is indeed an environment sequence for A_i. Denote the generic j^{th} element of $\mathcal{I}_i$ by $\mathcal{I}_{i,j}$. The sequence $\mathcal{I}_i$ is well defined since $\mathcal{I}_{i,m(n)}$ is defined whenever $m(n) = m(n-1)+1$ and $\mathcal{I}_{i,m(n)-1}$ and $\mathcal{I}_{i,m(n)}$ are defined whenever $m(n) = m(n-1)+2$. Showing that $\mathcal{I}_i$ is an environment sequence for A_i induces two proof obligations:

1. $\mathcal{I}_{i,j} \in in(A_i) \cup \{\lambda\}$ for all $j > 0$.

   This follows immediately from the definition of $\mathcal{I}_{i,j}$ in the induction.

2. There are infinitely many $j > 0$ such that $\mathcal{I}_{i,j} = \lambda$.

   Since $\mathcal{I}$ is an environment sequence, it contains infinitely many elements. Thus, the induction has infinitely many steps (i.e., $n \to \infty$). For every step, all cases of the induction except 1.4, 1.5, 3.1, and 3.2 define a new element $\mathcal{I}_{i,j}$ such that $\mathcal{I}_{i,j} = \lambda$. Thus, the proof obligation is met as long as there exists no $n_o \geq 0$ such that for all $n > n_o$ every step of the induction leads to case 1.4, 1.5, 3.1, or 3.2. For a contradiction assume such an n_o exists. Consider the following observations about cases 1.4, 1.5, 3.1 and 3.2. If the n^{th} step leads to cases 3.1 or 3.2, then $l(\alpha^n) = l(\alpha^{n-1})$. If the n^{th} step leads to case 1.4 or 1.5, then $l(\alpha^n) = l(\alpha^{n-1}) + 1$. Furthermore, cases 1.4 and 1.5 require that $p(\alpha^{n-1}) \neq i$, where $p(\alpha^{n-1}) = (l(\alpha^{n-1}) \bmod N) + 1$. Thus, since N is finite, there can be at most finitely many steps after the n_o^{th} step that lead to case 1.4 or 1.5, i.e., as many as necessary to get $p(\alpha^{n-1}) = i$. In other words, there exists a number $n_1 \geq n_0$ such that for each $n \geq n_1$ the n^{th} step leads to case 3.1 or 3.2. However, since $\mathcal{I}$ is an environment sequence, for infinitely many n such that $n > n_o$, $head(\mathcal{I}^{n-1}) = \lambda$. Now there is a contradiction since the n^{th} step cannot lead to case 3.1 or 3.2 when $head(\mathcal{I}^{n-1}) = \lambda$.

An immediate consequence of the fact that $\mathcal{I}_i$ contains infinitely many λ symbols is that $\lim_{n\to\infty} m(n) = \infty$. In fact, $m(n) > m(n-1)$ whenever $\mathcal{I}_{m(n)} = \lambda$.

The final step of the proof consists in showing that $(\alpha_i^n, \mathcal{I}_i^n)_{n\geq 0}$ is the outcome sequence of (g_i, f_i) given $\alpha\lceil A_i$ and $\mathcal{I}_i$, and thus that $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})\lceil A_i = \mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i, \mathcal{I}_i)$. Let $\mathcal{I}_i^n$ denote the suffix of $\mathcal{I}_i$ generated by removing the first n elements of $\mathcal{I}_i$. By definition, $(\alpha_i^0, \mathcal{I}_i^0) = (\alpha\lceil A_i, \mathcal{I}_i)$. Thus it must be verified that for all $n > 0$, $(\alpha_i^n, \mathcal{I}_i^n) = R_{(g_i,f_i)}(\alpha_i^{n-1}, \mathcal{I}_i^{n-1})$. This fact follows directly from P2 and P3 and the following observation: "For any strategy (g', f') defined on any safe I/O automaton A', any pair of executions $\alpha', \alpha'' \in exec^*(A')$, and any environment sequence $\mathcal{I}'$, $(\alpha', tail(\mathcal{I}')) = R_{(g',f')}(\alpha'', \mathcal{I}')$ iff $(\alpha', \varepsilon) = R_{(g',f')}(\alpha'', head(\mathcal{I}'))$." Since $(\alpha_i^n, \mathcal{I}_i^n)_{n\geq 0}$ is an outcome sequence of (g_i, f_i) given $\alpha\lceil A_i$ and $\mathcal{I}_i$, the definition of an outcome shows that $\mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i, \mathcal{I}_i) = \lim_{n\to\infty} \alpha_i^n$. Thus,

$$\begin{aligned}
\mathcal{O}_{(g,f)}(\alpha, \mathcal{I})\lceil A_i &\stackrel{1}{=} (\lim_{n\to\infty} \alpha^n)\lceil A_i \\
&\stackrel{2}{=} \lim_{n\to\infty}(\alpha^n\lceil A_i) \\
&\stackrel{3}{=} \lim_{n\to\infty}(\alpha_i^{m(n)}) \\
&\stackrel{4}{=} \lim_{n\to\infty}(\alpha_i^n) \\
&\stackrel{5}{=} \mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i, \mathcal{I}_i)
\end{aligned}$$

where step 1 follows from the definition of α^n, step 2 follows from the continuity of the projection operator, step 3 follows from P1, step 4 follows from the fact that $\lim_{n\to\infty} m(n) = \infty$ and the family $(\alpha_i^n)_{n\geq 0}$ form a chain ordered under prefix, and step 5 from the fact that $\mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i, \mathcal{I}_i) = \lim_{n\to\infty} \alpha_i^n$. ■

Lemma 3.27

Let $(A_1, L_1) \ldots, (A_N, L_N)$ be compatible live I/O automata and, for each $1 \leq i \leq N$, let (g_i, f_i) be an environment-free strategy for (A_i, L_i). Then $(g_1, f_1) \parallel \cdots \parallel (g_N, f_N)$ is an environment-free strategy for $(A_1, L_1) \parallel \cdots \parallel (A_N, L_N)$.

Proof. Let $(A, L) = (A_1, L_1) \parallel \cdots \parallel (A_N, L_N)$ and $(g, f) = (g_1, f_1) \parallel \cdots \parallel (g_N, f_N)$. From Definition 3.4, we know that A is a safe I/O automaton. Furthermore, from Definition 3.19, Lemma 3.5, and the fact that each $L_i \subseteq exec(A_i)$, the set L is a subset of $exec(A)$.

Consider any environment sequence $\mathcal{I}$ for A and any finite execution α of A. By Lemma 3.26 there exists for all A_i an environment sequence $\mathcal{I}_i$ such that $\mathcal{O}_{(g,f)}(\alpha,\mathcal{I})\lceil A_i = \mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i,\mathcal{I}_i)$. Since (g_i,f_i) is an environment-free strategy for (A_i,L_i), $\mathcal{O}_{(g_i,f_i)}(\alpha\lceil A_i,\mathcal{I}_i) \in L_i$ Consequently, $\mathcal{O}_{(g,f)}(\alpha,\mathcal{I})\lceil A_i \in L_i$ for all (A_i,L_i). From Definition 3.19, $\mathcal{O}_{(g,f)}(\alpha,\mathcal{I}) \in L$. Thus (g,f) is an environment-free strategy for (A,L). ∎

Proposition 3.28 (Closure of parallel composition)

Let $(A_1,L_1),\ldots,(A_N,L_N)$ be compatible live I/O automata. Then $(A_1,L_1) \parallel \cdots \parallel (A_N,L_N)$ is a live I/O automaton.

Proof. Let $(A,L) = (A_1,L_1) \parallel \cdots \parallel (A_N,L_N)$. From Definition 3.4, we know that A is a safe I/O automaton. Furthermore, from Definition 3.19, Lemma 3.5, and the fact that each $L_i \subseteq exec(A_i)$, the set L is a subset of $exec(A)$.

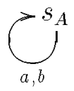
For each $1 \leq i \leq N$, let (g_i,f_i) be an environment-free strategy for (A_i,L_i). By Lemma 3.27 the strategy $(g,f) = (g_1,f_1) \parallel \cdots \parallel (g_N,f_N)$ is an environment-free strategy for (A,L). Therefore, the pair (A,L) is environment-free. Thus, from Definition 3.16, (A,L) is a live I/O automaton. ∎

Environment-freedom is a crucial property of live I/O automata since it guarantees that no pair of compatible live I/O automata constrain each other's environments. In particular, if pair (A,L) is not environment-free, the parallel composition operator may generate pairs that are not even live automata.

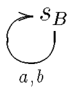
Example 3.29

Consider safe I/O automata A and B described by the state transition diagrams below.

A :



B :



For A, action b is an input action, and action a is an output action; for B, action a is an input action and action b is an output action. Let the liveness condition L_A for A be the set of executions α of A such that $trace(\alpha)$ ends in $(ab)^\infty$ or a^∞, and let the liveness condition L_B for B be the set of executions α of B such that $trace(\alpha)$ ends in $(aabb)^\infty$ or b^∞.

The pairs (A,L_A) and (B,L_B) are not environment-free. To see that (A,L_A) is not environment-free consider the environment sequence $\mathcal{I} = bb\lambda bb\lambda\cdots$; to see that (B,L_B) is not environment-free consider the environment sequence $\mathcal{I} = aaa\lambda aaa\lambda\cdots$.

Let $(C,L_C) = (A,L_A) \parallel (B,L_B)$. In this case, $L_C = \emptyset$. Thus L_C is not a liveness condition for C, which means that (C,L_C) is not even a live automaton. ∎

Example 3.29 also exposes the flaw in a simpler and more intuitive definition for environment-freedom we originally considered for this paper. The simpler definition, which is a natural generalization of the fairness condition of [LT87] and is also discussed in [LS89], states that "a pair (A, L) is environment-free if for each finite execution α of A and each (finite or infinite) sequence β of input actions there is an execution fragment α' of A such that $\alpha' \lceil in(A) = \beta$ and $\alpha \frown \alpha' \in L$." It is easy to see that the pairs (A, L_A) and (B, L_B) of Example 3.29 are both environment-free based on the simpler definition. However, the example shows that their composition cannot be a live I/O automaton. The problem with the simpler definition is that it allows the system to choose its relative speed with respect to the environment, and it allows the system to base its decisions on the future behavior of the environment. Example 3.29 shows that the simpler definition thus gives the system too much power for parallel composition to be closed.

3.5 Preorder Relations for Live I/O Automata

In [LT87, Dil88, AL93] the notion of implementation is expressed through some form of trace inclusion. Similar notions of implementation can be defined on live I/O automata. In particular it is possible to identify two preorder relations, the safe and the live preorders, which aim at capturing the safety and liveness aspects of live I/O automata, respectively.

Definition 3.30 (Trace preorders)

Given two live I/O automata (A_1, L_1) and (A_2, L_2) such that $esig(A_1) = esig(A_2)$, define the following preorders:

Safe: $(A_1, L_1) \sqsubseteq_{\mathrm{S}} (A_2, L_2)$ iff $traces(A_1) \subseteq traces(A_2)$
Live: $(A_1, L_1) \sqsubseteq_{\mathrm{L}} (A_2, L_2)$ iff $traces(L_1) \subseteq traces(L_2)$ ■

The safe preorder is the same as the unfair preorder of I/O automata [LT87], while the live preorder is a generalization of the fair preorder of [LT87]. In particular, the live preorder coincides with the fair preorder if, for each live I/O automaton (A, L), L is chosen to be the set of fair executions of A. The conformation preorder of [Dil88], which expresses the notion of implementation for complete trace structures, coincides with the live preorder when dealing with failure free complete trace structures. Finally, the notion of implementation of [AL93], which works in a state based model, coincides with the live preorder up to a different notion of traces arising from the state structure of the model. In [AL93], a system M_1 implements a system M_2 iff the set of "traces" of the realizable part of M_1 is a subset of the set of "traces" of the realizable part of M_2. Furthermore, if a system M is receptive, then M is equal to its realizable part. Thus, for receptive systems, the implementation notion of [AL93] is just the live trace preorder. The reader is referred to Section 3.6 for more details about realizability.

It is interesting to note that the live preorder implies the safe preorder whenever the involved automata have finite internal nondeterminism. On the other hand, if the involved automata do not have finite internal nondeterminism, the live preorder only implies finite trace inclusion.

Essentially, finite internal nondeterminism requires that a live I/O automaton has a finite internal branching structure. In particular, an external action can lead to only a finite number of states, and a state may enable at most a finite number of internal actions.

Definition 3.31 (Finite internal nondeterminism)

An automaton A has *finite internal nondeterminism* (FIN) iff, for each finite trace $\beta \in traces^*(A)$, the set $\{lstate(\alpha) \mid \alpha \in exec^*(A), trace(\alpha) = \beta\}$ is finite. ■

Proposition 3.32

Let (A_1, L_1) and (A_2, L_2) be two live I/O automata with $esig(A_1) = esig(A_2)$.

1. *If $(A_1, L_1) \sqsubseteq_{\mathrm{L}} (A_2, L_2)$ then $traces^*(A_1) \subseteq traces^*(A_2)$*
2. *If A_2 has FIN and $(A_1, L_1) \sqsubseteq_{\mathrm{L}} (A_2, L_2)$, then $(A_1, L_1) \sqsubseteq_{\mathrm{S}} (A_2, L_2)$*

Proof.

1. Let β be a finite trace of A_1. By definition of trace, there is an execution α_1 of A_1 such that $trace(\alpha_1) = \beta$. By definition of a live I/O automaton there exists an execution α_1' of A_1 such that $\alpha_1 \leq \alpha_1'$ and $\alpha_1' \in L_1$. Since $(A_1, L_1) \sqsubseteq_{\mathrm{L}} (A_2, L_2)$, there exists an execution α_2' of L_2 such that $trace(\alpha_1') = trace(\alpha_2')$. By definition of a live I/O automaton, α_2' is an execution of A_2, and, since the set of executions of an automaton is closed under prefix, there is a prefix α_2 of α_2' such that α_2 is an execution of A_2 and $trace(\alpha_2) = \beta$, i.e., β is a trace of A_2.
2. Finite trace inclusion follows directly from part 1. Infinite trace inclusion follows from finite trace inclusion, closure under prefix of trace sets, and the fact that trace sets of automata with finite internal nondeterminism are closed under prefix ordering limit [LV91]. ■

The proof of Proposition 3.32 supports the requirement of our definition of a liveness condition (Definition 3.2) that every safe execution be extendible to a live execution. Without this requirement, the live preorder could not be used to infer the safe preorder, i.e., neither part of Proposition 3.32 would hold.

An important goal of this paper is the substitutivity of the safe and live preorders for the operators of Section 3.4. In the case of the parallel composition operator, this means that an implementation of a system made up of several parallel components can be obtained by implementing each component separately.

Theorem 3.33 (Substitutivity)

Let $(A_i, L_i), (A_i', L_i')$, $i = 1, \ldots, N$ be live I/O automata, and let $\sqsubseteq_X$ be either $\sqsubseteq_{\mathrm{S}}$ or $\sqsubseteq_{\mathrm{L}}$. If, for each i, $(A_i, L_i) \sqsubseteq_X (A_i', L_i')$, then

1. *if* $(A_1, L_1), \ldots, (A_N, L_N)$ *are compatible and* $(A'_1, L'_1), \ldots, (A'_N, L'_N)$ *are compatible then* $(A_1, L_1)\|\cdots\|(A_N, L_N) \sqsubseteq_X (A'_1, L'_1)\|\cdots\|(A'_N, L'_N)$

2. *if* $\Lambda \subseteq local(A_1)$ *and* $\Lambda \subseteq local(A'_1)$ *then* $(A_1, L_1) \setminus \Lambda \sqsubseteq_X (A'_1, L'_1) \setminus \Lambda$

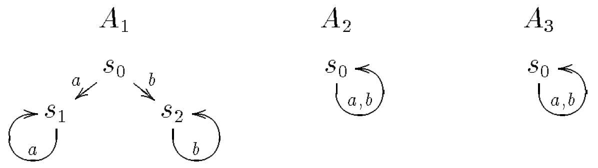
3. *if* ρ *is a mapping applicable to both* A_1 *and* A'_1 *then* $\rho((A_1, L_1)) \sqsubseteq_X \rho((A'_1, L'_1))$

Proof. The substitutivity results for the safe trace preorder are already proven in [LT87]. The substitutivity results for the live trace preorder follow directly from the definitions of the parallel composition, hiding, and renaming operators after observing, as it is proved in Corollaries 8, 13 and Lemma 15 of [LT87], that parallel composition, hiding and renaming of execution sets preserve trace equivalence. ∎

The following example shows that the absence of environment-freedom can lead to situations where the substitutivity result of Theorem 3.33 breaks down.

Example 3.34

Consider the safe I/O automata A_1, A_2, and A_3 with the transition diagrams below.



where a and b are output actions for A_1 and A_2 and are input actions for A_3. Let L_1 (resp. L_2) be the set of executions of A_1 (resp. A_2) containing at least one action and let L_3 be the set of executions of A_3 containing at least one occurrence of action a immediately followed by an occurrence of action b. It is easy to check that (A_1, L_1) and (A_2, L_2) are both environment-free, but (A_3, L_3) is not environment-free since it requires at least one input.

Observe that $(A_1, L_1) \sqsubseteq_{\mathrm{L}} (A_2, L_2)$ and that $(A_2, L_2)\|(A_3, L_3)$ is environment-free and thus a live I/O automaton. One might want to conclude that $(A_1, L_1)\|(A_3, L_3) \sqsubseteq_{\mathrm{L}} (A_2, L_2)\|(A_3, L_3)$. Unfortunately, this conclusion is false. In particular, let $(A, L) = (A_1, L_1)\|(A_3, L_3)$. Then, the set L is not a liveness condition since A_1 can never perform an action a followed by an action b. Thus, the fact that (A_3, L_3) is not environment-free causes situations where the parallel composition with (A_3, L_3) fails to lead to a pair (A, L) where L is a liveness condition. This in turn causes the substitutivity of the parallel composition operator to fail. ∎

There are several ways in which the live preorder can be justified as an adequate notion of implementation for live I/O automata. Since the live preorder captures the implementation notions of [LT87, Dil88, AL93] it can rest on the justifications provided for these implementation notions. For example, the fair preorder of [LT87] is justified by two observations. First, the fact that I/O automata are input-enabled guarantees that a system must respond to any environment. In our model the same property is guaranteed by the concept of environment-freedom. Second, by restricting attention to fair traces the correctness of an implementation is based only on executions where the system behaves fairly. In our model this property is guaranteed by restricting attention to live traces.

An additional justification for the live preorder as a notion of implementation is based on the concepts of safety and liveness properties. It is easy to see that the safe preorder preserves the safety properties of a system, i.e., the safe preorder guarantees that an implementation cannot do anything that is not allowed by the specification. The live preorder, on the other hand, preserves the liveness properties of a system, thus guaranteeing that an implementation must do something whenever it is required to by the specification. Informally, if after a sequence of actions β something has to happen, β is not a live trace of the specification, and thus not a live trace of the implementation. Therefore, even in the implementation something has to happen after β has occurred. If the involved systems have finite internal nondeterminism, then the live preorder implies the safe preorder. Thus the live preorder guarantees both safety and liveness properties.

3.6 Comparison with Other Models

This section compares our model with the models of [Dil88, LT87, AL93] and the work of [RWZ92].

The model of complete trace structures of [Dil88] is a special case of our model. Specifically, the model of [Dil88] does not include a state structure, so that the safe part of a live automaton in [Dil88] is given by a set of traces. Since there is no notion of a state in a complete trace structure, a strategy for a system is simpler than our strategies in the sense that function g is not necessary and that function f simply picks up a locally-controlled action based on previous environment moves. By ignoring the state structure of a system, the model in [Dil88] may erroneously view as receptive a state machine that is not environment-free based on our model since its traces may be receptive. Thus, complete trace structures are not adequate whenever the state structure of a system is important.

The I/O automaton model of [LT87] is also a special case of our model. An I/O automaton M of [LT87] can be represented in our model as the environment-free pair (A, L), where A is the I/O automaton M without the partition of its locally-controlled actions and L is the set of fair executions of M. The environment-free strategy (g, f) for (A, L) is defined in such a way that g picks up any possible next state in response to an input action, while f gives fair turns to proceed (say in a round robin way) to all the components of M that are continuously willing to perform some locally-controlled action. Thus [LT87] can only express some special cases of our general liveness conditions.

The model of [AL93] is based on unlabeled state transition systems and is suitable for the modeling of shared memory systems. An action in [AL93] is identified with a set of transitions, and transitions are partitioned into environment transitions and system transitions. The environment moves by performing an arbitrary finite number of environment transitions and the system responds by performing zero or one system transitions. Function g is not necessary in a strategy for a system of [AL93] since the environment chooses the next shared state in its move and does not modify the internal state. Function f chooses a new transition based on the past history of the system.

A fundamental difference between [AL93] and our work is in that we define environment-freedom by requiring the existence of a strategy that can "win the game" after any finite execution α, whereas [AL93] considers a weaker property, called *realizability*, where the requirement is the existence of a strategy that can win starting from any start state (cf. Example 3.17). The realizable part of a system of [AL93] is the set of behaviors that can be the outcome of some strategy. A system is then *receptive* if it coincides with its realizable part. The notion of receptiveness of [AL93] corresponds to our notion of environment-freedom, as can be derived easily from Proposition 9 of [AL93].

Example 3.17 shows a live automaton (A, L), which is not environment-free. However, (A, L) is realizable, and (A', L'), which is defined in the same example, is the realizable part of (A, L). In [AL93] systems are compared based on their realizable parts. Thus, it is necessary to determine the realizable part of a system *before* its safety properties can be determined, and for this reason realizable systems are closed under parallel composition in [AL93]. In other words, L can add new safety properties to A. However, later in [AL93] a notion of *machine-realizability* is introduced which separates safety and liveness properties and requires receptiveness, or equivalently environment-freedom, just like our live I/O automata.

Finally, it is easy to show, given our definition of environment-freedom, that the set of live traces of any live I/O automaton is union-game realizable according to [RWZ92], and thus describable by means of a standard I/O automaton of [LT87]. However, in general the I/O automaton description would involve a lot of encoding and would be extremely unnatural.

4 Timed Systems

The notion of liveness discussed in the previous section is now extended to the timed model. Section 4.1 introduces *timed automata* along with *timed executions* and *timed traces*, and shows the relationship between the new timed executions and the ordinary executions from the untimed model. Section 4.2 introduces *live timed automata*. Section 4.3 defines *safe timed I/O automata* by introducing the Input/Output distinction. Section 4.4 extends the notion of *environment-freedom* to the timed model and defines *live timed I/O automata*. Section 4.5 introduces several preorders on live timed I/O automata, one of which is used to express a notion of implementation. Finally, Section 4.6 compares our model with existing work. Since Examples 3.10, 3.17, 3.29, and 3.34 apply equally to the timed model, our discussion focuses on issues specific to the timed model.

4.1 Timed Automata

The following definition of a timed automaton is the same as the corresponding definition in [LV93b] except for the fact that our definition allows multiple internal actions. Also, the notions of timed executions and timed traces are the same as the definitions of [LV93b]. The definitions are repeated here but the reader is referred to [LV93b] for further details. Times are specified using a *dense* time domain T. In this work, as in [LV93b], let T be $\mathbb{R}^{\geq 0}$, the set of non-negative reals.

Definition 4.1 (Timed automaton)

A *timed automaton* A is an automaton whose set of external actions contains a special *time-passage* action ν. Define the set of *visible* actions to be $vis(A) \triangleq ext(A) \setminus \{\nu\}$.

As an additional component, a timed automaton contains a mapping $now_A : states(A) \to \mathsf{T}$ (called *now* when A is clear from context), indicating the current time in a given state.

Finally, A must satisfy the following five axioms

S1 If $s \in start(A)$ then $s.now = 0$.

S2 If $(s, a, s') \in steps(A)$ and $a \neq \nu$, then $s'.now = s.now$.

S3 If $(s, \nu, s') \in steps(A)$ then $s'.now > s.now$.

S4 If $(s, \nu, s') \in steps(A)$ and $(s', \nu, s'') \in steps(A)$, then $(s, \nu, s'') \in steps(A)$.

To be able to state the last axiom, the following auxiliary definition is needed. Let I be an interval of T. Then a function $\omega : I \to states(A)$ is an A*-trajectory*, sometimes called *trajectory* when A is clear from context, if

1. $\omega(t).now = t$ for all $t \in I$, and
2. $(\omega(t), \nu, \omega(t')) \in steps(A)$ for all $t, t' \in I$ with $t < t'$.

That is, ω assigns to each time t in the interval I a state having the given time t as its *now* component. The assignment is done in such a way that time-passage steps can span between any pair of states in the range of ω. Denote $inf(I)$ and $sup(I)$ by $ftime(\omega)$ and $ltime(\omega)$, respectively. If I is left closed, then denote $\omega(ftime(\omega))$ by $fstate(\omega)$. Similarly, if I is right closed, then denote $\omega(ltime(\omega))$ by $lstate(\omega)$. If I is closed, then ω is said to be an A-trajectory from $fstate(\omega)$ to $lstate(\omega)$. An A-trajectory ω whose domain $dom(\omega)$ is a singleton set $[t, t]$ is also denoted by the set $\{\omega(t)\}$. The range of ω is denoted by $rng(\omega)$.

The final axiom then becomes

S5 If $(s, \nu, s') \in steps(A)$ then there exists an A-trajectory from s to s'. ■

Axiom **S1** states that time must be 0 in any start state. Axiom **S2** says that non-time-passage steps occur instantaneously. In this framework, operations with some duration in time are modeled by a start action and an end action. Axiom **S3** says that time-passage steps cause time to increase. Axiom **S4** gives a natural property of time, namely that if time can pass in two steps, then it can also pass in a single step. Finally, Axiom **S5** says that if time can pass from time t to time t', then it is possible to associate states with all times in the interval $[t, t']$ in a consistent way. In [LV93b] the last axiom is explained further and compared to the weaker axiom that says the following: if time can pass in one step, then it can pass in two steps with the time of the intermediate state being any time in the interval.

Timed Executions

Section 3 introduced the notions of execution and trace for automata. These notions carry over to timed automata with the addition of one new idea.

In particular, the notion of execution only allows one to associate states with a countable number of points in time, whereas the trajectory axiom **S5** allows one to associate states with all real times. Also, the intuition about the execution of a timed system is that visible actions occur at points in time, and that time passes "continuously" between these points. These observations lead to the definition of a *timed execution*. The definition is close to the notion of *hybrid computation* of [MMP91] where continuous changes and discrete events alternate during the execution of a system.

A *timed execution fragment* Σ of a timed automaton A is a (finite or infinite) sequence of alternating A-trajectories and actions in $vis(A) \cup int(A)$, starting in a trajectory and, if the sequence is finite, ending in a trajectory

$$\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$$

such that the following holds for each index i:

1. If ω_i is not the last trajectory in Σ, then its domain is a closed interval. If ω_i is the last trajectory of Σ (when Σ is a finite sequence), then its domain is a left-closed interval (and either open or closed to the right).

2. If ω_i is not the last trajectory of Σ, then $(\mathit{lstate}(\omega_i), a_{i+1}, \mathit{fstate}(\omega_{i+1})) \in \mathit{steps}(A)$.

A *timed execution* is a timed execution fragment $\omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ for which $\mathit{fstate}(\omega_0)$ is a start state.

If Σ is a timed execution fragment, then define $\mathit{ftime}(\Sigma)$ and $\mathit{fstate}(\Sigma)$ to be $\mathit{ftime}(\omega_0)$ and $\mathit{fstate}(\omega_0)$, respectively, where ω_0 is the first trajectory of Σ. Also, define $\mathit{ltime}(\Sigma)$ to be the supremum of the union of the domains of the trajectories of Σ, i.e. the supremum of the *now* values of all the states in the ranges of the trajectories of Σ. Finally, if Σ is a finite sequence where the domain of the last trajectory ω is a closed interval, define $\mathit{lstate}(\Sigma)$ to be $\mathit{lstate}(\omega)$.

Finite, Admissible, and Zeno Timed Executions

The timed executions and timed execution fragments of a timed automaton can be partitioned into *finite*, *admissible*, and *Zeno* timed executions and timed execution fragments.

A timed execution (fragment) Σ is defined to be *finite*, if it is a finite sequence and the domain of the last trajectory is a closed interval. A timed execution (fragment) Σ is *admissible* if $ltime(\Sigma) = \infty$. Finally, a timed execution (fragment) Σ is *Zeno* if it is neither finite nor admissible.

There are basically two types of Zeno timed executions: those containing infinitely many occurrences of non-time-passage actions but for which there is a finite upper bound on the times in the domains of the trajectories, and those containing finitely many occurrences of non-time-passage actions and for which the domain of the last trajectory is right-open. Thus, Zeno timed executions represent executions of a timed automaton where an infinite amount of activity occurs in a bounded period of time. (For the second type of Zeno timed executions, the infinitely many time-passage steps needed to span the right-open interval should be thought of as an "infinite amount of activity".)

There are idealized processes that naturally exhibit Zeno behaviors. As an example consider a ball that bounces on the floor and loses a fraction of its energy at each bounce. Ideally the ball will bounce infinitely many times within a finite amount of time. Note, however, that our timed automaton model cannot suitably describe this process since there is no way of specifying what happens after the ball stops bouncing. Fortunately, Zeno behaviors do not occur in the systems we are interested in describing.

From now on, the focus will be on admissible timed executions since these executions correspond to the intuition that time is a force beyond control that happens to approach infinity. However, according to the definition of timed automata, it is possible to specify timed automata for which from some states no admissible timed executions fragments are possible. In particular, such a state may only allow a Zeno timed execution, or it may prevent time from advancing at all (in which case a *time deadlock* has occurred).

Denote by $t\text{-}frag^*(A)$, $t\text{-}frag^\infty(A)$, $t\text{-}frag^Z(A)$, and $t\text{-}frag(A)$ the sets of finite, admissible, Zeno, and all timed execution fragments of A. Similarly, denote by $t\text{-}exec^*(A)$, $t\text{-}exec^\infty(A)$, $t\text{-}exec^Z(A)$, and $t\text{-}exec(A)$ the sets of finite, admissible, Zeno, and all timed executions of A.

A finite timed execution fragment $\Sigma_1 = \omega_0 a_1 \omega_1 \cdots a_n \omega_n$ of A and a timed execution fragment $\Sigma_2 = \omega'_n a_{n+1} \omega_{n+1} a_{n+2} \omega_{n+2} \cdots$ of A can be *concatenated* if $lstate(\Sigma_1) = fstate(\Sigma_2)$. The concatenation, written $\Sigma_1 \frown \Sigma_2$, is defined to be $\Sigma = \omega_0 a_1 \omega_1 \cdots a_n (\omega_n \frown \omega'_n) a_{n+1} \omega_{n+1} a_{n+2} \omega_{n+2} \cdots$, where $\omega \frown \omega'(t)$, for any functions ω and ω' from intervals of time to $states(A)$, is defined to be $\omega(t)$ if t is in $dom(\omega)$, and $\omega'(t)$ if t is in $dom(\omega') \setminus dom(\omega)$. It is easy to see that Σ is a timed execution fragment of A.

The notion of timed prefix, called *t-prefix*, for timed execution fragments is defined as follows. A timed execution fragment Σ_1 of A is a *t-prefix* of a timed execution fragment Σ_2 of A, written $\Sigma_1 \leq_t \Sigma_2$, if either $\Sigma_1 = \Sigma_2$ or Σ_1 is finite and there exists a timed execution fragment Σ'_1 of A such that $\Sigma_2 = \Sigma_1 \frown \Sigma'_1$. Likewise, Σ_1 is a *t-suffix* of Σ_2 if there exists a finite timed execution fragment Σ'_1 such that $\Sigma_2 = \Sigma'_1 \frown \Sigma_1$.

For a finite timed execution fragment Σ_1 and a timed execution fragment Σ_2 with $\Sigma_1 \leq_t \Sigma_2$, define $\Sigma_2 - \Sigma_1$ to be the (unique) timed execution fragment Σ'_1 such that $\Sigma_2 = \Sigma_1 \frown \Sigma'_1$.

The *length* of a timed execution fragment Σ expresses the number of *visible* and *internal* actions in Σ. Thus, even though Σ is admissible or Zeno (and thus not finite), its length might be finite. Formally, define the length of $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ as

$$|\Sigma| \triangleq \begin{cases} n & \text{if } \Sigma \text{ is a finite sequence and ends in } \omega_n \\ \infty & \text{if } \Sigma \text{ is an infinite sequence} \end{cases}$$

The definition of ith *prefix* of $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$, for all $0 \leq i \leq |\Sigma|$, is

$$\Sigma|_i \triangleq \omega_0 a_1 \omega_1 \cdots a_i \omega_i$$

Define $\Sigma \lhd t$, read "Σ before t", for all $t \geq \mathit{ftime}(\Sigma)$, to be the t-prefix of Σ that includes exactly all states with times not bigger than t. Formally,

$$\Sigma \lhd t \triangleq \begin{cases} \Sigma & \text{if } \; t \geq \mathit{ltime}(\Sigma) \\ \Sigma' & \text{if } \; t < \mathit{ltime}(\Sigma) \text{ and there exists } \Sigma'' = \omega''_0 a''_1 \omega''_1 \cdots \text{ such that} \\ & \qquad \Sigma = \Sigma' \frown \Sigma'' \text{ and } \mathit{ltime}(\Sigma) = t \text{ and } |\mathit{dom}(\omega''_0)| > 1 \end{cases}$$

Likewise, define $\Sigma \rhd t$, read "Σ after t", for all $t < \mathit{ltime}(\Sigma)$ or all $t \leq \mathit{ltime}(\Sigma)$ when Σ is finite, to be the t-suffix of Σ that includes exactly all states with times not smaller than t. Formally,

$$\Sigma \rhd t \triangleq \begin{cases} \Sigma & \text{if } \; t \leq \mathit{ftime}(\Sigma) \\ \Sigma' & \text{if } \; t > \mathit{ftime}(\Sigma) \text{ and there exists } \Sigma'' = \omega''_0 a''_1 \omega''_1 \cdots \omega''_n \text{ such that} \\ & \qquad \Sigma = \Sigma'' \frown \Sigma' \text{ and } \mathit{ftime}(\Sigma') = t \text{ and } |\mathit{dom}(\omega''_n)| > 1 \end{cases}$$

Timed Traces

In the untimed model automata are compared based on their traces. This turns out to be inadequate in the timed model as the following example illustrates. The example is a slight modification of an example in [LV91].

Example 4.2

Let *Idle* be a timed automaton that lets time pass except that it performs a visible action a at time 50. More specifically, let the state set be $\mathsf{T} \times \{\mathit{true}, \mathit{false}\}$ with the initial state $(0, \mathit{true})$, and let the steps be

$$((t, b), \nu, (t', b)) \quad \text{if} \quad t < t' \wedge (b = \mathit{true} \Longrightarrow t' \leq 50), \text{ and}$$
$$((50, \mathit{true}), a, (50, \mathit{false})).$$

Then let *idle'* be the timed automaton that performs a at time 50 but also performs an internal action τ at time 37. Thus, the state space is $\mathsf{T} \times \{\mathit{true}, \mathit{false}\} \times \{\mathit{true}, \mathit{false}\}$, initially $(0, \mathit{true}, \mathit{true})$, and let the steps be

$$((t, b_1, b_2), \nu, (t', b_1, b_2)) \quad \text{if} \quad t < t' \wedge (b_1 = \mathit{true} \Longrightarrow t' \leq 37) \wedge (b_2 = \mathit{true} \Longrightarrow t' \leq 50),$$
$$((37, \mathit{true}, \mathit{true}), \tau, (37, \mathit{false}, \mathit{true})), \text{ and}$$
$$((50, \mathit{false}, \mathit{true}), a, (50, \mathit{false}, \mathit{false})).$$

Then *Idle* and *Idle'* do not have the same traces. In particular, *Idle* has a trace $\nu\, a$ that is not a trace for *Idle'*. (*Idle'* has a trace $\nu\, \nu\, a$ since it cannot let time pass to 50 in one step.)

This clearly contradicts the intuition about timed automata. Seen from "the outside" they both wait until time 50 and then perform a. The example explains why traces are not a good basis for comparing timed automata.

Note, that making ν internal would not solve the problem. In that case, a timed automaton that performs a at time 10 would have the same traces (namely a) as a timed automaton that performs a at time 50. ■

The problem in Example 4.2 arises because of the invisible nature of time-passage actions. This leads to *timed traces*, which consist of the visible actions together with their time of occurrence.

Timed Sequence Pairs

A *timed sequence* over a set K is defined to be a (finite or infinite) sequence δ over $K \times \mathsf{T}$ in which the second components of every pair (the *time* components) are nondecreasing. Define δ to be *Zeno* if it is infinite and the limit of the time components is finite. For any nonempty timed sequence δ, define $\mathit{ftime}(\delta)$ to be the time component of the first pair in δ.

As for timed execution fragments, the operators $\lhd$ and $\rhd$ are defined on timed sequences. Define $\delta \lhd t$, for all $t \in \mathsf{T}$, to be the longest prefix δ' of δ such that all time components of δ' are less than or equal to t. Similarly, define $\delta \rhd t$, for all $t \in \mathsf{T}$, to be the longest suffix[1] δ' such that all time components of δ' are greater than or equal to t.

A *timed sequence pair* over K is a pair $\gamma = (\delta, t)$, where δ is a timed sequence over K and $t \in \mathsf{T} \cup \{\infty\}$, such that t is greater than or equal to all time components in δ. Let $\mathit{seq}(\gamma)$ and $\mathit{ltime}(\gamma)$ denote the two respective components of γ. Then define $\mathit{ftime}(\gamma)$ to be equal $\mathit{ftime}(\mathit{seq}(\gamma))$ in case $\mathit{seq}(\gamma)$ is nonempty, and equal to $\mathit{ltime}(\gamma)$ otherwise. Denote by $\mathit{tsp}(K)$ the set of timed sequence pairs over K. A timed sequence pair γ is said to be *finite* if both $\mathit{seq}(\gamma)$ and $\mathit{ltime}(\gamma)$ are finite, and *admissible* if $\mathit{seq}(\gamma)$ is not Zeno and $\mathit{ltime}(\gamma) = \infty$.

Timed Traces of Timed Automata

Let $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ be a timed execution fragment of a timed automaton A. For each a_i, define the *time of occurrence* t_i to be $\mathit{ltime}(\omega_{i-1})$, or equivalently, $\mathit{ftime}(\omega_i)$. Then, define $t\text{-}\mathit{seq}(\Sigma)$ to be the sequence consisting of the actions in Σ paired with their time of occurrence:

$$t\text{-}\mathit{seq}(\Sigma) = (a_1, t_1)(a_2, t_2) \cdots$$

Then $t\text{-}\mathit{trace}(\Sigma)$, the *timed trace* of Σ, is defined to be the timed sequence pair over $\mathit{vis}(A)$

$$t\text{-}\mathit{trace}(\Sigma) \triangleq (t\text{-}\mathit{seq}(\Sigma) \upharpoonright (\mathit{vis}(A) \times \mathsf{T}), \mathit{ltime}(\Sigma)).$$

Thus, $t\text{-}\mathit{trace}(\Sigma)$ records the occurrences of *visible* actions together with their time of occurrence, and the limit time of the timed execution fragment. A timed trace suppresses both internal and time-passage actions.

[1] Strictly speaking, the suffix obtained by removing the shortest prefix.

Let $t\text{-}traces^*(A)$, $t\text{-}traces^\infty(A)$, $t\text{-}traces^Z(A)$, and $t\text{-}traces(A)$ denote the sets of timed traces of A obtained from finite, admissible, Zeno, and all timed executions of A, respectively.

Relationships Between Timed and Untimed Execution Fragments

There is a close relationship between timed execution fragments and ordinary execution fragments of a timed automaton. This leads to an alternative, but equivalent, definition of timed traces. All definitions and lemmas are taken from [LV93b].

Sampling

Roughly speaking, an (ordinary) execution fragment can be regarded as "sampling" the state information in a timed execution fragment at a countable number of points in time. Formally, we say that an execution fragment $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ of A *samples* a timed execution fragment $\Sigma = \omega_0 b_1 \omega_1 b_2 \omega_2 \cdots$ of A if there is a monotone increasing function $f : \mathbb{N}_0 \to \mathbb{N}_0$ such that the following conditions are satisfied.

1. $f(0) = 0$,
2. $b_i = a_{f(i)}$ for all $i \geq 1$,
3. $a_j = \nu$ for all j not in the range of f,
4. For all $i \geq 0$ such that ω_i is not the last trajectory in Σ,
   (a) $s_j \in rng(\omega_i)$ for all j, $f(i) \leq j < f(i+1)$,
   (b) $s_{f(i)}.now = ftime(\omega_i)$, and
   (c) $s_{f(i+1)-1}.now = ltime(\omega_i)$.
5. If ω_i is the last trajectory in Σ, then
   (a) $s_j \in rng(\omega_i)$ for all j, $f(i) \leq j$,
   (b) $s_{f(i)}.now = ftime(\omega_i)$, and
   (c) $sup\{s_j.now \mid f(i) \leq j\} = ltime(\omega_i)$.

In other words, the function f in this definition maps the (indices of) actions in Σ to corresponding (indices of) actions in α, in such a way that exactly the non-time-passage actions of α are included in the image. Condition 4 is a consistency condition relating the first and last times for each non-final trajectory to the times produced by the appropriate steps of α. Condition 5 gives a similar consistency condition for the first time of the final trajectory (if any); in place of the consistency condition for the last time, there is a "cofinality" condition asserting that the times grow to the same limit in both executions.

The following two straightforward lemmas show the relationship between timed execution fragments and ordinary execution fragments.

Lemma 4.3

Let A be a timed automaton. If α is an execution fragment of A, then there is a timed execution fragment Σ of A such that α samples Σ. ■

Lemma 4.4

Let A be a timed automaton. If Σ is a timed execution fragment of A, then there is an execution fragment α of A such that α samples Σ. ■

Define a state s to be *t-reachable* in timed automaton A provided that there is a finite timed execution Σ such that $lstate(\Sigma) = s$. The following lemma shows that t-reachability can equivalently be defined by means of ordinary executions.

Lemma 4.5

State s is t-reachable in A iff it is reachable in A.

Proof. Straightforward using Lemmas 4.3 and 4.4. ■

An important consequence of Lemma 4.5 is that any technique that can prove that a property holds for all final states of finite (ordinary) executions is a sound technique for proving that a property holds in all t-reachable states of a timed automaton. Most importantly, induction on the steps of ordinary executions is sound in this sense. Conversely, any technique that can prove that a property holds for all t-reachable states also proves that it holds for all reachable states.

Finite, Admissible and Zeno Execution Fragments

An execution fragment α is *finite* if it is a finite sequence. In the timed model, an execution fragment α is defined to be *admissible* if there is no finite upper bound on the *now* values of the states in α. Finally, an execution fragment is said to be *Zeno* if it is neither finite nor admissible.

Lemma 4.6

If α samples Σ then

1. *α is finite iff Σ is finite,*
2. *α is admissible iff Σ is admissible, and*
3. *α is Zeno iff Σ is Zeno.* ■

Timed Traces

It is possible to give a sensible definition of the timed trace of an ordinary execution fragment of a timed automaton. Namely, suppose $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ is a execution fragment of a timed automaton A. First, define $ltime(\alpha)$ to be the supremum of the now values of all the states in α. Then let δ be the sequence consisting of the actions in α paired with their times of occurrence:

$$\delta = (a_1, s_1.now)(a_2, s_2.now)\cdots.$$

Then $t\text{-}trace(\alpha)$, the *timed trace* of α, is defined to be the pair

$$t\text{-}trace(\alpha) \triangleq (\delta \upharpoonright (vis(A) \times \mathsf{T}), ltime(\alpha))$$

The following lemma shows that the definitions of timed traces for execution fragments and timed execution fragments are properly related:

Lemma 4.7

If α samples Σ then $t\text{-}trace(\alpha) = t\text{-}trace(\Sigma)$. ■

4.2 Live Timed Automata

The notion of *live timed automaton* is now introduced. The definition is similar to the definition of a live automaton in the untimed model (Definition 3.2) except for the fact that the liveness condition is a set of *timed* executions.

Definition 4.8 (Live timed automaton)

A *timed liveness condition* L for a timed automaton A is a subset of the timed executions of A such that any finite timed execution of A has an extension in L. Formally, $L \subseteq t\text{-}exec(A)$ such that for all $\Sigma \in t\text{-}exec^*(A)$ there exists a $\Sigma' \in t\text{-}frag(A)$, such that $\Sigma \frown \Sigma' \in L$.

A *live timed automaton* is a pair (A, L), where A is a timed automaton and L is a timed liveness condition for A. The timed executions of L are called the *live timed executions* of A. ■

4.3 Safe Timed I/O Automata

Definition 4.9 (Safe timed I/O automaton)

A *safe timed I/O automaton* is a timed automaton augmented with a *visible action signature*, $vsig(A) = (in(A), out(A))$, which partitions $vis(A)$ into input and output actions. A must be *input-enabled*.

The internal and output actions of a safe timed I/O automaton A are referred to as the *locally-controlled* actions of A, written $local(A)$. Thus, $local(A) = int(A) \cup out(A)$. ■

Parallel composition of safe timed I/O automata is defined similarly to the corresponding definition for the untimed model (Definition 3.4). However, the time-passage steps and the *now* mappings of the component safe timed I/O automata need special treatment. Specifically, time is only allowed to pass by a certain amount in the composition if all components allow the same amount of time to pass. Also, the state space of the composition consists of all states in the cartesian product of the component state spaces where the component states have the same *now* values. Thus, the components must agree on the time. The *now* mapping of the composition is then defined to be the *now* mapping of any of the components.

Definition 4.10 (Parallel composition)

Safe timed I/O automata $A_1, \ldots, A_N$ are *compatible* if for all $1 \leq i, j \leq N$ with $i \neq j$, the following conditions hold:

1. $out(A_i) \cap out(A_j) = \emptyset$
2. $int(A_i) \cap acts(A_j) = \emptyset$

The *parallel composition* $A_1 \| \cdots \| A_N$ of compatible safe timed I/O automata $A_1, \ldots, A_N$ is the safe timed I/O automaton A such that

1. $states(A) = \{(s_1, \ldots, s_N) \in states(A_1) \times \cdots \times states(A_N) \mid s_1.now_{A_1} = \cdots = s_N.now_{A_N}\}$
2. $start(A) = start(A_1) \times \cdots \times start(A_N)$
3. $(s_1, \ldots, s_N).now_A = s_1.now_{A_1}$ $(= s_2.now_{A_2} = \cdots = s_N.now_{A_N})$
4. $out(A) = out(A_1) \cup \cdots \cup out(A_N)$
5. $in(A) = (in(A_1) \cup \cdots \cup in(A_N)) \setminus out(A)$
6. $int(A) = int(A_1) \cup \cdots \cup int(A_N)$
7. $((s_1, \ldots, s_N), a, (s'_1, \ldots, s'_N)) \in steps(A)$ iff for all $1 \leq i \leq N$
   (a) if $a \in acts(A_i)$ then $(s_i, a, s'_i) \in steps(A_i)$
   (b) if $a \notin acts(A_i)$ then $s_i = s'_i$ ■

Note, how Condition 7 of the definition captures both time-passage steps (where all components participate) and other steps (where a subset of the components participate).

Lemma 3.5 carries over to the timed case. However, a new definition of projection is needed for timed executions. Specifically, let $A = A_1 \| \cdots \| A_N$. For any function ω from an interval of time to *states*(A), define $\omega \lceil A_i$ to be obtained from ω by projecting every state in the range of ω to A_i. Let $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ be an alternating sequence of functions from intervals of time to *states*(A) and actions from $acts(A) \setminus \{\nu\}$ such that Σ does not end in an action if it is a finite sequence. Then the projection $\Sigma \lceil A_i$ of Σ onto A_i is obtained by projecting each ω_k of

Σ onto A_i, removing each action a_j that is not an action of A_i, and concatenating each pair of (projected) functions ω_k, ω_{k+1} whose interleaved action is removed.

The following lemma relates the timed executions of a composed timed automaton with those of the component timed automata.

Lemma 4.11

Let $A = A_1 \| \cdots \| A_N$. Let $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ be an alternating sequence of functions from intervals of time to states(A) and actions from acts(A) $\setminus \{\nu\}$ such that Σ does not end in an action if it is a finite sequence, the domain of each function ω_j that is not the last function of Σ is closed, and, if Σ is a finite sequence, the domain of the last function of Σ is left closed. Let consistent(Σ) be the predicate that is true iff for each A_i and each j such that $a_j \notin acts(A_i)$, $lstate(\omega_{j-1})\lceil A_i = fstate(\omega_j)\lceil A_i$. Then,

1. *consistent(Σ) and $\Sigma\lceil A_i \in$ t-exec$^*(A_i)$, for all A_i, iff $\Sigma \in$ t-exec$^*(A)$.*

2. *consistent(Σ) and $\Sigma\lceil A_i \in$ t-exec$^\infty(A_i)$, for all A_i, iff $\Sigma \in$ t-exec$^\infty(A)$.*

3. *consistent(Σ) and $\Sigma\lceil A_i \in$ t-exec$^Z(A_i)$, for all A_i, iff $\Sigma \in$ t-exec$^Z(A)$.*

4. *consistent(Σ) and $\Sigma\lceil A_i \in$ t-exec(A_i), for all A_i, iff $\Sigma \in$ t-exec(A).*

If $\Sigma \in$ t-exec(A) then, for all i, ltime(Σ) = ltime$(\Sigma\lceil A_i)$. ■

We now turn attention to the timed versions of action hiding and action renaming. The only changes from the untimed model are the handling of the *now* component and the fact that the time-passage action, ν, may not be renamed.

Definition 4.12 (Action hiding)

Let A be a safe timed I/O automaton and let Λ be a set of actions such that $\Lambda \subseteq local(A)$. Then define $A \setminus \Lambda$ to be the safe timed I/O automaton such that

1. $states(A \setminus \Lambda) = states(A)$

2. $start(A \setminus \Lambda) = start(A)$

3. $now_{A\setminus\Lambda} = now_A$

4. $in(A \setminus \Lambda) = in(A)$

5. $out(A \setminus \Lambda) = out(A) \setminus \Lambda$

6. $int(A \setminus \Lambda) = int(A) \cup \Lambda$

7. $steps(A \setminus \Lambda) = steps(A)$ ■

Lemma 4.13

Let A be a safe timed I/O automaton and let Λ be a set of actions such that $\Lambda \subseteq local(A)$. Then

1. $t\text{-}exec^*(A \setminus \Lambda) = t\text{-}exec^*(A)$
2. $t\text{-}exec^\infty(A \setminus \Lambda) = t\text{-}exec^\infty(A)$
3. $t\text{-}exec^Z(A \setminus \Lambda) = t\text{-}exec^Z(A)$
4. $t\text{-}exec(A \setminus \Lambda) = t\text{-}exec(A)$ ∎

Definition 4.14 (Action renaming)

A mapping ρ from actions to actions is *applicable* to a safe timed I/O automaton A if it is injective, $acts(A) \subseteq dom(\rho)$, and $\rho(\nu) = \nu$. Given a safe timed I/O automaton and a mapping ρ applicable to A, define $\rho(A)$ to be the safe timed I/O automaton such that

1. $states(\rho(A)) = states(A)$
2. $start(\rho(A)) = start(A)$
3. $now_{\rho(A)} = now_A$
4. $in(\rho(A)) = \rho(in(A))$
5. $out(\rho(A)) = \rho(out(A))$
6. $int(\rho(A)) = \rho(int(A))$
7. $steps(\rho(A)) = \{(s, \rho(a), s') \mid (s, a, s') \in steps(A)\}$ ∎

Lemma 4.15

Let A be a safe timed I/O automaton and ρ be a mapping applicable to A. For any timed execution Σ of A, let $\rho(\Sigma)$ denote the sequence obtained by replacing each occurrence of every action a in Σ by $\rho(a)$, and for any set L of timed executions of A, let $\rho(L) = \{\rho(\Sigma) \mid \Sigma \in L\}$. Then

1. $t\text{-}exec^*(\rho(A)) = \rho(t\text{-}exec^*(A))$
2. $t\text{-}exec^\infty(\rho(A)) = \rho(t\text{-}exec^\infty(A))$
3. $t\text{-}exec^Z(\rho(A)) = \rho(t\text{-}exec^Z(A))$
4. $t\text{-}exec(\rho(A)) = \rho(t\text{-}exec(A))$ ∎

4.4 Live Timed I/O Automata

In order to define live timed I/O automata, the notion of environment-freedom is generalized to timed systems. As for the untimed model a live timed I/O automaton is environment-free if it can behave properly independently of the behavior of the environment. Specifically, a game is set up between a timed automaton and its environment and the timed automaton is environment-free iff it has a winning strategy against its environment.

The notion of strategy is similar to the one used for the untimed model. However, the presence of time has a strong impact on the type of interactions that can occur between a timed automaton and its environment.

In the untimed model the environment is allowed to provide any finite number of input actions at each move, and the system is allowed to perform at most one of its locally-controlled actions at each move. Thus, the fact that the environment can be arbitrarily fast with respect to the system, but not infinitely fast, is reflected in the structure of the environment moves. This structure is not needed in the timed model since actions in the timed model are associated with specific times. In particular, the relative speeds of the system and the environment are given directly by their timing constraints. The behavior of the environment during the game can be represented simply as a timed sequence over input actions.

In the untimed model a strategy is not allowed to base its decisions on any future input actions from the environment. In the timed model, not only is the strategy not allowed to know about the occurrence of future input actions, but the strategy is also not allowed to know anything about the *timing* of such input actions, e.g., that no inputs will arrive in the next ϵ time units. Thus, if a strategy in the timed model decides to let time pass, it is required to specify explicitly all intermediate states. By specifying all states at intermediate times for a time-passage step, the current state of the system will always be known should the time-passage step be interrupted by an input action.

As in the untimed model, a strategy in the timed model is a pair of function (g, f). Function f takes a finite timed execution and decides how the system behaves till its next locally-controlled action under the assumption that no input are received in the meantime; function g decides what state to reach whenever some input is received.

Definition 4.16 (Strategy)

Consider any safe timed I/O automaton A. A *strategy* defined on A is a pair of functions (g, f) where $g : \textit{t-exec}^*(A) \times \textit{in}(A) \rightarrow \textit{states}(A)$ and $f : \textit{t-exec}^*(A) \rightarrow (\textit{traj}(A) \times \textit{local}(A) \times \textit{states}(A)) \cup \textit{traj}(A)$, where $\textit{traj}(A)$ denotes the set of A-trajectories, such that

1. $g(\Sigma, a) = s$ implies $\Sigma a\{s\} \in \textit{t-exec}^*(A)$
2. $f(\Sigma) = (\omega, a, s)$ implies $\Sigma \frown \omega a\{s\} \in \textit{t-exec}^*(A)$
3. $f(\Sigma) = \omega$ implies $\Sigma \frown \omega \in \textit{t-exec}^\infty(A)$
4. f is *consistent*, i.e., if $f(\Sigma) = (\omega, a, s)$, then, for each t, $\textit{ftime}(\omega) \leq t \leq \textit{ltime}(\omega)$, $f(\Sigma \frown (\omega \lhd t)) = (\omega \rhd t, a, s)$, and, if $f(\Sigma) = \omega$, then, for each t, $\textit{ftime}(\omega) \leq t < \textit{ltime}(\omega)$, $f(\Sigma \frown (\omega \lhd t)) = \omega \rhd t$.

For notational convenience define $f(\Sigma).trj = \begin{cases} \omega & \text{if } f(\Sigma) = (\omega, a, s) \\ \omega & \text{if } f(\Sigma) = \omega \end{cases}$

∎

Condition 1 of Definition 4.16 states that g returns a "legal" next state given an input. Conditions 2 and 3 state the two possible system moves given by f: either f specifies time-passage followed by a local step, or f specifies that the system simply lets time pass forever. Note that f specifies all states during time passage. The consistency condition (Condition 4) for f says that, whenever after a finite timed execution Σ the system decides to behave according to $\omega a\{s\}$ or ω, after performing a part of ω the system would decide to behave according to the rest of $\omega a\{s\}$ or ω. In other words, a strategy decision cannot change in the absence of some inputs. The consistency condition is required for the closure of the composition operator.

The game between the system and the environment works as follows. The environment can provide any input at any time, while the system lets time pass and provides locally-controlled actions based on its strategy. It is very important for the system moves not to be based on the future moves of the environment. Specifically, at any point in time the system decides its next move using function f. If an input comes, the system will perform its current step just until the time at which the input occurs, and then use function g to compute the state reached as a result of the input.

A new problem arises when the system decides to perform an action at the same time at which the environment is providing some input. Our model does not rule out such race conditions. Practical examples of such situations arise whenever the system has some timeout mechanism and the input occurs exactly when the timeout period expires. The race conditions are modeled as nondeterministic choices. As a consequence, the *outcome*, i.e., the result of the game, for a timed strategy is a set of timed executions.

The following definition of the outcome of a strategy for safe timed I/O automata closely parallels the corresponding definition in the untimed model.

Definition 4.17 (Outcome of a strategy)

Let A be a safe timed I/O automaton and (g, f) a strategy defined on A. Define a *timed environment sequence* for A to be a timed sequence over $in(A)$, and define a timed environment sequence $\mathcal{I}$ for A to be *compatible* with a timed execution fragment Σ of A if either $\mathcal{I}$ is empty, or Σ is finite and $ltime(\Sigma) \leq ftime(\mathcal{I})$. Then define $R_{(g,f)}$, the *next-relation induced by* (g, f), as follows: for any $\Sigma, \Sigma' \in t\text{-}exec(A)$ and any $\mathcal{I}, \mathcal{I}'$ compatible with Σ, Σ', respectively, $((\Sigma, \mathcal{I}), (\Sigma', \mathcal{I}')) \in R_{(g,f)}$ iff

$$
(\Sigma', \mathcal{I}') = \begin{cases}
(\Sigma \frown \omega a\{s\}, \mathcal{I}) & \text{where } \Sigma \text{ is finite, } \mathcal{I} = \varepsilon,\ f(\Sigma) = (\omega, a, s), \\[2ex]
(\Sigma \frown \omega, \mathcal{I}) & \text{where } \Sigma \text{ is finite, } \mathcal{I} = \varepsilon,\ f(\Sigma) = \omega, \\[2ex]
(\Sigma \frown \omega a\{s\}, \mathcal{I}) & \text{where } \Sigma \text{ is finite, } \mathcal{I} = (b,t)\mathcal{I}'', f(\Sigma) = (\omega, a, s), \\
& \qquad\qquad \mathit{ltime}(\omega) \leq t, \\[2ex]
(\Sigma \frown \omega' a\{s'\}, \mathcal{I}'') & \text{where } \Sigma \text{ is finite, } \mathcal{I} = (a,t)\mathcal{I}'',\ f(\Sigma).\mathit{trj} = \omega, \\
& \qquad\qquad \mathit{ltime}(\omega) \geq t,\ \omega' = \omega \lhd t,\ g(\Sigma \frown \omega', a) = s', \text{ or} \\[2ex]
(\Sigma, \mathcal{I}) & \text{where } \Sigma \text{ is not finite.}
\end{cases}
$$

Let Σ be a finite timed execution of A, and $\mathcal{I}$ be a timed environment sequence for A *compatible* with Σ.

An *outcome sequence of* (g, f) *given* Σ *and* $\mathcal{I}$ is an infinite sequence $(\Sigma^n, \mathcal{I}^n)_{n \geq 0}$ that satisfies:

- $(\Sigma^0, \mathcal{I}^0) = (\Sigma, \mathcal{I})$ and
- for all $n > 0$, $((\Sigma^{n-1}, \mathcal{I}^{n-1}), (\Sigma^n, \mathcal{I}^n)) \in R_{(g,f)}$.

Note, that $(\Sigma^n)_{n \geq 0}$ forms a chain ordered by *t-prefix*.

The *outcome* $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I})$ of the strategy (g, f) given Σ and $\mathcal{I}$ is the set of timed executions Σ' for which there exists an outcome sequence $(\Sigma^n, \mathcal{I}^n)_{n \geq 0}$ of (g, f) given Σ and $\mathcal{I}$ such that $\Sigma' = \lim_{n \to \infty} \Sigma^n$. ∎

The set of outcome sequences of (g, f) given some Σ and $\mathcal{I}$ is determined step by step using the next-relation $R_{(g,f)}$. In the definition of $R_{(g,f)}$, the first, second, and third cases deal with different situations in which no input occurs during the system move chosen by f. The fourth case takes care of the situation in which inputs do occur during the system move chosen by f. Note, that the third and fourth cases may both be applicable whenever the next input action of $\mathcal{I}$ and the local action chosen by f occur at the same time. This is why the outcome is a *set* of timed executions. Finally, the fifth case is needed for technical convenience, since the second case generates an admissible timed execution.

The following lemma states that an outcome set is never empty and that an element of an outcome cannot be finite. Furthermore, if an element of an outcome is Zeno, it contains infinitely many actions (other than the implicit time-passage actions).

Lemma 4.18

Let A be a safe timed I/O automaton, (g, f) a strategy defined on A, Σ a finite timed execution of A, and $\mathcal{I}$ a timed environment sequence for A compatible with Σ. Then $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \neq \emptyset$ and $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq (t\text{-}exec^{\infty}(A) \cup t\text{-}exec^{Z}(A))$. Furthermore, if $\Sigma' \in \mathcal{O}_{(g,f)}(\Sigma, \mathcal{I})$ and $\Sigma' \in t\text{-}exec^{Z}(A)$, then $|\Sigma' \restriction acts(A)| = \infty$.

Proof. Let $R_{(g,f)}$ be the next-relation induced by (g, f). Construct an outcome sequence of (g, f) given Σ and $\mathcal{I}$ inductively as follows. Define $(\Sigma^0, \mathcal{I}^0) = (\Sigma, \mathcal{I})$. For any $n > 0$, assume $(\Sigma^{n-1}, \mathcal{I}^{n-1})$ has been defined. Then it is easy to see that the condition of at least one case in the definition of $R_{(g,f)}$ is satisfied. Thus, define $(\Sigma^n, \mathcal{I}^n)$ to be any pair such that $((\Sigma^{n-1}, \mathcal{I}^{n-1}), (\Sigma^n, \mathcal{I}^n)) \in R_{(g,f)}$. This inductively defined outcome sequence gives rise to an element in $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I})$. That proves that $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I})$ is not empty.

Now, let $(\Sigma^n, \mathcal{I}^n)$ be an arbitrary outcome sequence of (g, f) given Σ and $\mathcal{I}$. Clearly, $\Sigma^0 = \Sigma \in$ *t-exec*(A). Now assume, that $\Sigma^n \in$ *t-exec*(A). Then, by the four conditions of Definition 4.16, it is easy to see that also $\Sigma^{n+1} \in$ *t-exec*(A). Thus, by induction, $\Sigma^n \in$ *t-exec*(A) for all $n \geq 0$. Now, assume $\Sigma' = \lim_{n\to\infty}(\Sigma^n) \notin$ *t-exec*(A). Then there must be a finite *t-prefix* Σ'' of Σ' such that $\Sigma'' \notin$ *t-exec*$^*(A)$. Also, Σ'' must be a *t-prefix* of Σ^n for some n. However, this contradicts the fact that $\Sigma^n \in$ *t-exec*(A). Thus, $\Sigma' \in$ *t-exec*(A).

Now, assume that Σ' is finite. Then there exists a number n' such that for all $n > n'$, $\Sigma^n = \Sigma^{n-1} = \Sigma'$, but this contradicts the definition of $R_{(g,f)}$. Thus, $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq$ (*t-exec*$^\infty(A) \cup$ *t-exec*$^Z(A)$)

Finally, it is easy to see that if $\Sigma' \in$ *t-exec*$^Z(A)$, then Σ' is an infinite sequence of trajectories and actions. Only the second case in the definition of $R_{(g,f)}$ can lead to a finite sequence, but in this case the outcome will be admissible (cf. Definition 4.16 Condition 3). This proves the final part of the lemma. ∎

Another problem due to the explicit presence of time in the model is the capability of a system to block time. Under the reasonable assumption that it is natural for a system to require that time advances forever, a timed automaton that blocks time cannot be environment-free. Thus, we could assume that finite and Zeno timed executions are not live and that the environment cannot block time. However, as is illustrated in the following example due to Lamport, Zeno timed executions cannot be ignored completely.

Example 4.19

Consider two safe timed I/O automata A, B such that $in(A) = out(B) = \{b\}$ and $out(A) = in(B) = \{a\}$. Let A start by performing its output action a and let B start by waiting for some input. Furthermore, let both A and B reply to their n^{th} input with an output action exactly $1/2^n$ time units after the input has occurred.

Consider the following definition of environment-freedom, which assumes that the environment does not behave in a Zeno manner: a pair (A, L) is environment-free iff there exists a strategy (g, f) defined on A such that for each finite timed execution Σ of A and any admissible timed environment sequence $\mathcal{I}$ for A compatible with Σ we have $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq L$. Then it is easy to observe that, if L_A and L_B are defined to be the set of admissible timed executions of A and B, respectively, the pairs (A, L_A) and (B, L_B) are environment-free. However, the parallel composition of A and B yields no admissible executions, rather it only yields a Zeno timed execution, which blocks time. Thus, the parallel composition of (A, L_A) and (B, L_B) constrains the environment. Observe that (A, L_A) and (B, L_B) "unintentionally" collaborate to generate a Zeno timed execution: each pair looks like a Zeno environment to the other. ■

To eliminate the problem of Example 4.19 one must ensure that a system does not collaborate with its environment to generate a Zeno timed execution. We call *Zeno-tolerant* those timed executions where such a collaboration does not arise.

Definition 4.20 (Special types of timed executions)

Given a safe timed I/O automaton A, and given a timed execution Σ of A,

- Σ is said to be *environment-Zeno* if Σ is a Zeno timed execution that contains infinitely many input actions;
- Σ is said to be *system-Zeno* if Σ is a Zeno timed execution that either contains infinitely many locally-controlled actions or contains finitely many actions;
- Σ is said to be *Zeno-tolerant* if it is an environment-Zeno, non-system-Zeno timed execution; equivalently, Σ is Zeno-tolerant if
  1. $ltime(\Sigma)$ is finite,
  2. Σ contains infinitely many input actions, and
  3. Σ contains finitely many locally-controlled actions.

  Denote by $t\text{-}exec^{Zt}(A)$ the set of Zeno-tolerant timed executions of A. ■

A Zeno-tolerant strategy guarantees that the system never chooses to block time in order to win its game against the environment. That is, a Zeno-tolerant strategy produces Zeno timed executions only when applied to a Zeno timed environment sequence $\mathcal{I}$, and in these cases the outcome is Zeno-tolerant. Thus, the system does not respond to Zeno inputs by behaving in a Zeno fashion.

Definition 4.21 (Zeno-tolerant strategy)

A strategy (g, f) defined on a safe timed I/O automaton A is said to be *Zeno-tolerant* if, for every finite timed execution $\Sigma \in t\text{-}exec^*(A)$ and every timed environment sequence $\mathcal{I}$ for A compatible with Σ, $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq t\text{-}exec^\infty(A) \cup t\text{-}exec^{Zt}(A)$. ■

Now the definition of environment-freedom for the timed model is possible.

Definition 4.22 (Environment-freedom)

A pair (A, L) where A is a safe timed I/O automaton and $L \subseteq t\text{-}exec(A)$ is *environment-free* iff there exists a Zeno-tolerant strategy (g, f) defined on A such that for each finite timed execution Σ of A and each timed environment sequence $\mathcal{I}$ for A compatible with Σ, $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq L$. The pair (g, f) is called an *environment-free* strategy for (A, L). ■

A pair (A, L) is environment-free if, after any finite timed execution and with any (Zeno or non-Zeno) sequence of input actions, it can generate some admissible or Zeno-tolerant timed execution in A. Also, A must never generate one of its finite or system-Zeno timed executions, since it would constrain its environment in this case. Thus liveness conditions should not include any finite or system-Zeno timed execution. Zeno-tolerant timed executions are used only to handle illegal interactions, and therefore also should not be included in liveness conditions. This leads to the definition of *live timed I/O automata*, where the liveness condition contains only admissible timed executions, but the strategy is allowed to yield *Zeno-tolerant* outcomes when given a Zeno timed environment sequence.

Definition 4.23 (Live timed I/O automaton)

A *live timed I/O automaton* is a pair (A, L), where A is a safe timed I/O automaton and $L \subseteq \textit{t-exec}^{\infty}(A)$, such that the pair $(A, L \cup \textit{t-exec}^{Zt}(A))$ is environment-free. ■

Lemma 4.24

If (A, L) is a live timed I/O automaton, then L is a timed liveness condition for A.

Proof. Given a finite timed execution Σ of A, consider an environment-free strategy (g, f) for $(A, L \cup \textit{t-exec}^{Zt}(A))$. Consider any timed execution $\Sigma \frown \Sigma' \in \mathcal{O}_{(g,f)}(\Sigma, \varepsilon)$. Such a timed execution exists according to Lemma 4.18. The timed execution $\Sigma \frown \Sigma'$ is not Zeno-tolerant since it contains finitely many input actions. Therefore $\Sigma \frown \Sigma'$ is a timed execution of L, i.e., Σ can be extended to a timed execution of L. ■

As in the untimed model, the parallel composition, action hiding, and action renaming operators defined for safe timed I/O automata are extended to live timed I/O automata.

Definition 4.25 (Parallel composition)

Live timed I/O automata $(A_1, L_1), \ldots, (A_N, L_N)$ are *compatible* iff the safe timed I/O automata $A_1, \ldots, A_N$ are compatible.

The *parallel composition* $(A_1, L_1) \| \cdots \| (A_N, L_N)$ of compatible live timed I/O automata $(A_1, L_1), \ldots, (A_N, L_N)$ is defined to be the pair (A, L) where $A = A_1 \| \cdots \| A_N$ and $L = \{\Sigma \in \textit{t-exec}(A) \mid \Sigma \lceil A_1 \in L_1, \ldots, \Sigma \lceil A_N \in L_N\}$. ■

The restriction of the parallel composition operator to finitely many components can now be justified with the following example.

Example 4.26

Let $\{(A_i, L_i)\}_{i \geq 0}$ be a family of infinitely many live timed I/O automata such that each safe I/O automaton A_i has a unique distinct output action a_i which executes at time 1, and L_i is the set of admissible executions of A_i. The parallel composition $\|_{i \geq 0}(A_i, L_i)$ exhibits finite or Zeno timed executions only since it can never reach a time greater than 1. Specifically, infinitely many actions, i.e., the set $\{a_i\}_{i\geq 0}$, must be executed at time 1. Thus $\|_{i\geq 0}(A_i, L_i)$ cannot be an environment-free pair. ■

Definition 4.27 (Action Hiding)

Let (A, L) be a live timed I/O automaton and let Λ be a set of actions such that $\Lambda \subseteq \mathit{local}(A)$. Then define $(A, L) \setminus \Lambda$ to be the pair $(A \setminus \Lambda, L)$. ■

Definition 4.28 (Action Renaming)

A mapping ρ from actions to actions is *applicable* to a live timed I/O automaton (A, L) if it is applicable to A. Let Σ be a timed execution of (A, L). Define $\rho(\Sigma)$ to be the sequence that results from replacing each occurrence of every action a in Σ by $\rho(a)$. Given a live timed I/O automaton and a mapping ρ applicable to (A, L), define $\rho((A, L))$ to be the pair $(\rho(A), \{\rho(\Sigma) \mid \Sigma \in L\})$. ■

As expected, the three operators above are closed for live timed I/O automata in the sense that they produce a new live timed I/O automaton. As for the untimed model, this is easy to prove for action hiding and renaming, but fairly complicated for parallel composition.

Proposition 4.29 (Closure of action hiding)

Let (A, L) be a live timed I/O automaton and let $\Lambda \subseteq \mathit{local}(A)$. Then $(A, L) \setminus \Lambda$ is a live timed I/O automaton.

Proof. Let $(A_\Lambda, L_\Lambda) = (A, L) \setminus \Lambda$, i.e., $(A_\Lambda, L_\Lambda) = (A \setminus \Lambda, L)$ by Definition 4.27. From Definition 4.12 A_Λ is a safe timed I/O automaton. Furthermore, Lemma 4.13 gives $t\text{-}exec^\infty(A_\Lambda) = t\text{-}exec^\infty(A)$. Therefore $L_\Lambda \subseteq t\text{-}exec^\infty(A_\Lambda)$ as required by the definition of live timed I/O automata (Definition 4.23).

To show that the pair $(A_\Lambda, L_\Lambda \cup L'_\Lambda)$, where $L'_\Lambda = t\text{-}exec^{Zt}(A_\Lambda)$, is environment-free, it suffices to note that any environment-free strategy for $(A, L \cup L')$, where $L' = t\text{-}exec^{Zt}(A)$, is also an environment-free strategy for $(A_\Lambda, L_\Lambda \cup L'_\Lambda)$. In fact the hiding operator simply changes some output actions into internal actions. The remaining structure of a live timed I/O automaton, including its set of locally-controlled actions, is not affected. ■

Proposition 4.30 (Closure of action renaming)

Let (A, L) be a live timed I/O automaton and let ρ be a mapping applicable to (A, L). Then $\rho((A, L))$ is a live timed I/O automaton.

Proof. Consider any timed execution Σ of (A, L). Let $\rho(\Sigma)$ be the sequence obtained by replacing each occurrence of every action a in Σ by $\rho(a)$. If S is a set of timed executions of (A, L), let $\rho(S) = \{\rho(\Sigma) \mid \Sigma \in S\}$.

Let $(A_\rho, L_\rho) = \rho((A, L))$, i.e., $(A_\rho, L_\rho) = (\rho(A), \rho(L))$ by Definition 4.28. From Definition 4.14 A_ρ is a safe timed I/O automaton. Furthermore, Lemma 4.15 gives $t\text{-}exec^\infty(A_\rho) =$

$\rho(t\text{-}exec^{\infty}(A))$. Therefore, since $L \subseteq t\text{-}exec^{\infty}(A)$, $L_\rho \subseteq t\text{-}exec^{\infty}(A_\rho)$ as required by the definition of live timed I/O automata (Definition 4.23).

Let $L' = t\text{-}exec^{Zt}(A)$ and let (g, f) be an environment-free strategy for $(A, L \cup L')$. Furthermore, let $L'_\rho = t\text{-}exec^{Zt}(A_\rho)$. Then Lemma 4.15 and the fact that ρ maps $local(A)$ to $local(A_\rho)$ and $in(A)$ to $in(A_\rho)$ implies that $L'_\rho = \rho(L')$, which in turn implies $L_\rho \cup L'_\rho = \rho(L \cup L')$. Now, define a strategy (g_ρ, f_ρ) for A_ρ as follows:

$$g_\rho(\rho(\Sigma), \rho(a)) \quad = \quad g(\Sigma, a)$$

$$f_\rho(\rho(\Sigma)) \quad = \quad \begin{cases} (\omega, \rho(a), s') & \text{if } f(\Sigma) = (\omega, a, s') \\ \omega & \text{if } f(\Sigma) = \omega \end{cases}$$

It is now trivial to verify that (g_ρ, f_ρ) is an environment-free strategy for $(A_\rho, L_\rho \cup L'_\rho)$. Consequently, (A_ρ, L_ρ) is a live timed I/O automaton. ■

As in the untimed model, the proof of closure of the parallel composition operator is considerably more complicated than the proof of closure for action hiding and action renaming. For compatible live timed I/O automata $(A_1, L_1), \ldots, (A_N, L_N)$, let (A, L) denote the parallel composition $(A_1, L_1)\| \cdots \|(A_N, L_N)$. In order to prove that (A, L) is a live timed I/O automaton one must show that $(A, L \cup t\text{-}exec^{Zt}(A))$ is environment-free, which, in turn, requires finding an environment-free strategy for $(A, L \cup t\text{-}exec^{Zt}(A))$.

The proof proceeds by first defining a strategy (g, f) for (A, L) based on a strategy (g_i, f_i) for each $(A_i, L_i \cup t\text{-}exec^{Zt}(A_i))$, and then proving that (g, f) is an environment-free strategy for $(A, L \cup t\text{-}exec^{Zt}(A))$.

Function g computes, given input a, the next state according to the g_i functions of those components of A for which a is an input action, and simply leave the state unchanged for those components for which a is not an action. Function f determines, using each f_i, which component wishes to execute the next locally-controlled action. Say this is the k^{th} component and it wishes to perform action a at time t. Then each component A_i evolves based on f_i up to time t. Furthermore, at time t, A_k takes a step based on f_k and each A_i for which a is an input action takes a step based on g_i.

Definition 4.31 (Parallel composition of (timed) strategies)

Let $A = A_1\| \cdots \|A_N$ be the parallel composition of compatible safe timed I/O automata $A_1, \ldots, A_N$, and let (g_i, f_i), for each $1 \leq i \leq N$, be a strategy defined on A_i.

The *parallel composition* $(g_1, f_1)\| \cdots \|(g_N, f_N)$ of the strategies $(g_1, f_1), \ldots, (g_N, f_N)$ is the pair of functions (g, f)

$$g : t\text{-}exec^*(A) \times in(A) \to states(A)$$
$$f : t\text{-}exec^*(A) \to (traj(A) \times local(A) \times states(A)) \cup traj(A)$$

such that

$$g(\Sigma, a) = s \quad \text{where, for all } 1 \leq i \leq N, \quad s\lceil A_i = \begin{cases} g_i(\Sigma\lceil A_i, a) & \text{for } a \in in(A_i) \\ lstate(\Sigma)\lceil A_i & \text{for } a \notin acts(A_i) \end{cases}$$

and f is defined as follows: For all $1 \leq i \leq N$, define ω_i to be $f_i(\Sigma \lceil A_i).trj$. Pick any k, say the smallest, such that $ltime(\omega_k) = \min_{1 \leq i \leq N}(ltime(\omega_i))$. Define ω such that

$$\omega \lceil A_i = \begin{cases} \omega_k & \text{if } i = k \\ \omega_i \lhd ltime(\omega_k) & \text{if } i \neq k \end{cases}$$

The definition of $f(\Sigma)$ has two cases.

1. If $f_k(\Sigma \lceil A_k) = (\omega_k, a, s_k)$ then $f(\Sigma) = (\omega, a, s)$,

   where, for all $1 \leq i \leq N$, $s \lceil A_i = \begin{cases} s_k & \text{if } i = k \\ g_i((\Sigma \frown \omega) \lceil A_i, a) & \text{if } i \neq k \text{ and } a \in in(A_i) \\ lstate(\omega) \lceil A_i & \text{if } i \neq k \text{ and } a \notin acts(A_i) \end{cases}$

2. If $f_k(\Sigma \lceil A_k) = \omega_k$ then $f(\Sigma) = \omega$. ■

Lemma 4.32

Let $A_1, \ldots, A_N$ be compatible safe timed I/O automata and let, for each $1 \leq i \leq N$, (g_i, f_i) be a strategy defined on A_i. Then $(g_1, f_1) \| \cdots \| (g_N, f_N)$ is a strategy defined on $A_1 \| \cdots \| A_N$.

Proof. Let $(g, f) = (g_1, f_1) \| \cdots \| (g_N, f_N)$ and $A = A_1 \| \cdots \| A_N$. To prove that (g, f) is a strategy defined on A, the four conditions of Definition 4.16 must be checked:

1–3. Conditions 1–3 are trivial to check given the definitions of g and f, and the fact that, for all i, (g_i, f_i) is a strategy defined on A_i.

4. Consider the *consistency condition* of Definition 4.16. First assume, for an arbitrary $\Sigma \in t\text{-}exec^*(A)$, that $f(\Sigma) = (\omega, a, s)$, and let t be an arbitrary time such that $ftime(\omega) \leq t \leq ltime(\omega)$. It must be shown that $f(\Sigma \frown (\omega \lhd t)) = (\omega \rhd t, a, s)$.

   Let k be such that $f_k(\Sigma \lceil A_k) = (\omega_k, a, s_k)$. This k exists by definition of f. Then, for all i, either $f_i(\Sigma \lceil A_i) = (\omega_i, a_i, s_i)$, with $ltime(\omega_i) \geq ltime(\omega_k)$, or $f_i(\Sigma \lceil A_i) = \omega_i$, with $ltime(\omega_i) = \infty > ltime(\omega_k)$. For all i, $ftime(\omega_i) = ftime(\omega_k)$. Thus, for all i, $ftime(\omega_i) \leq t \leq ltime(\omega_i)$, so since f_i is consistent

   $$f_i((\Sigma \lceil A_i) \frown (\omega_i \lhd t)) = f_i((\Sigma \frown (\omega \lhd t)) \lceil A_i) = \begin{cases} ((\omega_i \rhd t), a_i, s_i) & \text{if } f_i(\Sigma \lceil A_i) = (\omega_i, a_i, s_i) \\ \omega_i \rhd t & \text{if } f_i(\Sigma \lceil A_i) = \omega_i \end{cases}$$

   Since by definition k is the smallest index such that $ltime(\omega_k) = \min_{1 \leq i \leq N}(ltime(\omega_i))$ and since $ltime(\omega_i) = ltime(\omega_i \rhd t)$, k is the smallest index such that $ltime(\omega_k \rhd t) = \min_{1 \leq i \leq N}(ltime(\omega_i \rhd t))$.

   Now, since $f_k((\Sigma \frown (\omega \lhd t)) \lceil A_i) = ((\omega_k \rhd t), a, s_k)$, the definition of f gives: $f(\Sigma \frown (\omega \lhd t)) = (\omega', a, s')$, where $\omega' \lceil A_i = \omega_i \rhd t = (\omega \rhd t) \lceil A_i$ which implies that $\omega' = \omega \rhd t$, and s' is defined as follows: $s' \lceil A_k = s_k = s \lceil A_i$, $s' \lceil A_i = g_i(((\Sigma \frown (\omega \lhd t)) \frown \omega') \lceil A_i, a) = g_i((\Sigma \frown \omega) \lceil A_i, a) = s \lceil A_i$ if $i \neq k$ and $a \in in(A_i)$, and $s' \lceil A_i = lstate((\Sigma \frown (\omega \lhd t)) \frown \omega') \lceil A_i = lstate(\Sigma \frown \omega) \lceil A_i = s \lceil A_i$ if $i \neq k$ and $a \notin acts(A_i)$. Hence, for all i, $s' \lceil A_i = s \lceil A_i$, which implies $s' = s$.

Thus, $(\omega', a, s') = (\omega \rhd t, a, s)$, which finally gives $f(\Sigma \frown (\omega \lhd t)) = (\omega \rhd t, a, s)$, as required.

In a similar fashion, it can be proved that if for arbitrary $\Sigma \in t\text{-}exec^*(A)$, $f(\Sigma) = \omega$, then, for any $ftime(\omega) \leq t < ltime(\omega)(= \infty)$, it is the case that $f(\Sigma \frown (\omega \lhd t)) = (\omega \rhd t)$.

This concludes the proof that (g, f) is a strategy defined on A. ■

The following lemma is the key lemma for showing that the strategy of Definition 4.31 is environment-free if the component strategies are environment-free. Specifically, up to a technical condition, the projection of an outcome of (g, f) onto a component A_i is an outcome of (g_i, f_i). Intuitively this means that even though the composed system uses its composed strategy to find possible outcomes, up to a technical restriction it still looks to each component as if it was using its own component strategy. The one restriction in the generality of the lemma stems from the following situation. If the system receives Zeno inputs that are not inputs to the ith component, then the ith component observes that time is blocked even though the strategy for the ith component lets time pass forever. Thus, if an outcome of the composed system is Zeno, then the following lemma only applies to the components that perform infinitely many actions in the given outcome. Note that the inputs that the ith component "sees" are either inputs to the composed system or inputs from other components of the system.

Lemma 4.33

Let $A_1, \ldots, A_N$ be compatible safe timed I/O automata and let, for each $1 \leq i \leq N$, (g_i, f_i) be a strategy defined on A_i. Let $A = A_1 \| \cdots \| A_N$ and $(g, f) = (g_1, f_1) \| \cdots \| (g_N, f_N)$.

Furthermore, let Σ be an arbitrary finite timed execution of A, $\mathcal{I}$ be an arbitrary timed environment sequence for A compatible with Σ, Σ' be an arbitrary timed execution of $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I})$, and i, with $1 \leq i \leq N$, be an arbitrary index such that if Σ' is Zeno then $|\Sigma' \upharpoonright acts(A_i)| = \infty$.

Then there exists a timed environment sequence $\mathcal{I}_i$ for A_i compatible with $\Sigma\lceil A_i$, such that $\Sigma'\lceil A_i \in \mathcal{O}_{(g_i,f_i)}(\Sigma\lceil A_i, \mathcal{I}_i)$.

Proof. Definition 4.10 implies that A is a safe timed I/O automaton. Furthermore, by Lemma 4.32, (g, f) is a strategy defined on A.

Let $R_{(g,f)}$ and $R_{(g_i,f_i)}$ be the next-relations induced by (g, f) and (g_i, f_i), respectively. Also, let $(\Sigma^n, \mathcal{I}^n)_{n\geq 0}$ be an outcome sequence of (g, f) given Σ and $\mathcal{I}$ such that $\Sigma' = \lim_{n\to\infty} \Sigma^n$. Since $(\Sigma^n)_{n\geq 0}$ forms an infinite chain ordered by t-prefix and $\Sigma^0 = \Sigma$, $\Sigma \leq_t \Sigma'$. Define $\mathcal{I}_i = t\text{-}seq(\Sigma' - \Sigma) \upharpoonright (in(A_i) \times \mathsf{T})$. Then either $\mathcal{I}_i$ is empty or $ftime(\mathcal{I}_i) \geq ltime(\Sigma) = ltime(\Sigma\lceil A_i)$. Thus, $\mathcal{I}_i$ is compatible with $\Sigma\lceil A_i$.

Let $\mathbb{N}_0 \to \mathbb{N}_0$ be the signature of a total, nondecreasing mapping m. Define $m(n)$ inductively on n such that $m(0) = 0$ and either $m(n) = m(n-1)$ or $m(n) = m(n-1) + 1$ (for $n > 0$). Simultaneously the induction defines $(\Sigma_i^0, \mathcal{I}_i^0)$ and, if $n > 0$ and $m(n) = m(n-1) + 1$, $(\Sigma_i^{m(n)}, \mathcal{I}_i^{m(n)})$. Furthermore, the same induction, proves

P1 Σ^n is finite iff $\Sigma_i^{m(n)}$ is finite.

P2 $ftime(f_i(\Sigma_i^{m(n)}).trj) \leq ltime(\Sigma^n) \leq ltime(f_i(\Sigma_i^{m(n)}).trj)$ if $\Sigma_i^{m(n)}$ is finite.

P3 $\Sigma^n\lceil A_i = \begin{cases} \Sigma_i^{m(n)} \frown (f_i(\Sigma_i^{m(n)}).trj \lhd ltime(\Sigma^n)) & \text{if } \Sigma_i^{m(n)} \text{ is finite} \\ \Sigma_i^{m(n)} & \text{otherwise} \end{cases}$

P4 $\mathcal{I}_i^{m(n)} = \begin{cases} t\text{-}seq(\Sigma' - \Sigma^n) \upharpoonright (in(A_i) \times \mathsf{T}) & \text{if } \Sigma_i^{m(n)} \text{ is finite} \\ \varepsilon & \text{otherwise} \end{cases}$

P5 If $n > 0$ and $m(n) = m(n-1) + 1$ then $((\Sigma_i^{m(n-1)}, \mathcal{I}_i^{m(n-1)}), (\Sigma_i^{m(n)}, \mathcal{I}_i^{m(n)})) \in R_{(g_i, f_i)}$.

Intuitively, $(\Sigma_i^j, \mathcal{I}_i^j)$ represents an outcome sequence of (g_i, f_i) given $\Sigma\lceil A_i$ and $\mathcal{I}_i$, and the mapping m associates each Σ^n with the first Σ_i^j such that $\Sigma^n\lceil A_i \leq_t \Sigma_i^j$.

Note that all the statements are well-defined. For P3, when $\Sigma_i^{m(n)}$ is finite, the application of the $\lhd$ operator is well-defined since (g_i, f_i) is a strategy defined on A_i and statement P2 applies; for P4, when $\Sigma_i^{m(n)}$ is finite, $\Sigma' - \Sigma^n$ is well-defined since by P1 Σ^n is finite, and furthermore $\Sigma^n \leq_t \Sigma'$ by definition. These observations are not repeated below. Finally, note that P5 is not needed in the induction step. P5 is included in the induction for convenience.

Base case $n = 0$**:**

Define: $m(0) = 0$
$\Sigma_i^{m(0)} = \Sigma^0\lceil A_i$
$\mathcal{I}_i^{m(0)} = \mathcal{I}_i$

P1 $\Sigma^0 = \Sigma$ is finite by definition. Since $\Sigma_i^{m(0)} = \Sigma^0\lceil A_i$, Lemma 4.11 implies that also $\Sigma_i^{m(0)}$ is finite.

P2 By P1, $\Sigma_i^{m(0)}$ is finite. Since $\Sigma_i^{m(0)} = \Sigma^0\lceil A_i$ and $\Sigma^0 = \Sigma$, it is necessary to prove that $ftime(f_i(\Sigma\lceil A_i).trj) \leq ltime(\Sigma) \leq ltime(f_i(\Sigma\lceil A_i).trj)$. From the fact that (g_i, f_i) is a strategy defined on A_i and Lemma 4.11 conclude that $ftime(f_i(\Sigma\lceil A_i).trj) = ltime(\Sigma)$, and since $ftime(f_i(\Sigma\lceil A_i).trj) \leq ltime(f_i(\Sigma\lceil A_i).trj)$, the result follows.

P3 First note that $\Sigma_i^{m(0)}$ is finite. As in P2, $ftime(f_i(\Sigma_i^{m(0)}).trj) = ltime(\Sigma^0)$. Thus, $\Sigma^0\lceil A_i = \Sigma_i^{m(0)} = \Sigma_i^{m(0)} \frown (f_i(\Sigma_i^{m(0)}).trj \lhd ltime(\Sigma^0))$, as required.

P4 First note that $\Sigma_i^{m(0)}$ is finite. Then $\mathcal{I}_i^{m(0)} = \mathcal{I}_i = t\text{-}seq(\Sigma' - \Sigma) \upharpoonright (in(A_i) \times \mathsf{T}) = t\text{-}seq(\Sigma' - \Sigma^0) \upharpoonright (in(A_i) \times \mathsf{T})$, as required.

P5 Vacuously satisfied.

Inductive step $n > 0$**:**

Assume P1–P5 hold for all $k < n$. Consider cases:

Case 1 Σ^{n-1} is not finite.

Since $((\Sigma^{n-1}, \mathcal{I}^{n-1}), (\Sigma^n, \mathcal{I}^n)) \in R_{(g,f)}$ and Σ^{n-1} is not finite $(\Sigma^{n-1}, \mathcal{I}^{n-1}) = (\Sigma^n, \mathcal{I}^n)$.

Define: $m(n) = m(n-1) + 1$
$\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)}$
$\mathcal{I}_i^{m(n)} = \mathcal{I}_i^{m(n-1)}$

P1 Since $\Sigma^n = \Sigma^{n-1}$ and $\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)}$ the result follows from induction hypothesis P1.

P2 Since $\Sigma^n = \Sigma^{n-1}$, Σ^n is not finite and by P1 $\Sigma_i^{m(n)}$ is not finite. Thus, P2 is vacuously satisfied.

P3 Again, since $\Sigma^n = \Sigma^{n-1}$ and $\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)}$, and none of these timed executions are finite (by P1), the result follows from induction hypothesis P3.

P4 As for P3 the result follows from an induction hypothesis; here P4.

P5 Since $\Sigma_i^{m(n-1)}$ is not finite, the result follows directly from the definition of $\Sigma_i^{m(n)}$, $\mathcal{I}_i^{m(n)}$, and $R_{(g_i,f_i)}$.

Case 2 Σ^{n-1} is finite and Σ^n is not finite.

Since $((\Sigma^{n-1}, \mathcal{I}^{n-1}), (\Sigma^n, \mathcal{I}^n)) \in R_{(g,f)}$, Σ^{n-1} is finite, and Σ^{n-1} is not finite, by Definition 4.17, $\Sigma^n = \Sigma^{n-1} \frown \omega$, where $\omega = f(\Sigma^{n-1})$, and $\mathcal{I}^n = \mathcal{I}^{n-1} = \varepsilon$.

By induction hypothesis P1, $\Sigma_i^{m(n-1)}$ is finite. Thus

$$\begin{aligned} \omega \lceil A_i &\stackrel{1}{=} f_i(\Sigma^{n-1} \lceil A_i) \\ &\stackrel{2}{=} f_i(\Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd \mathit{ltime}(\Sigma^{n-1}))) \\ &\stackrel{3}{=} f_i(\Sigma_i^{m(n-1)}) \rhd \mathit{ltime}(\Sigma^{n-1}) \end{aligned}$$

where step 1 follows from definition of (g, f) (Definition 4.31) and the fact that $\mathit{ltime}(\omega) = \infty$ (because (g, f) is a strategy), step 2 follows from induction hypothesis P3, and step 3 follows from consistency of f_i (cf. Condition 4 of Definition 4.16).

Define: $m(n) = m(n-1) + 1$
$\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)} \frown f_i(\Sigma_i^{m(n-1)})$
$\mathcal{I}_i^{m(n)} = \mathcal{I}_i^{m(n-1)}$

P1 Neither Σ^n nor $\Sigma_i^{m(n)}$ are finite.

P2 Vacuously satisfied.

P3 Since $\Sigma_i^{m(n)}$ is admissible, prove that $\Sigma^n \lceil A_i = \Sigma_i^{m(n)}$. In particular

$$\begin{aligned} \Sigma^n \lceil A_i &\stackrel{1}{=} (\Sigma^{n-1} \frown \omega) \lceil A_i \\ &\stackrel{2}{=} (\Sigma^{n-1} \lceil A_i) \frown (\omega \lceil A_i) \\ &\stackrel{3}{=} (\Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}) \lhd \mathit{ltime}(\Sigma^{n-1}))) \frown (\omega \lceil A_i) \\ &\stackrel{4}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}) \lhd \mathit{ltime}(\Sigma^{n-1})) \frown (f_i(\Sigma_i^{m(n-1)}) \rhd \mathit{ltime}(\Sigma^{n-1})) \\ &\stackrel{5}{=} \Sigma_i^{m(n-1)} \frown f_i(\Sigma_i^{m(n-1)}) \\ &\stackrel{6}{=} \Sigma_i^{m(n)} \end{aligned}$$

where steps 1, 2, and 6 are trivial, step 3 follows from induction hypothesis P3, step 4 follows from the property of $\omega\lceil A_i$ proved above, and step 5 follows from the definition of the operators $\lhd$ and $\rhd$.

P4 First note that $\Sigma_i^{m(n)}$ is not finite and $\Sigma_i^{m(n-1)}$ is finite. Then

$$\begin{aligned}
\mathcal{I}_i^{m(n)} &\stackrel{1}{=} \mathcal{I}_i^{m(n-1)} \\
&\stackrel{2}{=} \textit{t-seq}(\Sigma' - \Sigma^{n-1}) \upharpoonright (in(A_i) \times \mathsf{T}) \\
&\stackrel{3}{=} \textit{t-seq}(\Sigma^n - \Sigma^{n-1}) \upharpoonright (in(A_i) \times \mathsf{T}) \\
&\stackrel{4}{=} \textit{t-seq}(\omega) \upharpoonright (in(A_i) \times \mathsf{T}) \\
&\stackrel{5}{=} \varepsilon
\end{aligned}$$

where steps 1, 4, and 5 are trivial, step 2 follows from induction hypothesis P4, and step 3 follows from the fact that Σ^n is admissible and thus a fixpoint in $(\Sigma^k)_{k\geq 0}$.

P5 Since $\Sigma_i^{m(n-1)}$ is finite, $\mathcal{I}_i^{m(n)} = \mathcal{I}_i^{m(n-1)} = \varepsilon$ (by P4), and $\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)} \frown f_i(\Sigma_i^{m(n-1)})$, the result follows from the second case in the definition of $R_{(g_i,f_i)}$ (Definition 4.16).

Case 3 Σ^{n-1} and Σ^n are finite.

The definition of $R_{(g,f)}$ gives three cases to consider. (The first, third, and fourth cases in Definition 4.17.) Consider the first and third cases at the same time.

Case 3.1 First and third cases.

From definition of $R_{(g,f)}$ note that $\Sigma^n = \Sigma^{n-1} \frown \omega a\{s\}$, $f(\Sigma^{n-1}) = (\omega, a, s)$, and

$$\mathcal{I}^{n-1} = \begin{cases} \varepsilon & \text{or} \\ (b,t)\mathcal{I}' & \text{with } \textit{ltime}(\omega) \leq t. \end{cases}$$

In both cases $\mathcal{I}^n = \mathcal{I}^{n-1}$.

Case 3.1.1 $a \notin \textit{acts}(A_i)$

Define: $m(n) = m(n-1)$

P1 Induction hypothesis P1 and the case assumptions imply that both Σ^n and $\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)}$ are finite.

P2 Since $\Sigma_i^{m(n)}$ is finite, we must prove that $\textit{ftime}(f_i(\Sigma_i^{m(n)}).\textit{trj}) \leq \textit{ltime}(\Sigma^n) \leq \textit{ltime}(f_i(\Sigma_i^{m(n)}).\textit{trj})$.

The first inequality holds by induction hypothesis P2 and the facts that $\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)}$ and $\textit{ltime}(\Sigma^n) \geq \textit{ltime}(\Sigma^{n-1})$.

For the second inequality

$$\begin{aligned}
\textit{ltime}(\Sigma^n) &\stackrel{1}{=} \textit{ltime}(\omega) \\
&\stackrel{2}{\leq} \textit{ltime}(f_i(\Sigma^{n-1}\lceil A_i).\textit{trj}) \\
&\stackrel{3}{=} \textit{ltime}(f_i(\Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).\textit{trj} \lhd \textit{ltime}(\Sigma^{n-1}))).\textit{trj}) \\
&\stackrel{4}{=} \textit{ltime}(f_i(\Sigma_i^{m(n-1)}).\textit{trj} \rhd \textit{ltime}(\Sigma^{n-1})) \\
&\stackrel{5}{=} \textit{ltime}(f_i(\Sigma_i^{m(n-1)}).\textit{trj}) \\
&\stackrel{6}{=} \textit{ltime}(f_i(\Sigma_i^{m(n)}).\textit{trj})
\end{aligned}$$

where steps 1 and 6 is trivial, step 2 follows from definition of (g, f) (cf. Definition 4.31), step 3 follows from induction hypothesis P3, step 4 follows from consistency of f_i (cf. Condition 4 of Definition 4.16), and step 5 follows from the fact that the $\rhd$ operator preserves the limit time.

P3 First note that $\Sigma_i^{m(n)}$ is finite. Then

$$\begin{aligned}
\Sigma^n \lceil A_i &\stackrel{1}{=} (\Sigma^{n-1} \frown \omega a\{s\})\lceil A_i \\
&\stackrel{2}{=} \Sigma^{n-1}\lceil A_i \frown \omega\lceil A_i \\
&\stackrel{3}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1})) \frown \omega\lceil A_i \\
&\stackrel{4}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1})) \frown \\
&\quad (f_i(\Sigma^{n-1}\lceil A_i).trj \lhd ltime(\Sigma^n)) \\
&\stackrel{5}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1})) \frown \\
&\quad (f_i(\Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1}))).trj \lhd \\
&\quad ltime(\Sigma^n)) \\
&\stackrel{6}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1})) \frown \\
&\qquad ((f_i(\Sigma_i^{m(n-1)}).trj \rhd ltime(\Sigma^{n-1})) \lhd ltime(\Sigma^n)) \\
&\stackrel{7}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^n)) \\
&\stackrel{8}{=} \Sigma_i^{m(n)} \frown (f_i(\Sigma_i^{m(n)}).trj \lhd ltime(\Sigma^n))
\end{aligned}$$

where steps 1 and 8 are trivial, step 2 follows from the fact that $a \notin acts(A_i)$, step 3 follows from induction hypothesis P3, step 4 follows from the definition of (g, f) (cf. Definition 4.31), step 5 follows from induction hypothesis P3, step 6 follows from consistency of f_i (cf. Condition 4 of Definition 4.16), and step 7 follows from the definition of the operators and the fact that $\Sigma^{n-1} \leq_t \Sigma^n$.

P4 Again note that $\Sigma_i^{m(n)}$ is finite. Then

$$\begin{aligned}
\mathcal{I}_i^{m(n)} &\stackrel{1}{=} \mathcal{I}_i^{m(n-1)} \\
&\stackrel{2}{=} t\text{-}seq(\Sigma' - \Sigma^{n-1}) \upharpoonright (in(A_i) \times \mathsf{T}) \\
&\stackrel{3}{=} ((a, s.now) \,\hat{}\, t\text{-}seq(\Sigma' - \Sigma^n)) \upharpoonright (in(A_i) \times \mathsf{T}) \\
&\stackrel{4}{=} t\text{-}seq(\Sigma' - \Sigma^n) \upharpoonright (in(A_i) \times \mathsf{T})
\end{aligned}$$

where step 1 is trivial, step 2 follows from induction hypothesis P4, step 3 follows from definition of *t-seq* and the fact that $\Sigma^n = \Sigma^{n-1} \frown \omega a\{s\}$, and step 4 follows from the fact that $a \notin acts(A_i)$ and thus $a \notin in(A_i)$.

P5 Vacuously satisfied.

Case 3.1.2 $a \in local(A_i)$

Since by induction hypothesis P1$\Sigma_i^{m(n-1)}$ is finite,

$$\begin{aligned}
(\omega\lceil A_i, a, s\lceil A_i) &\stackrel{1}{=} f_i(\Sigma^{n-1}\lceil A_i) \\
&\stackrel{2}{=} f_i(\Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1}))) \\
&\stackrel{3}{=} f_i(\Sigma_i^{m(n-1)}) \rhd ltime(\Sigma^{n-1})
\end{aligned}$$

where step 1 follows from the definition of (g, f) (Definition 4.31) and the fact that $a \in local(A_i)$, step 2 follows from induction hypothesis P3, and step 3

follows from consistency of f_i (cf. Condition 4 of Definition 4.16).
Thus, $f_i(\Sigma_i^{m(n-1)}) = (\omega_i, a, s_i)$, where $\omega_i \rhd ltime(\Sigma^{n-1}) = \omega\lceil A_i$ and $s_i = s\lceil A_i$.

Define: $m(n) = m(n-1) + 1$

$$\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)} \frown (\omega_i a\{s_i\})$$
$$\mathcal{I}_i^{m(n)} = \mathcal{I}_i^{m(n-1)}$$

P1 Both Σ^n and $\Sigma_i^{m(n)}$ are finite.

P2 Since $\Sigma_i^{m(n)}$ is finite, we must prove that $ftime(f_i(\Sigma_i^{m(n)}).trj) \leq ltime(\Sigma^n) \leq ltime(f_i(\Sigma_i^{m(n)}).trj)$.
Since $ftime(f_i(\Sigma_i^{m(n)}).trj) = ltime(\omega_i) = ltime(\omega) = ltime(\Sigma^n)$, the first inequivalence holds.
The proof of the second inequality is similar to the one in subcase 3.1.1, only here the last step should use $ltime(f_i(\Sigma_i^{m(n-1)}).trj) \leq ltime(f_i(\Sigma_i^{m(n)}).trj)$.

P3 First note that both $\Sigma_i^{m(n-1)}$ and $\Sigma_i^{m(n)}$ are finite.

$$\begin{aligned}
\Sigma^n\lceil A_i &\stackrel{1}{=} (\Sigma^{n-1} \frown \omega a\{s\})\lceil A_i \\
&\stackrel{2}{=} \Sigma^{n-1}\lceil A_i \frown (\omega\lceil A_i)a\{s\lceil A_i\} \\
&\stackrel{3}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1})) \frown (\omega\lceil A_i)a\{s\lceil A_i\} \\
&\stackrel{4}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1})) \frown \\
&\qquad (f_i(\Sigma_i^{m(n-1)}).trj \rhd ltime(\Sigma^{n-1}))a\{s\lceil A_i\} \\
&\stackrel{5}{=} \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj)a\{s\lceil A_i\} \\
&\stackrel{6}{=} \Sigma_i^{m(n-1)} \frown \omega_i a\{s_i\} \\
&\stackrel{7}{=} \Sigma_i^{m(n)} \\
&\stackrel{8}{=} \Sigma_i^{m(n)} \frown (f_i(\Sigma_i^{m(n)}).trj \lhd ltime(\Sigma^n))
\end{aligned}$$

where steps 1, 6, and 7 are trivial, step 2 follows from the fact that $a \in acts(A_i)$, step 3 follows from induction hypothesis P3, step 4 follows from the properties of $\omega\lceil A_i$ proved at the beginning of this case, step 5 follows from definition of $\lhd$ and $\rhd$, and finally step 8 follows from the fact that $ltime(\Sigma^n) = ltime(\Sigma_i^{m(n)}) = ftime(f_i(\Sigma_i^{m(n)}).trj)$.

P4 Again, note that both $\Sigma_i^{m(n-1)}$ and $\Sigma_i^{m(n)}$ are finite.

$$\begin{aligned}
\mathcal{I}_i^{m(n)} &\stackrel{1}{=} \mathcal{I}_i^{m(n-1)} \\
&\stackrel{2}{=} t\text{-}seq(\Sigma' - \Sigma^{n-1}) \restriction (in(A_i) \times \mathsf{T}) \\
&\stackrel{3}{=} ((a, s.now)\,\hat{}\; t\text{-}seq(\Sigma' - \Sigma^n)) \restriction (in(A_i) \times \mathsf{T}) \\
&\stackrel{4}{=} t\text{-}seq(\Sigma' - \Sigma^n) \restriction (in(A_i) \times \mathsf{T})
\end{aligned}$$

where step 1 is trivial, step 2 follows from induction hypothesis P4, step 3 follows from definition of *t-seq* and the fact that $\Sigma^n = \Sigma^{n-1} \frown \omega a\{s\}$, and step 4 follows from $a \in local(A_i)$ and thus $a \notin in(A_i)$.

P5 If $\mathcal{I}_i^{m(n-1)} = \varepsilon$, the result follows from the first case in the definition of $R_{(g_i,f_i)}$ (Definition 4.17).
Now, assume $\mathcal{I}_i^{m(n-1)} = (b, t')\mathcal{I}_i'$. Then $a \neq b$ since $a \in local(A_i)$ and $b \in in(A_i)$. Now, since $\mathcal{I}_i^{m(n-1)} = t\text{-}seq(\Sigma' - \Sigma^{n-1}) \restriction (in(A_i) \times \mathsf{T})$ and $(a, s.now)$ is first on $t\text{-}seq(\Sigma' - \Sigma^{n-1})$, conclude that $s.now \leq t'$. Thus,

$ltime(\omega_i) = ltime(\omega) = s.now \leq t'$. Then the result follows from the third case in the definition of $R_{(g_i,f_i)}$.

Case 3.1.3 $a \in in(A_i)$

Let $\omega_i = f_i(\Sigma_i^{m(n-1)}).trj$.

Similar to proof P2 of subcase 3.1.1, it is easy to prove: $ftime(\omega_i) \leq ltime(\Sigma^n) \leq ltime(\omega_i)$.

Then, let $\omega_i' = \omega_i \lhd ltime(\Sigma^n)$ and $s_i' = g_i(\Sigma_i^{m(n-1)} \frown \omega_i', a)$.

Furthermore,

$$\begin{aligned}
\mathcal{I}_i^{m(n-1)} &\stackrel{1}{=} t\text{-}seq(\Sigma' - \Sigma^{n-1}) \upharpoonright (in(A_i) \times \mathsf{T}) \\
&\stackrel{2}{=} ((a, s.now) \,\hat{}\, t\text{-}seq(\Sigma' - \Sigma^{n-1})) \upharpoonright (in(A_i) \times \mathsf{T}) \\
&\stackrel{3}{=} (a, s.now) \,\hat{}\, (t\text{-}seq(\Sigma' - \Sigma^{n}) \upharpoonright (in(A_i) \times \mathsf{T}))
\end{aligned}$$

where step 1 follows from induction hypothesis P4, step 2 follows from the definition of Σ^n, and step 3 follows from the fact that $a \in in(A_i)$. Thus, in particular $\mathcal{I}_i^{m(n-1)}$ is not empty.

Define: $m(n) = m(n-1) + 1$

$\Sigma_i^{m(n)} = \Sigma_i^{m(n-1)} \frown (\omega_i' a \{s_i'\})$

$\mathcal{I}_i^{m(n)} = tail(\mathcal{I}_i^{m(n-1)})$

P1 Both Σ^n and $\Sigma_i^{m(n)}$ are finite.

P2 Similar to the proof of P2 in subcase 3.1.2.

P3 First note that both $\Sigma_i^{m(n-1)}$ and $\Sigma_i^{m(n)}$ are finite.

First,

$$\begin{aligned}
\omega \lceil A_i &\stackrel{1}{=} f_i(\Sigma^{n-1} \lceil A_i).trj \lhd ltime(\Sigma^n) \\
&\stackrel{2}{=} f_i(\Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1}))).trj \lhd ltime(\Sigma^n) \\
&\stackrel{3}{=} (f_i(\Sigma_i^{m(n-1)}).trj \rhd ltime(\Sigma^{n-1})) \lhd ltime(\Sigma^n) \\
&\stackrel{4}{=} (\omega_i \rhd ltime(\Sigma^{n-1})) \lhd ltime(\Sigma^n)
\end{aligned}$$

where step 1 follows from the definition of (g, f) (Definition 4.31), step 2 follows from induction hypothesis P3, step 3 follows from consistency of f_i (cf. Condition 4 of Definition 4.16), and step 4 follows from the definition of ω_i.

Second,

$$\begin{aligned}
s \lceil A_i &\stackrel{1}{=} g_i((\Sigma^{n-1} \frown \omega) \lceil A_i, a) \\
&\stackrel{2}{=} g_i(\Sigma^{n-1} \lceil A_i \frown \omega \lceil A_i, a) \\
&\stackrel{3}{=} g_i(\Sigma_i^{m(n-1)} \frown (\omega_i \lhd ltime(\Sigma^{n-1})) \frown \\
&\qquad\qquad ((\omega_i \rhd ltime(\Sigma^{n-1})) \lhd ltime(\Sigma^n)), a) \\
&\stackrel{4}{=} g_i(\Sigma_i^{m(n-1)} \frown (\omega_i \lhd ltime(\Sigma^n)), a) \\
&\stackrel{5}{=} g_i(\Sigma_i^{m(n-1)} \frown \omega_i', a) \\
&\stackrel{6}{=} s_i'
\end{aligned}$$

where steps 2, 5, and 6 are trivial, step 1 follows from the definition of (g, f) (Definition 4.31), step 3 follows from induction hypothesis P3, the property of $\omega\lceil A_i$ proved above, and definition of ω_i, and step 4 follows from the definition of the operators.

Finally,

$$\begin{array}{rcl} \Sigma^n\lceil A_i & \stackrel{1}{=} & \Sigma^{n-1}\lceil A_i \frown (\omega\lceil A_i)a\{s\lceil A_i\} \\ & \stackrel{2}{=} & \Sigma_i^{m(n-1)} \frown (f_i(\Sigma_i^{m(n-1)}).trj \lhd ltime(\Sigma^{n-1})) \frown (\omega\lceil A_i)a\{s\lceil A_i\} \\ & \stackrel{3}{=} & \Sigma_i^{m(n-1)} \frown (\omega_i \lhd ltime(\Sigma^{n-1})) \frown \\ & & \quad ((\omega_i \rhd ltime(\Sigma^{n-1})) \lhd ltime(\Sigma^n))a\{s_i'\} \\ & \stackrel{4}{=} & \Sigma_i^{m(n-1)} \frown \omega_i' a\{s_i'\} \\ & \stackrel{5}{=} & \Sigma_i^{m(n)} \\ & \stackrel{6}{=} & \Sigma_i^{m(n)} \frown (f_i(\Sigma_i^{m(n)}).trj \lhd ltime(\Sigma^n)) \end{array}$$

where steps 1 and 5 are trivial, step 2 follows from induction hypothesis P3, step 3 follows from the property of $\omega\lceil A_i$ proved above and the definition of ω_i, step 4 follows from the definition of the operators, and finally step 6 follows from the fact that $ftime(f_i(\Sigma_i^{m(n)}).trj) = ltime(\Sigma_i^{m(n)}) = ltime(\omega_i') = ltime(\Sigma_n)$.

P4 First note that both $\Sigma_i^{m(n-1)}$ and $\Sigma_i^{m(n)}$ are finite.

$$\begin{array}{rcl} \mathcal{I}_i^{m(n)} & \stackrel{1}{=} & tail(\mathcal{I}_i^{m(n-1)}) \\ & \stackrel{2}{=} & tail((a, s.now) \hat{} (t\text{-}seq(\Sigma' - \Sigma^n) \upharpoonright (in(A_i) \times \mathsf{T}))) \\ & \stackrel{3}{=} & t\text{-}seq(\Sigma' - \Sigma^n) \upharpoonright (in(A_i) \times \mathsf{T}) \end{array}$$

where steps 1 and 3 are trivial and step 2 follows from the property of $\mathcal{I}_i^{m(n-1)}$ proved at the beginning of this case.

P5 This follows directly from the fourth case in the definition of $R_{(g_i,f_i)}$ (Definition 4.17).

Case 3.2 Fourth case.

The definition of $R_{(g,f)}$ gives us $\Sigma^n = \Sigma^{n-1} \frown \omega' a\{s'\}$, $\mathcal{I}^{n-1} = (a,t)\mathcal{I}^n$, $f(\Sigma^{n-1}).trj = \omega$, $ltime(\omega) \geq t$, $\omega' = \omega \lhd t$, and $g(\Sigma^{n-1} \frown \omega', a) = s'$. Distinguish three subcases.

Case 3.2.1 $a \notin acts(A_i)$

Similar to subcase 3.1.1.

Case 3.2.2 $a \in local(A_i)$

This situation cannot occur since $a \in in(A)$ (cf. Definition 4.10).

Case 3.2.3 $a \in in(A_i)$

Similar to subcase 3.1.3.

This concludes the inductive definition and induction proof. Since $m(0) = 0$, $m(n)$ is either $m(n-1)+1$ or $m(n-1)$, and $(\Sigma_i^{m(0)}, \mathcal{I}_i^{m(0)})$ and $(\Sigma_i^{m(n)}, \mathcal{I}_i^{m(n)})$ are defined every time $m(n) = m(n-1)+1$, $(\Sigma_i^n, \mathcal{I}_i^n)_{0 \leq n \leq (\lim_{k \to \infty} m(k))}$ is defined. Furthermore, by the base case and the proof of P5 for every n with $m(n) = m(n-1)+1$:

- $(\Sigma_i^0, \mathcal{I}_i^0) = (\Sigma\lceil A_i, \mathcal{I}_i)$ and

- for all $0 < n \leq (\lim_{k\to\infty} m(k))$, $((\Sigma_i^{n-1}, \mathcal{I}_i^{n-1}), (\Sigma_i^n, \mathcal{I}_i^n)) \in R_{(g_i,f_i)}$.

Using the results of the induction, the lemma can now be proven. Consider cases based on $(\Sigma^n)_{n\geq 0}$.

1. One of Σ^n is not finite.

   Then, by definition of $R_{(g,f)}$, there exists a number $n' > 0$ such that $\Sigma^{n'-1}$ is finite, $\Sigma^{n'}$ is admissible, and for all $n > n'$, $\Sigma^n = \Sigma^{n'}$.

   In the induction above each $n > n'$ is handled by case 1. Since this case sets $m(n) = m(n-1)+1$, $\lim_{n\to\infty} m(n) = \infty$. Thus, $(\Sigma_i^n, \mathcal{I}_i^n)_{n\geq 0}$ is defined and this sequence then forms an outcome sequence of (g_i, f_i) given $\Sigma\lceil A_i$ and $\mathcal{I}_i$. Then

   $$\begin{aligned} \Sigma'\lceil A_i &\stackrel{1}{=} (\lim_{n\to\infty} \Sigma^n)\lceil A_i \\ &\stackrel{2}{=} \lim_{n\to\infty}(\Sigma^n\lceil A_i) \\ &\stackrel{3}{=} \lim_{n\to\infty} \Sigma_i^{m(n)} \\ &\stackrel{4}{=} \lim_{n\to\infty} \Sigma_i^n \end{aligned}$$

   where step 1 follows from the definition of Σ', step 2 follows from the continuity of the projection operator, step 3 follows from P3 since for all $n > n'$, $\Sigma^n\lceil A_i = \Sigma_i^{m(n)}$, and step 4 follows from the fact that $\lim_{n\to\infty} m(n) = \infty$.

   Thus, $\Sigma'\lceil A_i \in \mathcal{O}_{(g_i,f_i)}(\Sigma\lceil A_i, \mathcal{I}_i)$, as required.

2. All Σ^n are finite.

   (a) $\lim_{n\to\infty} m(n) \neq \infty$

   Since m is nondecreasing, there exist natural numbers n' and m' such that for all $n \geq n'$, $m(n) = m'$. Thus, $(\Sigma_i^n, \mathcal{I}_i^n)_{0\leq n\leq m'}$ is defined. In the induction above each $n > n'$ is handled by either case 3.1.1 or 3.2.1 since all other cases set $m(n) = m(n-1)+1$. Cases 3.1.1 and 3.2.1 correspond to adding an action not in $acts(A_i)$ to Σ^n. Thus, $\Sigma' - \Sigma^{n'}$ contains no actions from $acts(A_i)$ which implies that $|\Sigma' \restriction acts(A_i)| = |\Sigma^{n'} \restriction acts(A_i)| \neq \infty$ since $\Sigma^{n'}$ is finite. Thus, by hypothesis Σ' is not Zeno. Lemma 4.18 then implies that Σ' is admissible.

   Now, for all $n \geq n'$, P1 and P2 imply that $ltime(\Sigma^n) \leq ltime(f_i(\Sigma_i^{m'}).trj)$. Since Σ' is admissible, $\lim_{n\to\infty} ltime(\Sigma^n) = \infty$ which implies that $ltime(f_i(\Sigma_i^{m'}).trj) = \infty$. Thus, $f_i(\Sigma_i^{m'}) = \omega$ for some ω with $ltime(\omega) = \infty$. Furthermore, by P4, $\mathcal{I}_i^{m'} = t\text{-}seq(\Sigma' - \Sigma^{n'}) \restriction (in(A_i) \times \mathsf{T}) = \varepsilon$.

   Define $(\Sigma_i^{m'+1}, \mathcal{I}_i^{m'+1}) = (\Sigma_i^{m'} \frown \omega, \mathcal{I}_i^{m'})$. Then by definition of $R_{(g_i,f_i)}$ (Definition 4.17), $((\Sigma_i^{m'}, \mathcal{I}_i^{m'}), (\Sigma_i^{m'+1}, \mathcal{I}_i^{m'+1})) \in R_{(g_i,f_i)}$. Furthermore, for all $k > m'+1$, define $(\Sigma_i^k, \mathcal{I}_i^k) = (\Sigma_i^{k-1}, \mathcal{I}_i^{k-1})$.

   Again, clearly $((\Sigma_i^{k-1}, \mathcal{I}_i^{k-1}), (\Sigma_i^k, \mathcal{I}_i^k)) \in R_{(g_i,f_i)}$.

   Thus, $(\Sigma_i^n, \mathcal{I}_i^n)_{n\geq 0}$ is an outcome sequence of (g_i, f_i) given $\Sigma\lceil A_i$ and $\mathcal{I}_i$. Now,

$$
\begin{aligned}
\Sigma' \lceil A_i &\stackrel{1}{=} (\lim_{n\to\infty} \Sigma^n)\lceil A_i \\
&\stackrel{2}{=} \lim_{n\to\infty}(\Sigma^n \lceil A_i) \\
&\stackrel{3}{=} \lim_{n\to\infty}(\Sigma_i^{m'} \frown (f_i(\Sigma_i^{m'}).trj \lhd \mathit{ltime}(\Sigma^n))) \\
&\stackrel{4}{=} \Sigma_i^{m'} \frown \omega \\
&\stackrel{5}{=} \lim_{n\to\infty}(\Sigma_i^n)
\end{aligned}
$$

where step 1 follows from the definition of Σ', step 2 follows from the continuity of the projection operator, step 3 follows from P3, step 4 follows from the definition of ω and the fact that $\lim_{n\to\infty} \mathit{ltime}(\Sigma^n) = \infty$, and step 5 follows from the fact that $\Sigma_i^n = \Sigma_i^{m'} \frown \omega$ for all $n > m'$.

Thus, $\Sigma'\lceil A_i \in \mathcal{O}_{(g_i,f_i)}(\Sigma\lceil A_i, \mathcal{I}_i)$, as required.

(b) $\lim_{n\to\infty} m(n) = \infty$

In this situation $(\Sigma_i^n, \mathcal{I}_i^n)_{n\geq 0}$ is an outcome sequence of (g_i, f_i) given $\Sigma\lceil A_i$ and $\mathcal{I}_i$. Then

$$
\begin{aligned}
\Sigma' \lceil A_i &\stackrel{1}{=} (\lim_{n\to\infty} \Sigma^n)\lceil A_i \\
&\stackrel{2}{=} \lim_{n\to\infty}(\Sigma^n \lceil A_i) \\
&\stackrel{3}{=} \lim_{n\to\infty}(\Sigma_i^{m(n)} \frown (f_i(\Sigma_i^{m(n)}).trj \lhd \mathit{ltime}(\Sigma^n))) \\
&\stackrel{4}{=} \lim_{n\to\infty}(\Sigma_i^{m(n)}) \\
&\stackrel{5}{=} \lim_{n\to\infty}(\Sigma_i^n)
\end{aligned}
$$

where step 1 follows from the definition of Σ', step 2 follows from the continuity of the projection operator, step 3 follows from P3, step 4 follows from the fact that for all n, $\Sigma_i^{m(n)} \leq_t \Sigma_i^{m(n)} \frown (f_i(\Sigma_i^{m(n)}).trj \lhd \mathit{ltime}(\Sigma^n)) \leq_t \Sigma_i^{m(n)+1}$ (this follows directly from the definition of $\Sigma_i^{m(n)}$ for all n where $m(n) = m(n-1)+1$), and finally step 5 follows from the fact that $\lim_{n\to\infty} m(n) = \infty$.

Thus, $\Sigma'\lceil A_i \in \mathcal{O}_{(g_i,f_i)}(\Sigma\lceil A_i, \mathcal{I}_i)$, as required.

This concludes the proof. ■

Lemma 4.34

Let $(A_1, L_1), \ldots, (A_N, L_N)$ be compatible live timed I/O automata and let, for each $1 \leq i \leq N$, (g_i, f_i) be an environment-free strategy for $(A_i, L_i \cup t\text{-}exec^{Zt}(A_i))$. Furthermore, let $(A, L) = (A_1, L_1)\| \cdots \|(A_N, L_N)$. Then $(g, f) = (g_1, f_1)\| \cdots \|(g_N, f_N)$ is an environment-free strategy for $(A, L \cup t\text{-}exec^{Zt}(A))$.

Proof. Definition 4.22 given the following proof obligations.

1. A is a safe timed I/O automaton,
2. $L \cup t\text{-}exec^{Zt}(A) \subseteq t\text{-}exec(A)$,
3. $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq L \cup t\text{-}exec^{Zt}(A)$, for all $\Sigma \in t\text{-}exec^*(A)$ and all timed environment sequences $\mathcal{I}$ for A that are compatible with Σ, and

4. (g, f) is Zeno-tolerant.

Consider the points one at a time.

1. Definition 4.25 along with Definition 4.10 directly implies that A is a safe timed I/O automaton.

2. By Definition 4.25, $L = \{\Sigma \in \textit{t-exec}(A) \mid \Sigma\lceil A_1 \in L_1, \ldots, \Sigma\lceil A_N \in L_N\}$. Thus, Lemma 4.11 implies $L \subseteq \textit{t-exec}(A)$. Finally, since $\textit{t-exec}^{Zt}(A) \subseteq \textit{t-exec}(A)$, the result follows.

3. Let $\Sigma \in \textit{t-exec}^*(A)$ be an arbitrary finite timed execution of A and $\mathcal{I}$ be an arbitrary timed environment sequence for A that is compatible with Σ. Note, since (g_i, f_i) is an environment-free strategy for $(A_i, L_i \cup \textit{t-exec}^{Zt}(A_i))$, (g_i, f_i) is, by Definition 4.22, a Zeno-tolerant strategy defined on A_i. Let Σ' be an arbitrary element of the outcome $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I})$. By Lemma 4.18, Σ is either Zeno or admissible.

    - Assume Σ' is Zeno.
      By Lemma 4.18, Σ' contains infinitely many actions ($|\Sigma' \restriction \textit{acts}(A)| = \infty$). Assume Σ' is not Zeno-tolerant ($\Sigma' \in \textit{t-exec}^{Zt}(A)$). Then $|\Sigma' \restriction \textit{local}(A)| = \infty$. Since each locally-controlled action in Σ' belongs to the locally-controlled actions of one of the component automata of A, and there are only finitely many such components, there exists an i such that $|\Sigma' \restriction \textit{local}(A_i)| = \infty$ which also implies $|(\Sigma'\lceil A_i) \restriction \textit{local}(A_i)| = \infty$. Lemma 4.33 now implies the existence of a timed sequence $\mathcal{I}_i$ over $\textit{in}(A_i)$ compatible with $\Sigma\lceil A_i$ such that $\Sigma'\lceil A_i \in \mathcal{O}_{(g_i,f_i)}(\Sigma\lceil A_i, \mathcal{I}_i)$. Since $\Sigma'\lceil A_i$ is Zeno (by Lemma 4.11) and $|(\Sigma'\lceil A_i) \restriction \textit{local}(A_i)| = \infty$, there is a contradiction to the fact that (g_i, f_i) is Zeno-tolerant.
      Thus, $\Sigma' \in \textit{t-exec}^{Zt}(A)$. That suffices.
    - Assume Σ' is admissible.
      By Lemma 4.33, for each $1 \leq i \leq N$ there exists a timed sequence $\mathcal{I}_i$ over $\textit{in}(A_i)$ compatible with $\Sigma\lceil A_i$, such that $\Sigma'\lceil A_i \in \mathcal{O}_{(g_i,f_i)}(\Sigma\lceil A_i, \mathcal{I}_i)$. By Lemma 4.11, $\Sigma'\lceil A_i$ is admissible. Now the fact that (g_i, f_i) is an environment-free for the pair $(A_i, L_i \cup \textit{t-exec}^{Zt}(A_i))$ implies that $\Sigma'\lceil A_i \in L_i$. This implies, by Definition 4.25, that $\Sigma' \in L$. That suffices.

4. To prove that (g, f) is Zeno-tolerant (cf. Definition 4.21), it suffices to note that the previous case says that $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq L \cup \textit{t-exec}^{Zt}(A)$, where $L \subseteq \textit{t-exec}^{\infty}(A)$. ■

The main result, closure of the parallel composition operator, can now be proven.

Proposition 4.35 (Closure of parallel composition)

Let $(A_1, L_1), \ldots, (A_N, L_N)$ be compatible live timed I/O automata. Then the parallel composition $(A_1, L_1)\|\ldots\|(A_N, L_N)$ is a live timed I/O automaton.

Proof. Let $(A, L) = (A_1, L_1)\|\cdots\|(A_N, L_N)$. Definition 4.10 implies that A is a safe timed I/O automaton. Furthermore, since $L_i \subseteq \textit{t-exec}^\infty(A_i)$, Lemma 4.11 and Definition 4.25 show that $L \subseteq \textit{t-exec}^\infty(A)$.

For each $1 \leq i \leq N$, let (g_i, f_i) be an environment-free strategy for $(A_i, L_i \cup \textit{t-exec}^{Zt}(A_i))$. By Lemma 4.34 the strategy $(g, f) = (g_1, f_1)\|\cdots\|(g_N, f_N)$ is an environment-free strategy for $(A, L \cup \textit{t-exec}^{Zt}(A))$. Therefore, the pair $(A, L \cup \textit{t-exec}^{Zt}(A))$ is environment-free. By Definition 4.23 of live timed I/O automata, the result now follows. ■

4.5 Preorder Relations for Live Timed I/O Automata

For safe timed I/O automata there are several ways of defining a timed trace preorder that depend upon which kinds of traces are being considered. A naive choice would be to consider all the timed traces of a safe timed I/O automaton; however, one might not be interested in, e.g., the Zeno timed traces of a system. For the live preorder, on the other hand, there is a unique natural choice.

Definition 4.36 (Timed trace preorders)

Given two live timed I/O automata (A_1, L_1) and (A_2, L_2) such that $\textit{esig}(A_1) = \textit{esig}(A_2)$ define the following preorders:

Safe:	$(A_1, L_1) \sqsubseteq_{\mathrm{St}} (A_2, L_2)$	iff	$\textit{t-traces}(A_1) \subseteq \textit{t-traces}(A_2)$.
Safe-finite:	$(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{*} (A_2, L_2)$	iff	$\textit{t-traces}^*(A_1) \subseteq \textit{t-traces}^*(A_2)$.
Safe-admissible:	$(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{\infty} (A_2, L_2)$	iff	$\textit{t-traces}^\infty(A_1) \subseteq \textit{t-traces}^\infty(A_2)$.
Safe-non-Zeno:	$(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{\mathrm{nz}} (A_2, L_2)$	iff	$(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{*} (A_2, L_2)$ and $(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{\infty} (A_2, L_2)$.
Live:	$(A_1, L_1) \sqsubseteq_{\mathrm{Lt}} (A_2, L_2)$	iff	$\textit{t-traces}(L_1) \subseteq \textit{t-traces}(L_2)$. ■

The safe-non-Zeno preorder is the relation that is used in [VL92]. This preorder is used in [VL92] instead of the more natural safe-admissible preorder since finite timed traces are need for substitutivity of a sequential composition operator.

It is interesting to note that the live preorder implies the safe preorder whenever the involved safe timed I/O automata have *timed finite internal nondeterminism*. On the other hand, if the involved safe timed I/O automata do not have timed finite internal nondeterminism, then the live preorder only implies finite timed trace inclusion. Essentially, timed finite internal nondeterminism requires that a timed automaton has a finite internal branching structure. In particular, a finite timed trace can lead to at most finitely many states.

Definition 4.37 (Timed finite internal nondeterminism)

A timed automaton A has *timed finite internal nondeterminism* (t-FIN) iff, for each trace $\gamma \in \textit{t-traces}^*(A)$, the set $\{\textit{lstate}(\Sigma) \mid \textit{t-trace}(\Sigma) = \gamma\}$ is finite. ■

Proposition 4.38

Let (A_1, L_1) and (A_2, L_2) be two live timed I/O automata with $\textit{vsig}(A_1) = \textit{vsig}(A_2)$.

1. *If* $(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{\infty} (A_2, L_2)$ *then* $(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{*} (A_2, L_2)$.
2. *If* A_2 *has t-FIN and* $(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{*} (A_2, L_2)$ *then* $(A_1, L_1) \sqsubseteq_{\mathrm{St}} (A_2, L_2)$.
3. *If* $(A_1, L_1) \sqsubseteq_{\mathrm{Lt}} (A_2, L_2)$ *then* $(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{*} (A_2, L_2)$.

Proof.

1. Let γ be a finite timed trace of A_1. By definition of timed trace, there is a timed execution Σ_1 of A_1 such that $t\text{-}trace(\Sigma_1) = \gamma$. By definition of live timed I/O automaton there exists an admissible timed execution Σ_1' of A_1 such that $\Sigma_1 \leq_t \Sigma_1'$ and $t\text{-}trace(\Sigma_1') \in L_1$ (just apply any environment-free strategy for (A_1, L_1) to Σ_1 and to an admissible timed environment sequence for A compatible with Σ_1). By definition of live timed I/O automaton, Σ_1' is a timed execution of A_1. Since $(A_1, L_1) \sqsubseteq_{\mathrm{St}}^{\infty} (A_2, L_2)$, there exists a timed execution Σ_2' of A_2 such that $t\text{-}trace(\Sigma_1') = t\text{-}trace(\Sigma_2')$. Since the set of timed executions of a timed I/O automaton is closed under t-prefix, there is a t-prefix Σ_2 of Σ_2' such that Σ_2 is a timed execution of A_2 and $t\text{-}trace(\Sigma_2) = \gamma$, i.e., γ is a timed trace of A_2.
2. This is a standard result that appears in [LV91].
3. Let γ be a finite timed trace of A_1. By definition of timed trace, there is a timed execution Σ_1 of A_1 such that $t\text{-}trace(\Sigma_1) = \gamma$. By definition of live timed I/O automaton there exists a timed execution Σ_1' of A_1 such that $\Sigma_1 \leq_t \Sigma_1'$ and $t\text{-}trace(\Sigma_1') \in L_1$. Since $(A_1, L_1) \sqsubseteq_{\mathrm{Lt}} (A_2, L_2)$, there exists a timed execution Σ_2' of L_2 such that $t\text{-}trace(\Sigma_1') = t\text{-}trace(\Sigma_2')$. By definition of timed live I/O automaton, Σ_2' is a timed execution of A_2, and, since the set of timed executions of a timed automaton is closed under t-prefix, there is a t-prefix Σ_2 of Σ_2' such that Σ_2 is a timed execution of A_2 and $t\text{-}trace(\Sigma_2) = \gamma$, i.e., γ is a timed trace of A_2. ■

The important property of the safe and live preorders is that they are substitutive for the operators of Section 4.4. In the case of the parallel composition operator, this means that an implementation of a system made up of several parallel components can be obtained by implementing each component separately.

Theorem 4.39 (Substitutivity)
Let $(A_i, L_i), (A_i', L_i')$, $i = 1, \ldots, N$ *be live timed I/O automata, and let* $\sqsubseteq_X$ *be one relation among* $\sqsubseteq_{\mathrm{St}}$, $\sqsubseteq_{\mathrm{St}}^{*}$, $\sqsubseteq_{\mathrm{St}}^{\infty}$, $\sqsubseteq_{\mathrm{St}}^{\mathrm{nz}}$ *and* $\sqsubseteq_{\mathrm{Lt}}$. *If, for each* i, $(A_i, L_i) \sqsubseteq_X (A_i', L_i')$, *then*

1. *if* $(A_1, L_1), \ldots, (A_N, L_N)$ *are compatible and* $(A_1', L_1'), \ldots, (A_N', L_N')$ *are compatible then* $(A_1, L_1) \| \cdots \| (A_N, L_N) \sqsubseteq_X (A_1', L_1') \| \cdots \| (A_N', L_N')$.
2. *if* $\Lambda \subseteq local(A_1)$ *and* $\Lambda \subseteq local(A_1')$ *then* $(A_1, L_1) \setminus \Lambda \sqsubseteq_X (A_1', L_1') \setminus \Lambda$

3. if ρ is a mapping applicable to both A_1 and A'_1 then
$\rho((A_1, L_1)) \sqsubseteq_X \rho((A'_1, L'_1))$

Proof. The substitutivity results are a direct consequence of Lemmas 4.11, 4.13, and 4.15, and the observation, analogous to the one of the untimed model, that parallel composition, hiding and renaming of timed execution sets preserve timed trace equivalence. ∎

4.6 Comparison with Other Timed Models

This section compares our timed model with the work of [AL91b, MMT91, VL92].

The formalism that is used in [AL91b] is the Temporal Logic of Actions (TLA) [Lam91] extended with a new variable *now* that models time. A specification S consists of the conjunction of three formulas $\mathit{Init} \wedge \Pi \wedge L$ where Init represents the initial configurations of S, Π is a *safety property*, and L is a *liveness property*. The subformula $\mathit{Init} \wedge \Pi$ corresponds to our safe timed I/O automata, while the subformula L corresponds to our timed liveness conditions. In [AL91b] L can also be satisfied by finite or Zeno executions or by executions that do not satisfy $\mathit{Init} \wedge \Pi$. The formula L is a liveness condition for $\mathit{Init} \wedge \Pi$ based on our definition iff the pair $(\mathit{Init} \wedge \Pi, L)$ is machine-closed based on the definition in [AL91b].

There is a special formula NZ in [AL91b] that is used to express non-Zenoness, i.e., that time advances forever. Time blocking or Zeno behaviors are undesirable in [AL91b] as well as in our model; however, it is possible for the safety part of a specification to describe systems for which time cannot advance past a given upper bound whenever a particular state is reached. Such a situation is eliminated in [AL91b] by requiring the pair (Π, NZ) to be machine-closed. In our model, on the other hand, the same situation is eliminated by the fact that system-Zeno executions are not allowed in the liveness part of a live timed I/O automaton and that a live timed I/O automaton is machine-closed by definition.

A major difference between our notion of environment-freedom and the notion of receptiveness of [AL91b] is in the role of time: in our model no one is allowed to have control over time; in [AL91b] either the system or its environment must have control over time. We believe that it is more reasonable to assume that no one has control over time, and thus consider our model easier to understand.

The model of [MMT91] is an extension to the timed model of the I/O automaton model of [LT87]. The locally-controlled actions of an automaton are partitioned into classes, each one of which is associated with a lower bound (possibly 0 but not ∞) and an upper bound (possibly ∞ but not 0). Actions from one class with lower bound c_1 and upper bound c_2 must stay enabled for at least c_1 time units before one of them can be performed, and cannot stay enabled more that c_2 time units without any one of them being performed.

An automaton M of [MMT91] can be represented in our model as a pair (A, L) where A is a safe timed I/O automaton with a transition relation that satisfies all the timing constraints of M, and L is the set of all admissible executions of A. It is easy to check that (A, L) is environment-free and that admissible timed trace inclusion in [MMT91] coincides with live trace inclusion in our model. However, there are liveness conditions that can be represented in our model but cannot be represented naturally in the model of [MMT91].

The work in [VL92] does not deal with general liveness properties, and argues that finite and admissible timed traces inclusion is generally sufficient to express a useful notion of implementation whenever time is involved. The work in [SLL93], however, has shown that liveness is useful even in a timed model. In general, the automata of [VL92] are not receptive, however, in order to avoid trivial implementations, [VL92] assumes some form of I/O distinction and some form of receptiveness at the lower level of implementation. There is a very close connection between the technical definitions of *I/O feasibility* and *strong I/O feasibility* of [VL92] and our notion of environment-freedom. It is possible to represent each timed I/O automaton A of [VL92] with the pair (A, L) where L is the set of admissible executions of A. The notion of I/O feasibility of [VL92], which requires each finite timed execution of A to be extendible to an admissible timed execution of A using locally-controlled actions only, is stronger than requiring that L is a liveness condition for A and weaker than requiring that (A, L) is a live timed I/O automaton. In order to have closure under parallel composition, [VL92] introduces a stronger requirement on I/O automata called strong I/O feasibility. Strong I/O feasibility adds to I/O feasibility the requirement that the safe part of an I/O automaton A does not exhibit any system-Zeno execution. However, environment-freedom, which is weaker than strong I/O feasibility since the safe part of a live timed I/O automaton is allowed to exhibit system-Zeno behaviors, is sufficient to guarantee closure under parallel composition and hence substitutivity.

5 Embedding the Untimed Model in the Timed Model

The untimed model, presented in Section 3, is used to specify systems where the amount of time that passes between actions is considered unimportant. Many problems in distributed computing can be stated and solved using this model. However, it is not possible to state anything about, e.g., response times in the untimed model. It is implicitly assumed that the final implementation on a physical machine is "fast enough" for practical use.

An untimed system can be thought of as a timed system that allows arbitrary time-passage. This indicates that the timed model is, in some sense, more general than the untimed model, and that one could use the timed model in situations where one would usually use the untimed model. However, the timed model is more complicated than the untimed model due to the time-passage action, the *now* component, etc. Furthermore, it does not seem natural to be required to deal with time, when the problem to be solved does not mention time.

Thus, one would like to work in the untimed model as much as possible and only switch to the timed model when it is needed. Sometimes, however, an algorithm that uses time implements a specification that does not use time. For example, [LLS93] shows how an untimed specification (of the at-most-once message delivery problem) is implemented by a system that assumes upper time bounds on certain process steps and channel delays. Fischer's mutual exclusion algorithm [Fis85, AL91b] is another such example. Figure 1 depicts the stepwise development one would use for an implementation proof like the one in [LLS93]. The stepwise

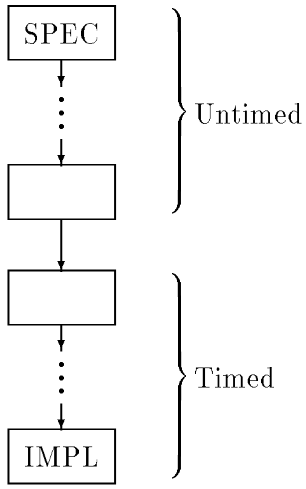


Figure 1: A stepwise development from an untimed specification to a timed implementation.

development in Figure 1, however, raises the issue of what it means to implement an untimed specification with a timed implementation. Our approach to this issue is to convert the untimed systems in the stepwise development to timed systems by applying a *patient* operator that adds arbitrary time-passage steps. The patient operator we use is similar to the one of [NS92, VL92]. To complement the *patient* operator, this section proves the *Embedding Theorem* which states that a concrete level implements an abstract level in the untimed model if and only if the *patient* version of the concrete level implements the *patient* version of the abstract level in the timed model. Thus, the first part of the stepwise development of Figure 1 can be carried out entirely in the simpler untimed model, and the last part in the timed model. In the intermediate development step which goes from untimed to timed, one must prove that the timed level implements the *patient* version of the untimed level. The embedding theorem can then be applied to show that the implementation IMPL implements the *patient* version of the specification SPEC.

Definition 5.1 (Patient operator on safe I/O automata)

Let A be a safe (untimed) I/O automaton where $\nu \notin acts(A)$. Then define $patient(A)$ to be the safe timed I/O automaton with

- $states(patient(A)) = states(A) \times \mathsf{T}$

  If $s = (s', t)$ is a state of $patient(A)$, we let $s.basic$ denote s'.

- $start(patient(A)) = start(A) \times \{0\}$

- $now_{patient(A)}((s, t)) = t$

- $ext(patient(A)) = ext(A) \cup \{\nu\}$
- $in(patient(A)) = in(A)$
- $out(patient(A)) = out(A)$
- $int(patient(A)) = int(A)$
- $steps(patient(A))$ consists of the steps
  - $\{((s,t), a, (s',t)) \mid (s,a,s') \in steps(A)\}$
  - $\{((s,t), \nu, (s,t')) \mid t' > t\}$ ■

The following trivial lemma states that the basic state of a *patient* automaton does not change during time-passage in a timed execution.

Lemma 5.2

Let A be a safe I/O automaton with $\nu \notin acts(A)$ and let $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ be a timed execution of $patient(A)$. Then, for all i and all $s, s' \in rng(\omega_i)$, $s.basic = s'.basic$. ■

In order to state what it means to apply the *patient* operator to a live I/O automaton, the following auxiliary definition of what it means to *untime* a timed execution is needed. Let A be a safe I/O automaton with $\nu \notin acts(A)$ and let $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ be a timed execution of $patient(A)$. Then define

$$untime(\Sigma) = (fstate(\omega_0).basic) a_1 (fstate(\omega_1).basic) a_2 (fstate(\omega_2).basic) \cdots$$

Similarly, let $\gamma = ((a_1,t_1)(a_2,t_2)\cdots, t)$ be a timed trace of $patient(A)$. Then define

$$untime(\gamma) = a_1 a_2 \cdots$$

Lemma 5.3

Let A be a safe I/O automaton with $\nu \notin acts(A)$. Then $\Sigma \in t\text{-}exec(patient(A))$ iff $untime(\Sigma) \in exec(A)$. Furthermore, if Σ is finite, then $untime(\Sigma)$ is finite.

Proof. The proof of this lemma is trivial using Lemma 5.2, Definition 5.1, and the definition of *untime*. ■

The patient operator can now be extended to live I/O automata. For any live I/O automaton (A, L), the patient live I/O automaton of (A, L) should be the live timed I/O automaton whose safety part is $patient(A)$ and whose liveness part consists of all those admissible executions that, when made *untime*d, are in L. Thus, the liveness condition of the patient live I/O automaton allows time to pass arbitrarily, as long as the liveness prescribed by L is satisfied sooner or later. This is formalized in the following definition.

Definition 5.4 (Patient operator on live I/O automaton)

Let (A, L) be a live I/O automaton with $\nu \notin acts(A)$. Then, define $patient_A(L) = \{\Sigma \in t\text{-}exec^{\infty}(patient(A)) \mid untime(\Sigma) \in L\}$, and define $patient(A, L)$, the *patient live I/O automaton* of (A, L), to be the pair $(patient(A), patient_A(L))$. ■

One must prove that for any live I/O automaton (A, L), $patient(A, L)$ is a live timed I/O automaton. This means showing the existence of an environment-free strategy for the pair $(patient(A), patient_A(L) \cup t\text{-}exec^{Zt}(patient(A)))$. This is accomplished by defining the *patient strategy* of an (untimed) strategy (g, f) defined on A, and showing that the patient strategy of (g, f) is environment-free for $(A_p, L_p \cup t\text{-}exec^{Zt}(A_p))$, where $(A_p, L_p) = patient(A, L)$, if (g, f) is environment-free for (A, L). To ensure that the patient strategy of (g, f) is Zeno-tolerant, which is required for environment-freedom, the patient strategy of (g, f) insists on letting time pass some fixed positive amount of time δ before making a local step.

To formalize this idea, the following definition is needed.

Definition 5.5

For any safe timed I/O automaton A and any finite timed execution Σ of A, define $lloctime(\Sigma)$ to be the time of occurrence of the last locally-controlled action in Σ, or 0 if no such action exists. Formally, let $\Sigma = \omega_0 a_1 \omega_1 \cdots a_n \omega_n$. If $a_1, \ldots, a_n \notin local(A)$, then define $lloctime(\Sigma) = 0$; otherwise, define $lloctime(\Sigma) = ftime(\omega_k)$ where $a_k \in local(A)$ and $a_{k+1}, \ldots, a_n \notin local(A)$. ■

Definition 5.6 (Patient strategy)

Let A be a safe I/O automaton with $\nu \notin acts(A)$ and (g, f) be an (untimed) strategy defined on A. Furthermore, let $A_p = patient(A)$. Then define the *patient strategy* of (g, f) with respect to some positive real number δ, written $patient_\delta(g, f)$, to be the pair of functions

$$g_p : t\text{-}exec^*(A_p) \times in(A_p) \rightarrow states(A_p)$$

$$f_p : t\text{-}exec^*(A_p) \rightarrow (traj(A_p) \times local(A_p) \times states(A_p)) \cup traj(A_p)$$

defined in the following way

$$g_p(\Sigma, a) \triangleq (g(untime(\Sigma), a), ltime(\Sigma))$$

$$f_p(\Sigma) \triangleq \begin{cases} (\omega, a, s) & \text{if } f(untime(\Sigma)) = (a, s.basic),\ s.now = ltime(\omega) \text{ and} \\ & \quad dom(\omega) = [ltime(\Sigma), max(ltime(\Sigma), lloctime(\Sigma) + \delta)] \\ & \quad rng(\omega) = \{(lstate(\Sigma).basic, t) \mid t \in dom(\omega)\} \\ \omega & \text{if } f(untime(\Sigma)) = \bot \text{ and} \\ & \quad dom(\omega) = [ltime(\Sigma), \infty] \\ & \quad rng(\omega) = \{(lstate(\Sigma).basic, t) \mid t \in dom(\omega)\} \end{cases}$$

■

For finite timed executions Σ of A_p, Lemma 5.3 implies that $untime(\Sigma)$ is a finite execution of A. Also, by Definition 5.1, A and A_p have the same input, output, and internal actions. Thus, in the definition of (g_p, f_p), the domains and ranges of g and f are compatible with the usage of g and f.

The following lemma states that the patient strategy is indeed a strategy.

Lemma 5.7

Let A be a safe I/O automaton with $\nu \notin acts(A)$, (g, f) an (untimed) strategy defined on A, and δ any positive real number. Then $patient_\delta(g, f)$ is a (timed) strategy defined on $patient(A)$.

Proof. Let $(g_p, f_p) = patient_\delta(g, f)$ and $A_p = patient(A)$. To verify that (g_p, f_p) is a (timed) strategy defined on A_p check the four conditions of Definition 4.16.

1. For the first condition, which deals with g_p, let, for arbitrary $\Sigma \in t\text{-}exec^*(A_p)$ and $a \in in(A_p)$, $g_p(\Sigma, a) = s = (s', t)$. By the definition of g_p and the fact that (g, f) is a strategy defined on A (cf. Definition 3.11), $(lstate(untime(\Sigma)), a, s') \in steps(A)$ which, by definition of *untime*, Lemma 5.2, and the fact that Σ is finite is the same as $(lstate(\Sigma).basic, a, s') \in steps(A)$. Finally, Definition 5.1 and the fact that $t = ltime(\Sigma)$ gives $(lstate(\Sigma), a, s) \in steps(A_p)$, which suffices.

2. For the second condition let $f_p(\Sigma) = (\omega, a, s)$. Similar to the first condition, it is easy to see that $(lstate(\omega), a, s)$ is a step of A_p. Then by the definition of ω and the fact that A_p allows time to pass arbitrarily, $\omega a\{s\}$ is a timed execution fragment of A_p and $fstate(\omega) = lstate(\Sigma)$. Thus, $\Sigma \frown \omega a\{s\} \in t\text{-}exec^*(A_p)$ as required.

3. The argument parallels that for Condition 2.

4. Finally, the fourth condition, dealing with consistence of f_p, is considered.

   (a) Assume $f_p(\Sigma) = (\omega, a, s)$ and let t be a time such that $ftime(\omega) \leq t \leq ltime(\omega)$. By definition of f_p we have $f(untime(\Sigma)) = (a, s.basic)$, $s.now = ltime(\omega)$, and $dom(\omega) = [ltime(\Sigma), max(ltime(\Sigma), lloctime(\Sigma) + \delta)]$ and $rng(\omega) = \{(lstate(\Sigma).basic, t') \mid t' \in dom(\omega)\}$.
   By definition of *untime*, we have $untime(\Sigma \frown (\omega \lhd t)) = untime(\Sigma)$, which implies $f(untime(\Sigma \frown (\omega \lhd t))) = f(untime(\Sigma))$.
   Thus, $f_p(\Sigma \frown (\omega \lhd t)) = (\omega', a, s')$ with $f(untime(\Sigma \frown (\omega \lhd t))) = (a, s'.basic) = (a, s.basic)$, $s'.now = ltime(\omega')$, and $dom(\omega') = [ltime(\Sigma \frown (\omega \lhd t)), max(ltime(\Sigma \frown (\omega \lhd t)), lloctime(\Sigma \frown (\omega \lhd t)) + \delta)]$ and $rng(\omega') = \{(lstate(\Sigma \frown (\omega \lhd t)).basic, t') \mid t' \in dom(\omega')\}$.
   Now, by Lemma 5.2 $lstate(\Sigma \frown (\omega \lhd t)).basic = lstate(\Sigma).basic$, and furthermore we have, by the definitions of $\lhd$ and *lloctime*, that $ltime(\Sigma \frown (\omega \lhd t)) = t$ and $lloctime(\Sigma \frown (\omega \lhd t)) = lloctime(\Sigma)$

Assume $ltime(\Sigma) > lloctime(\Sigma) + \delta$. Then $ftime(\omega) = ltime(\omega) = t = ltime(\Sigma)$, and $max(ltime(\Sigma \frown (\omega \lhd t)), lloctime(\Sigma \frown (\omega \lhd t)) + \delta) = max(ltime(\Sigma), lloctime(\Sigma) + \delta)$. Then assume $ltime(\Sigma) \leq lloctime(\Sigma) + \delta$. Then $t \leq ltime(\omega) = lloctime(\Sigma) + \delta$, and again we have $max(ltime(\Sigma \frown (\omega \lhd t)), lloctime(\Sigma \frown (\omega \lhd t)) + \delta) = max(t, lloctime(\Sigma) + \delta) = lloctime(\Sigma) + \delta = max(ltime(\Sigma), lloctime(\Sigma) + \delta)$.

Thus, we have $dom(\omega') = [ltime(\Sigma \frown (\omega \lhd t)), max(ltime(\Sigma \frown (\omega \lhd t)), lloctime(\Sigma \frown (\omega \lhd t)) + \delta)] = [t, max(ltime(\Sigma), lloctime(\Sigma) + \delta)] = dom(\omega \rhd t)$ and $rng(\omega') = \{(lstate(\Sigma \frown (\omega \lhd t)).basic, t') \mid t' \in dom(\omega)\} = \{(lstate(\Sigma).basic, t') \mid t' \in dom(\omega \rhd t)\} = rng(\omega \rhd t)$. Therefore, $\omega' = \omega \rhd t$ and $s'.now = ltime(\omega') = ltime(\omega)$ such that $s' = s$.

Hence, finally conclude that $f_p(\Sigma \frown (\omega \lhd t)) = ((\omega \rhd t), a, s)$, as required.

(b) Assume $f_p(\Sigma) = \omega$. This case is handled similarly to the previous case.

Thus, (g_p, f_p) is a strategy defined on A_p. ■

The proof that for any environment-free (untimed) strategy (g, f) for a live I/O automaton (A, L), and any positive δ, the patient strategy $patient_\delta(g, f)$ is an environment-free (timed) strategy for $(A_p, L_p \cup t\text{-}exec^{Zt}(A_p))$, where $(A_p, L_p) = patient(A, L)$, uses two technical lemmas. The first of these lemmas states that if Σ' is an *admissible* timed execution of an outcome of $patient_\delta(g, f)$, then $untime(\Sigma')$ is an outcome of (g, f). This expresses the intuitive idea that the only significant difference between (g, f) and $patient(g, f)$ is due to time-passage. The second lemma states that the difference in the time of occurrence of any two *locally-controlled* actions in a timed execution of an outcome of $patient_\delta(g, f)$, is at least δ. This is, of course, due to the fact that $patient_\delta(g, f)$ insists on letting time pass for at least δ time units between local steps.

Lemma 5.8

Let A be a safe I/O automaton with $\nu \notin acts(A)$ and let (g, f) be an (untimed) strategy defined on A. Let $A_p = patient(A)$ and $(g_p, f_p) = patient_\delta(g, f)$ for some arbitrary positive real number δ. Then, for all $\Sigma \in t\text{-}exec^*(A_p)$, all timed environment sequences $\mathcal{I}_p$ for A_p compatible with Σ, and all admissible $\Sigma' \in \mathcal{O}_{(g_p, f_p)}(\Sigma, \mathcal{I}_p)$, there exists an environment sequence $\mathcal{I}$ for A such that $untime(\Sigma') = \mathcal{O}_{(g,f)}(untime(\Sigma), \mathcal{I})$.

Proof. First note that by Lemma 5.7, (g_p, f_p) is a strategy defined on A_p.

Let $\Sigma \in t\text{-}exec^*(A_p)$ be an arbitrary finite timed execution of A, $\mathcal{I}_p$ an arbitrary timed environment sequence for A_p compatible with Σ, and Σ' be an arbitrary admissible timed execution of the outcome $\mathcal{O}_{(g_p, f_p)}(\Sigma, \mathcal{I}_p)$. Let $R_{(g_p, f_p)}$ be the next-relation induced by (g_p, f_p) and $R_{(g,f)}$ the next-function induced by (g, f). Also, let $(\Sigma^n, \mathcal{I}_p^n)_{n \geq 0}$ be an outcome sequence of (g_p, f_p) given Σ and $\mathcal{I}_p$ such that $\Sigma' = \lim_{n \to \infty} \Sigma^n$.

Let $\mathbb{N}_0 \to \mathbb{N}_0$ be the signature of a total, nondecreasing mapping m. Define $m(n)$ inductively on n. Furthermore, define $\mathcal{I}^{-0}$ (a finite sequence over $in(A) \cup \{\lambda\}$) and for each $n > 0$

and each $m(n-1) < k \leq m(n)$ define $\mathcal{I}^{-k}$ such that $\mathcal{I}^{-(k-1)} \leq \mathcal{I}^{-k}$. After the definition show that this leads to a chain $(\mathcal{I}^{-n})_{n\geq 0}$, ordered by prefix, and let $\mathcal{I} = \lim_{n\to\infty} \mathcal{I}^{-n}$.

Base case $n = 0$:

Define: $m(0) = 0$
$\mathcal{I}^{-0} = \varepsilon$

Inductive step $n > 0$:

$(\Sigma^n, \mathcal{I}_p^n)$ is related to $(\Sigma^{n-1}, \mathcal{I}_p^{n-1})$ according to exactly one of the five cases in the definition of $R_{(g_p,f_p)}$ (cf. Definition 4.17). Consider these five cases:

1. Define: $m(n) = m(n-1)+1$
   $\mathcal{I}^{-m(n)} = \mathcal{I}^{-(m(n)-1)}\lambda$

2. Define: $m(n) = m(n-1)$

3. Define: $m(n) = m(n-1)+1$
   $\mathcal{I}^{-m(n)} = \mathcal{I}^{-(m(n)-1)}\lambda$

4. Here let $(a,t) = head(\mathcal{I}_p^{n-1})$.

   (a) Assume $f_p(\Sigma^{n-1}) = \omega$

   Define: $m(n) = m(n-1)+2$
   $\mathcal{I}^{-(m(n)-1)} = \mathcal{I}^{-(m(n)-2)}\lambda$
   $\mathcal{I}^{-m(n)} = \mathcal{I}^{-(m(n)-1)}a$

   (b) Assume $f_p(\Sigma^{n-1}) = (\omega, b, s)$

   Define: $m(n) = m(n-1)+1$
   $\mathcal{I}^{-m(n)} = \mathcal{I}^{-(m(n)-1)}a$

5. Define: $m(n) = m(n-1)+1$
   $\mathcal{I}^{-m(n)} = \mathcal{I}^{-(m(n)-1)}\lambda$

This concludes the inductive definition. Only case 2 above does not increment m. However, this case occurs at most once, namely if Σ^{n-1} is finite and Σ^n is not. Thus, $\lim_{n\to\infty} m(n) = \infty$. This also implies that $(\mathcal{I}^{-n})_{n\geq\infty}$ is a chain ordered by prefix. Now, define $\mathcal{I} = \lim_{n\to\infty} \mathcal{I}^{-n}$ and, for all n, let $\mathcal{I}^n = \mathcal{I} - \mathcal{I}^{-n}$. (Thus, $\mathcal{I}^{-n} \,\hat{}\, \mathcal{I}^n = \mathcal{I}$.)

We now argue that $\mathcal{I}$ is an environment sequence for A. With an argument similar to the one that shows $\lim_{n\to\infty} m(n) = \infty$, it is easy to see that $\mathcal{I}$ is infinite. Now, assume that $\mathcal{I}$ does not contain infinitely many occurrences of λ. This implies that there exists a number n' such that for all $n > n'$, the inductive step is handled by case 4b above. Let, for all $k \geq n'$, $f_p(\Sigma^k) = (\omega^k, a^k, s^k)$. Then by definition of f_p, $ltime(\omega^k) = max(ltime(\Sigma^k), lloctime(\Sigma^k) + \delta)$ which, since case 4b adds input actions, equals $max(ltime(\Sigma^k), lloctime(\Sigma^{n'}) + \delta)$.

- Assume $lloctime(\Sigma^{n'}) + \delta \geq ltime(\Sigma^{n'})$. Then $ltime(\Sigma^{n'+1}) \leq ltime(\omega^{n'}) = lloctime(\Sigma^{n'}) + \delta$. By induction it is easy to see that for all $k \geq n'$, $ltime(\Sigma^k) \leq lloctime(\Sigma^{n'})$, which implies that $ltime(\Sigma') \leq lloctime(\Sigma^{n'})$. But this contradicts the fact that Σ' is admissible.

- Assume $lloctime(\Sigma^{n'}) + \delta < ltime(\Sigma^{n'})$. Then $ltime(\Sigma^{n'+1}) \leq ltime(\omega^{n'}) = ltime(\Sigma^{n'})$, and, by definition of $R_{(g_p,f_p)}$, $ltime(\Sigma^{n'+1}) \geq ltime(\Sigma^{n'})$. Thus, $ltime(\Sigma^{n'+1}) = ltime(\Sigma^{n'})$. Again, by induction, $ltime(\Sigma') \leq ltime(\Sigma^{n'})$, and since $\Sigma^{n'}$ is finite, this contradicts the fact that Σ' is admissible.

Thus, both cases lead to a contradiction, which allows the conclusion that $\mathcal{I}$ contains infinitely many occurrences of λ. Finally, $\mathcal{I}$ consists of actions from $in(A_p) \cup \{\lambda\}$. Since, by Definition 5.1, $in(A) = in(A_p)$, $\mathcal{I}$ is an infinite sequence over $in(A) \cup \{\lambda\}$ containing infinitely many occurrences of λ. Thus, $\mathcal{I}$ is an environment sequence for A.

Similar to the way $(\mathcal{I}^{-n})_{n\geq 0}$ is defined, now define a chain $(\alpha^n)_{n\geq 0}$ of executions of A, ordered by prefix. Thus, define α^0 and, for each $n > 0$ and each $m(n-1) < k \leq m(n)$, α^k such that $\alpha^{k-1} \leq \alpha^k$. In the same induction prove, for each n:

P1 $untime(\Sigma^n) = \alpha^{m(n)}$.

P2 If $n > 0$ and for each $m(n-1) < k \leq m(n)$, $R_{(g,f)}(\alpha^{k-1}, \mathcal{I}^{k-1}) = (\alpha^k, \mathcal{I}^k)$.

Base case $n = 0$:

Define: $\alpha^0 = untime(\Sigma)$

P1 $untime(\Sigma^0) = untime(\Sigma) = \alpha^0 = \alpha^{m(0)}$.

P2 Vacuously satisfied.

Inductive step $n > 0$:

Assume P1 as induction hypothesis. Again consider the five cases in the definition of $R_{(g_p,f_p)}$.

Case 1 Here $(\Sigma^n, \mathcal{I}_p^n) = (\Sigma^{n-1} \frown \omega a\{s\}, \mathcal{I}_p^{n-1})$ with $f_p(\Sigma^{n-1}) = (\omega, a, s)$. Then, by definition of (g_p, f_p), $f(untime(\Sigma^{n-1})) = (a, s.basic)$. Furthermore, by the induction hypothesis and the definition of $m(n)$, $untime(\Sigma^{n-1}) = \alpha^{m(n-1)} = \alpha^{m(n)-1}$.

Define: $\alpha^{m(n)} = \alpha^{m(n)-1} a(s.basic)$

P1 $untime(\Sigma^n) = untime(\Sigma^{n-1} \frown \omega a\{s\}) = untime(\Sigma^{n-1}) \frown untime(\omega a\{s\}) = \alpha^{m(n-1)} \frown (fstate(\omega).basic) a(s.basic) = \alpha^{m(n)-1} a(s.basic) = \alpha^{m(n)}$.

P2 This condition must be shown for $k = m(n)$.
Definition of $\mathcal{I}^{-m(n)}$ above implies $\mathcal{I}^{m(n)-1} = \mathcal{I} - \mathcal{I}^{-(m(n)-1)} = \lambda(\mathcal{I} - \mathcal{I}^{-m(n)}) = \lambda \mathcal{I}^{m(n)}$.
By case 1 of the definition of $R_{(g,f)}$ (cf. Definition 3.12) the result now follows.

Case 2 Here $\Sigma^n = \Sigma^{n-1} \frown \omega$ where $\omega = f_p(\Sigma^{n-1})$. Also $m(n) = m(n-1)$ in this case.

P1 $untime(\Sigma^n) = untime(\Sigma^{n-1} \frown \omega) = untime(\Sigma^{n-1}) = \alpha^{m(n-1)} = \alpha^{m(n)}$.

P2 Vacuously satisfied.

Case 3 This case is handled similarly to case 1.

Case 4 Here, let $(a,t) = head(\mathcal{I}_p^{n-1})$. Then, $(\Sigma^n, \mathcal{I}_p^n) = (\Sigma^{n-1} \frown \omega' a\{s'\}, tail(\mathcal{I}_p^{n-1}))$, where $\omega' = (f_p(\Sigma^{n-1}).trj) \lhd t$ and $s' = g_p(\Sigma^{n-1} \frown \omega', a)$. Then, the induction hypothesis and the definition of (g_p, f_p), imply $g(untime(\Sigma^{n-1} \frown \omega'), a) = g(untime(\Sigma^{n-1}), a) = g(\alpha^{m(n-1)}, a) = s'.basic$.

Case 4.1 Assume $f_p(\Sigma^{n-1}) = \omega$

Then the induction hypothesis and the definitions of (g_p, f_p) and $m(n)$, imply $f(untime(\Sigma^{n-1})) = f(\alpha^{m(n-1)}) = f(\alpha^{m(n)-2}) = \perp$.

Define: $\alpha^{m(n)-1} = \alpha^{m(n)-2}$
$\alpha^{m(n)} = \alpha^{m(n)-1} a(s'.basic)$

P1 $untime(\Sigma^n) = untime(\Sigma^{n-1} \frown \omega' a\{s'\}) = untime(\Sigma^{n-1}) a(s'.basic) = \alpha^{m(n-1)} a(s'.basic) = \alpha^{m(n)-2} a(s'.basic) = \alpha^{m(n)-1} a(s'.basic) = \alpha^{m(n)}$.

P2 This condition must be shown for $k = m(n) - 1$ and $k = m(n)$.
As for the previous cases, it is easy to see that $\mathcal{I}^{m(n)-2} = \lambda \mathcal{I}^{m(n)-1}$. Then the result for $k = m(n) - 1$ directly follows from case 2 of the definition of $R_{(g,f)}$.
Similarly, $\mathcal{I}^{m(n)-1} = a\mathcal{I}^{m(n)}$. Furthermore, $g(\alpha^{m(n-1)}, a) = s'.basic$ which implies $g(\alpha^{m(n)-2}, a) = g(\alpha^{m(n)-1}, a) = s'.basic$. Now the result for $k = m(n)$ follows from case 3 of the definition of $R_{(g,f)}$.

Case 4.2 Assume $f_p(\Sigma^{n-1}) = (\omega, b, s)$

Define: $\alpha^{m(n)} = \alpha^{m(n)-1} a(s'.basic)$

P1 $untime(\Sigma^n) = untime(\Sigma^{n-1} \frown \omega' a\{s'\}) = untime(\Sigma^{n-1}) a(s'.basic) = \alpha^{m(n-1)} a(s'.basic) = \alpha^{m(n)-1} a(s'.basic) = \alpha^{m(n)}$.

P2 This condition must be shown for $k = m(n)$.
Since $\mathcal{I}^{m(n)-1} = a\mathcal{I}^{m(n)}$ and $g(\alpha^{m(n-1)}, a) = g(\alpha^{m(n)-1}, a) = s'.basic$, the result follows from case 3 of the definition of $R_{(g,f)}$.

Case 5 In this case $(\Sigma^n, \mathcal{I}_p^n) = (\Sigma^{n-1}, \mathcal{I}_p^{n-1})$.

Define: $\alpha^{m(n)} = \alpha^{m(n)-1}$

P1 The induction hypothesis and the definition of $m(n)$ imply that $untime(\Sigma^n) = untime(\Sigma^{n-1}) = \alpha^{m(n-1)} = \alpha^{m(n)-1} = \alpha^{m(n)}$.

P2 This condition must be shown for $k = m(n)$.

By definition of $(\Sigma^n)_{n\geq 0}$, there exists an $n' < n$ such that $\Sigma^{n'}$ is finite, $f_p(\Sigma^{n'}) = \omega$, for some ω, and $\Sigma^{n'+1} = \Sigma^{n'+2} = \cdots = \Sigma^{n-1} = \Sigma^n = \Sigma^{n'} \frown \omega$.

Then Definition 5.6 of (g_p, f_p) implies that $f(\mathit{untime}(\Sigma^{n'})) = \bot$ and then, since $\mathit{untime}(\Sigma^{n'}) = \mathit{untime}(\Sigma^{n-1})$, $f(\mathit{untime}(\Sigma^{n-1})) = \bot$. Thus, by the induction hypothesis and definition of $m(n)$, $f(\alpha^{m(n-1)}) = f(\alpha^{m(n)-1}) = \bot$,

By definition of $\mathcal{I}^{-m(n)}$, $\mathcal{I}^{m(n)-1} = \mathcal{I} - \mathcal{I}^{-(m(n)-1)} = \lambda(\mathcal{I} - \mathcal{I}^{-m(n)}) = \lambda\mathcal{I}^{m(n)}$.

Now, since $\alpha^{m(n)} = \alpha^{m(n)-1}$, $f(\alpha^{m(n)-1}) = \bot$, and $\mathcal{I}^{m(n)-1} = \lambda\mathcal{I}^{m(n)}$, the result follows from case 2 in the definition of $R_{(g,f)}$ (cf. Definition 3.12).

This concludes the inductive definition and proof. By P2 and the fact that $\lim_{n\to\infty} m(n) = \infty$, $(\alpha^n, \mathcal{I}^n)_{n\geq\infty}$ is the outcome sequence of (g, f) given $\mathit{untime}(\Sigma)$ and $\mathcal{I}$. Now,

$$
\begin{aligned}
\mathit{untime}(\Sigma') &\stackrel{1}{=} \mathit{untime}(\lim_{n\to\infty} \Sigma^n) \\
&\stackrel{2}{=} \lim_{n\to\infty}(\mathit{untime}(\Sigma^n)) \\
&\stackrel{3}{=} \lim_{n\to\infty} \alpha^{m(n)} \\
&\stackrel{4}{=} \lim_{n\to\infty} \alpha^n \\
&\stackrel{5}{=} \mathcal{O}_{(g,f)}(\mathit{untime}(\Sigma), \mathcal{I})
\end{aligned}
$$

where step 1 follows from the definition of Σ', step 2 follows from continuity of *untime* (easy to verify), step 3 follows from P1 in the induction proof, step 4 follows from the fact that $\lim_{n\to\infty} m(n) = \infty$, and finally step 5 follows from the fact that $(\alpha^n, \mathcal{I}^n)_{n\geq\infty}$ is the outcome sequence of (g, f) given $\mathit{untime}(\Sigma)$ and $\mathcal{I}$. This concludes the proof. ∎

Lemma 5.9

Let A be a safe I/O automaton with $\nu \notin \mathit{acts}(A)$ and let (g, f) be an (untimed) strategy defined on A. Let $A_p = \mathit{patient}(A)$ and $(g_p, f_p) = \mathit{patient}_\delta(g, f)$ for some arbitrary positive real number δ. Let $\Sigma \in$ t-exec$^*(A_p)$ be an arbitrary finite timed execution of A_p, $\mathcal{I}$ an arbitrary timed environment sequence for A_p compatible with Σ, and Σ' an arbitrary timed execution of the outcome $\mathcal{O}_{(g_p,f_p)}(\Sigma, \mathcal{I})$. Then for any two elements (a_1, t_1) and (a_2, t_2) in t-seq$(\Sigma' - \Sigma) \upharpoonright (\mathit{local}(A_p) \times \mathsf{T})$, $|t_2 - t_1| \geq \delta$.

Proof. Let (a_1, t_1) and (a_2, t_2) be two arbitrary pairs in $\gamma = $ t-seq$(\Sigma' - \Sigma) \upharpoonright (\mathit{local}(A_p) \times \mathsf{T})$ and assume, without loss of generality, that (a_1, t_1) occurs before (a_2, t_2) in γ. This implies that $t_2 \geq t_1$. Furthermore, assume, again without loss of generality, that (a_1, t_1) and (a_2, t_2) are consecutive in γ. Let $(\Sigma^n, \mathcal{I}^n)_{n\geq 0}$ be an outcome sequence of (g_p, f_p) given Σ and $\mathcal{I}$ such that $\Sigma' = \lim_{n\to\infty} \Sigma^n$.

Definition 4.17 now implies the existence of a number n such that (a_2, t_2) is not in t-seq$(\Sigma^n - \Sigma) \upharpoonright (\mathit{local}(A_p) \times \mathsf{T})$ and $\Sigma^{n+1} = \Sigma^n \frown \omega a_2\{s\}$ with $f_p(\Sigma^n) = (\omega, a_2, s)$ and $\mathit{ltime}(\omega) = t_2$. Also, (a_1, t_1) must be in t-seq$(\Sigma^n - \Sigma) \upharpoonright (\mathit{local}(A_p) \times \mathsf{T})$ since otherwise it could not occur before (a_2, t_2) in γ. Let $t_l = \mathit{lloctime}(\Sigma^n)$. Since $a_1 \in \mathit{local}(A_p)$, $t_1 \leq t_l$.

By definition of f_p (Definition 5.6), $\mathit{ltime}(\omega) = \mathit{max}(\mathit{ltime}(\Sigma^n), \mathit{lloctime}(\Sigma^n) + \delta)$. Thus, $t_2 = \mathit{ltime}(\omega) \geq t_l + \delta \geq t_1 + \delta$, or equivalently, $t_2 - t_1 \geq \delta$. That suffices. ∎

It is now possible to prove that for any environment-free strategy (g, f) for a live I/O automaton (A, L) and any positive δ, $patient_\delta(g, f)$ is an environment-free (timed) strategy for $(A_p, L_p \cup t\text{-}exec^{Zt}(A_p))$, where $(A_p, L_p) = patient(A, L)$.

Lemma 5.10

Let (A, L) be a live I/O automaton with $\nu \notin acts(A)$ and let (g, f) be an (untimed) environment-free strategy for (A, L). Furthermore, let $(A_p, L_p) = patient(A, L)$. Then, for any positive real number δ, $patient_\delta(g, f)$ is a (timed) environment-free strategy for $(A_p, L_p \cup t\text{-}exec^{Zt}(A_p))$.

Proof. Let δ be an arbitrary positive real number and let $(g_p, f_p) = patient_\delta(g, f)$. Note that by Lemma 5.7 (g_p, f_p) is a (timed) strategy defined on A_p. By Definition 4.22 one must show that

1. A_p is a safe timed I/O automaton,
2. $L_p \cup t\text{-}exec^{Zt}(A_p) \subseteq t\text{-}exec(A_p)$,
3. $\mathcal{O}_{(g_p,f_p)}(\Sigma, \mathcal{I}_p) \subseteq L_p \cup t\text{-}exec^{Zt}(A_p)$, for all $\Sigma \in t\text{-}exec^*(A_p)$ and all timed environment sequences $\mathcal{I}_p$ for A_p compatible with Σ, and
4. (g_p, f_p) is Zeno-tolerant.

Consider the points one at a time.

1. Definition 5.1 directly implies that A_p is a safe timed I/O automaton.
2. By Definition 5.4, $L_p \subseteq t\text{-}exec^\infty(A_p)$ and since also $t\text{-}exec^{Zt}(A_p) \subseteq t\text{-}exec(A_p)$, the result follows.
3. Let $\Sigma \in t\text{-}exec^*(A_p)$ be an arbitrary finite timed execution of A_p and $\mathcal{I}_p$ be an arbitrary timed environment sequence for A_p compatible with Σ. Let $\Sigma' \in \mathcal{O}_{(g_p,f_p)}(\Sigma, \mathcal{I}_p)$ be an arbitrary element of the outcome. By Lemma 4.18, Σ is either Zeno or admissible.
   - Assume Σ' is Zeno.
     Then, by Lemma 5.9 there are only finitely many locally-controlled actions of A_p in Σ'. Now, Lemma 4.18 implies that Σ' contains infinitely many input actions. Thus, $\Sigma \in t\text{-}exec^{Zt}(A_p)$. That suffices.
   - Assume Σ' is admissible.
     By Lemma 5.8 there exists an environment sequence $\mathcal{I}$ for A such that $untime(\Sigma') = \mathcal{O}_{(g,f)}(untime(\Sigma), \mathcal{I})$. The fact that (g, f) is an environment-free strategy for (A, L) implies $untime(\Sigma') \in L$. This implies, by Definition 5.4, that $\Sigma' \in L_p$. That suffices.
4. To prove that (g_p, f_p) is Zeno-tolerant (cf. Definition 4.21), it suffices to note that the previous case implies the following. For arbitrary $\Sigma \in t\text{-}exec^*(A_p)$ and arbitrary timed environment sequences $\mathcal{I}_p$ for A_p compatible with Σ, $\mathcal{O}_{(g_p,f_p)}(\Sigma, \mathcal{I}_p) \subseteq L_p \cup t\text{-}exec^{Zt}(A_p)$, where $L_p \subseteq t\text{-}exec^\infty(A_p)$ by Definition 5.4. ■

Finally, we can prove that for any live I/O automaton (A, L), $patient(A, L)$ is a live timed I/O automaton.

Proposition 5.11

Let (A, L) be a live I/O automaton. Then $patient(A, L)$ is a live timed I/O automaton.

Proof. Let $(A_p, L_p) = patient(A, L)$. Definition 5.1 implies that A_p is a safe timed I/O automaton. Furthermore, $L \subseteq t\text{-}exec^{\infty}(A_p)$ by Definition 5.4. Finally, Lemma 5.10 implies that the pair $(A_p, L_p \cup t\text{-}exec^{Zt}(A_p))$ is environment-free. By Definition 4.23, this suffices. ■

Now attention is turned to proving the Embedding Theorem, which states that the safe and live preorders of live I/O automata are preserved by the patient operator. A few preliminary lemmas are needed.

Lemma 5.12

Let A be a safe I/O automaton with $\nu \notin acts(A)$ and let $A_p = patient(A)$. Furthermore, let $\Sigma \in t\text{-}exec(A_p)$. Then,

$$untime(t\text{-}trace_{A_p}(\Sigma)) = trace_A(untime(\Sigma))$$

Proof. Let $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$.

Then, $t\text{-}trace_{A_p}(\Sigma) = ((a_1, ftime(\omega_1))(a_2, ftime(\omega_2)) \cdots \restriction (vis(A_p) \times \mathsf{T}), ltime(\Sigma))$ and it follows that $untime(t\text{-}trace_{A_p}(\Sigma)) = a_1 a_2 \cdots \restriction vis(A_p)$.

Now, $untime(\Sigma) = (fstate(\omega_0).basic) a_1 (fstate(\omega_1).basic) a_2 (fstate(\omega_2).basic) \cdots$ and it follows that $trace_A(untime(\Sigma)) = a_1 a_2 \cdots \restriction ext(A)$.

By Definition 5.1, $vis(A_p) = ext(A)$, so the result follows. ■

Lemma 5.13

Let (A, L) be a live I/O automaton with $\nu \notin acts(A)$. Then,

1. *If $\gamma \in t\text{-}traces(patient(A))$ then $untime(\gamma) \in traces(A)$.*
2. *If $\beta \in traces(A)$ and $\gamma \in tsp(ext(A))$ with $\beta = untime(\gamma)$ such that if $seq(\gamma)$ is Zeno, then $ltime(\gamma)$ is the limit of the times in $seq(\gamma)$, then $\gamma \in t\text{-}traces(patient(A))$.*
3. *If $\gamma \in t\text{-}traces(patient_A(L))$ then $untime(\gamma) \in traces(L)$.*
4. *If $\beta \in traces(L)$ and $\gamma \in tsp(ext(A))$ is admissible with $\beta = untime(\gamma)$, then $\gamma \in t\text{-}traces(patient_A(L))$.*

Proof. Let $(A_p, L_p) = patient(A, L)$. Consider the four parts separately.

1. Let $\gamma \in$ *t-traces*(A_p) and $\Sigma \in$ *t-exec*(A_p) such that *t-trace*$(\Sigma) = \gamma$. Then $untime(\gamma) = untime(\textit{t-trace}(\Sigma))$ which, by Lemma 5.12, equals $trace(untime(\Sigma))$. By Lemma 5.3, $untime(\Sigma) \in exec(A)$. That suffices.

2. Let $\beta = a^1 a^2 a^3 \cdots \in traces(A)$ and $\gamma \in tsp(ext(A))$ such that $\beta = untime(\gamma)$. Thus, $\gamma = ((a^1, t^1)(a^2, t^2)(a^3, t^3)\cdots, t_l)$ for nondecreasing times $t^1, t^2, t^3, \ldots$ and time t_l (possibly ∞) greater than or equal to t^i for all i in γ. Also, let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots \in exec(A)$ such that $\beta = trace(\alpha)$ and α is finite if β (and thus $seq(\gamma)$) is finite.

   By definition of *trace*, each external action a_i in α corresponds to an action $a^j (= a_i)$ in β and thus a pair (a^j, t^j) in $seq(\gamma)$. Define t^j to be the time of occurrence t_i of a_i. For internal actions a_i in α, define the time of occurrence t_i to be the time of occurrence of the previous external action in α or 0 if no such action exists. Define, if α is finite with a_n being its last action, $t_{n+1} = t_l$ (possibly ∞).

   Now, define $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ where $dom(\omega_0) = [0, t_1]$, $rng(\omega_0) = \{(s_0, t) \mid t \in dom(\omega_0)\}$, $dom(\omega_1) = [t_1, t_2]$, $rng(\omega_1) = \{(s_1, t) \mid t \in dom(\omega_1)\}$, and $dom(\omega_2) = [t_2, t_3]$, $rng(\omega_2) = \{(s_2, t) \mid t \in dom(\omega_2)\}$. Then clearly, by Definition 5.1, $\Sigma \in$ *t-exec*(A_p), and furthermore *t-trace*$(\Sigma) = (seq(\gamma), ltime(\Sigma))$. (Note, that $vis(A_p) = ext(A)$.)

   If α is finite, then, depending on t_l, Σ is admissible or finite, but in both cases $ltime(\Sigma) = t_l$.

   If α is infinite, then $seq(\gamma)$ is infinite and
   (1) if $seq(\gamma)$ is Zeno, then $ltime(\Sigma)$ equals the limit of the times in $seq(\gamma)$, which equals t_l by assumption, and
   (2) if $seq(\gamma)$ is admissible, then Σ is admissible.

   Thus, in all cases $ltime(\Sigma) = t_l$. Finally, conclude that *t-trace*$(\Sigma) = \gamma$ which implies that $\gamma \in$ *t-traces*(A_p) as required.

3. Let $\gamma \in$ *t-traces*(L_p) and $\Sigma \in L_p$ such that *t-trace*$(\Sigma) = \gamma$. Then $untime(\gamma) = untime(\textit{t-trace}(\Sigma))$ which, by Lemma 5.12, equals $trace(untime(\Sigma))$. Definition 5.4 and the fact that $\Sigma \in L_p$ imply that $untime(\Sigma) \in L$. That suffices.

4. This proof is similar to the proof of Part 2 except that α is chosen from L and thus might be infinite even though β is finite. If this is the case the times of occurrence of the internal actions in the diverging suffix of α are chosen to increase by some fixed amount, say, 1. Then Σ is admissible, and clearly $\alpha = untime(\Sigma)$, so by Definition 5.4, $\Sigma \in L_p$. Thus, $\gamma \in$ *t-traces*(L_p). ∎

Theorem 5.14 (Embedding Theorem)

Let (A, L) and (B, M) be live I/O automata with $\nu \notin (acts(A) \cup acts(B))$. Then

1. $(A, L) \sqsubseteq_{\mathrm{S}} (B, M)$ *iff* $patient(A, L) \sqsubseteq_{\mathrm{St}} patient(B, M)$.

2. $(A, L) \sqsubseteq_{\mathrm{L}} (B, M)$ *iff* $patient(A, L) \sqsubseteq_{\mathrm{Lt}} patient(B, M)$.

Proof. Let $(A_p, L_p) = \mathit{patient}(A, L)$ and $(B_p, M_p) = \mathit{patient}(B, M)$. The two parts of the lemma are considered separately.

1. $\Longrightarrow$: Let $\gamma \in \mathit{t\text{-}traces}(A_p)$. By Lemma 5.13 Part 1, $\beta = \mathit{untime}(\gamma) \in \mathit{traces}(A)$, which implies, since $(A, L) \sqsubseteq_{\mathrm{S}} (B, M)$, that $\beta \in \mathit{traces}(B)$. Now, the fact that γ is a timed sequence pair over $\mathit{vis}(A_p) = \mathit{vis}(B_p) = \mathit{ext}(B)$ and the fact that γ satisfies the property $\mathit{seq}(\gamma)$ being Zeno implies $\mathit{ltime}(\gamma)$ is the limit of the times in $\mathit{seq}(\gamma)$, Lemma 5.13 Part 2 implies that $\gamma \in \mathit{t\text{-}traces}(B_p)$, as required.

   $\Longleftarrow$: Let $\beta \in \mathit{traces}(A)$ and let γ be any, say, admissible timed sequence pair over $\mathit{ext}(A)$ such that $\mathit{untime}(\gamma) = \beta$. (Such a timed sequence pair clearly exists.) Then, by Lemma 5.13 Part 2, $\gamma \in \mathit{t\text{-}traces}(A_p)$. Thus, the assumption that $\mathit{patient}(A, L) \sqsubseteq_{\mathrm{St}} \mathit{patient}(B, M)$ implies $\gamma \in \mathit{t\text{-}traces}(B_p)$. Lemma 5.13 Part 1 shows that $\beta = \mathit{untime}(\gamma) \in \mathit{traces}(B)$, as required.

2. Similar to Part 1 by using Lemma 5.13 Parts 3 and 4. ∎

Finally we prove a result which is important when doing specification and verification in a modular fashion. Namely, the *patient* operator commutes with the three operators on safe and live (timed) I/O automata. First, let $\equiv_{\mathrm{St}}$ and $\equiv_{\mathrm{Lt}}$ denote the *kernels* of the preorders $\sqsubseteq_{\mathrm{St}}$ and $\sqsubseteq_{\mathrm{Lt}}$, respectively.[2]

Proposition 5.15

Let (A, L) and $(A_1, L_1), \ldots, (A_N, L_N)$ be live I/O automata and let $\equiv_X$ be one of $\equiv_{\mathrm{St}}$ and $\equiv_{\mathrm{Lt}}$.

1. *Let $(A_1, L_1), \ldots, (A_N, L_N)$ be compatible. Then,*

   $\mathit{patient}((A_1, L_1)\| \cdots \|(A_N, L_N)) \equiv_X \mathit{patient}(A_1, L_1)\| \cdots \|\mathit{patient}(A_N, L_N)$

2. *Let $\Lambda \subseteq \mathit{local}(A)$. Then,*

   $\mathit{patient}((A, L) \setminus \Lambda) \equiv_X \mathit{patient}(A, L) \setminus \Lambda$

3. *Let ρ be an action mapping applicable to A and let ρ_ν be ρ extended with the mapping $[\nu \mapsto \nu]$. Then,*

   $\mathit{patient}(\rho(A, L)) \equiv_X \rho_\nu(\mathit{patient}(A, L))$

Proof. We show the proofs for $\equiv_{\mathrm{St}}$. The proofs for $\equiv_{\mathrm{Lt}}$ are similar.

1. First note that since $(A_1, L_1), \ldots, (A_N, L_N)$ are compatible, then also $\mathit{patient}(A_1, L_1)$, $\ldots$, $\mathit{patient}(A_N, L_N)$ are compatible.

[2] The kernel of a preorder $\sqsubseteq$ is defined to be the equivalence $\equiv$ defined by $x \equiv y \triangleq x \sqsubseteq y \wedge y \sqsubseteq x$.

Observe the simple fact that for each timed execution Σ, $untime(\Sigma)\lceil A_i = untime(\Sigma\lceil A_i)$. Then,

$$\begin{array}{lll} & \Sigma \in t\text{-}exec(patient(A_1\|\cdots\|A_N)) & \\ \text{iff} & untime(\Sigma) \in exec(A_1\|\cdots\|A_N) & \text{Lemma 5.3} \\ \text{iff} & \forall_{1\leq i\leq N} : untime(\Sigma)\lceil A_i \in exec(A_i) & \text{Lemma 3.5} \\ \text{iff} & \forall_{1\leq i\leq N} : untime(\Sigma\lceil A_i) \in exec(A_i) & \text{observation above} \\ \text{iff} & \forall_{1\leq i\leq N} : \Sigma\lceil A_i \in t\text{-}exec(patient(A_i)) & \text{Lemma 5.3} \\ \text{iff} & \Sigma \in t\text{-}exec(patient(A_1)\|\cdots\|patient(A_N)) & \text{Lemma 4.11} \end{array}$$

That suffices.

2. Note that since $\Lambda \subseteq local(A)$, also $\Lambda \subseteq local(patient(A))$. Then,

$$\begin{array}{lll} & \Sigma \in t\text{-}exec(patient(A \setminus \Lambda)) & \\ \text{iff} & untime(\Sigma) \in exec(A \setminus \Lambda) & \text{Lemma 5.3} \\ \text{iff} & untime(\Sigma) \in exec(A) & \text{Lemma 3.7} \\ \text{iff} & \Sigma \in t\text{-}exec(patient(A)) & \text{Lemma 5.3} \\ \text{iff} & \Sigma \in t\text{-}exec(patient(A) \setminus \Lambda) & \text{Lemma 4.13} \end{array}$$

That suffices.

3. First note that since ρ is applicable to A, ρ_ν is applicable to $patient(A)$. Also note that, since each renaming function ρ is injective, there is an inverse function $\rho^{-1} : \rho(dom(\rho)) \rightarrow dom(\rho)$ such that $\rho^{-1}(b)$ is the unique a satisfying $\rho(a) = b$.

   Observe the simple fact that for any timed execution Σ and any rename function ρ', $\rho'(untime(\Sigma)) = untime(\rho'_\nu(\Sigma))$, where ρ'_ν is obtained from ρ' by adding the mapping $[\nu \mapsto \nu]$. Then,

$$\begin{array}{lll} & \Sigma \in t\text{-}exec(patient(\rho(A)) & \\ \text{iff} & untime(\Sigma) \in exec(\rho(A)) & \text{Lemma 5.3} \\ \text{iff} & \rho^{-1}(untime(\Sigma)) \in exec(A) & \text{Lemma 3.9} \\ \text{iff} & untime(\rho_\nu^{-1}(\Sigma)) \in exec(A) & \text{observation above} \\ \text{iff} & \rho_\nu^{-1}(\Sigma) \in t\text{-}exec(patient(A)) & \text{Lemma 5.3} \\ \text{iff} & \Sigma \in t\text{-}exec(\rho(patient(A))) & \text{Lemma 4.15} \end{array}$$

   That suffices. ∎

6 Proof Techniques

This section presents a number of techniques to prove the safe preorder and the live preorder on live (timed) I/O automata. The techniques are based on results in [LV93a]([LV93b]), which show that several *simulation relations* between (timed) automata are sound with respect to the *safe* preorder. This section also shows that a stronger result, called the *Execution Correspondence Theorem*, can be proven for the simulations of [LV93a]([LV93b]). Specifically, that there is a certain correspondence between the *executions* of the involved automata and not only between their *traces*. In the untimed model, liveness conditions of live I/O automata are stated in terms of executions and not in terms of traces, thus the *Execution Correspondence Theorem* can form the basis for proofs of the *live* preorder. In the timed model, where liveness conditions are given in terms of timed executions, the timed version of *Execution Correspondence Theorem* along with a *sampling characterization* of the live timed executions is used as the basis for proofs of the live preorder.

The proof that a live (timed) I/O automaton A implements (based on the live preorder) another live (timed) I/O automaton B consists of two main steps. First a simulation relation between the safe (timed) I/O automata parts is proven. Because of the soundness of the simulation relations with respect to the safe preorder(s), the simulation relation already implies that the safe preorder(s) holds. The second step, which is described in detail in this section, uses the simulation relation found in the first step and the Execution Correspondence Theorem to prove the live preorder.

Ideas similar to those of the Execution Correspondence Theorem appear in the soundness proofs of the simulations for the safe preorder given in [LT87, LV93a]. The contribution of this section is to formally state and prove the Execution Correspondence Theorem for a large class of simulations and to show how it can be used as the basis for proving the live preorder.

Several pragmatic considerations support the approach to verification taken in this section. For example, when proving the safe and live preorders in the untimed setting, it is often difficult to reason directly about the traces of the involved live I/O automata. In particular, the traces of an automaton are defined implicitly as the traces of the executions of the automaton, and the liveness condition of a live automaton is usually defined implicitly to be a set of executions of the automaton that satisfy certain properties, typically specified in some temporal logic. Thus, the sets of traces and live traces are not directly available. Rather, they are derived from automata, temporal logic formulas, etc. As a result, simulation based proof techniques which use the information available directly, e.g., automata, and which are sound with respect to the safe and live preorders, are attractive.

Furthermore, using our proof methodology, the main complexity of a correctness proof for the safe and live preorders is found in the simulation proof. Fortunately, simulation proofs have a nice case structure that scales well to large examples and provides good intuitive insight into the automata for which the simulation relation is being proven. Another practical advantage of our proof methodology for the live preorder is that it proves the safe preorder as a side result. The work in [SLL93, LLS93] shows why this can be useful. In [SLL93, LLS93] the five-packet-handshake protocol of [Bel76] is shown to guarantee a safety property of "at-most-once message delivery", as well as liveness properties such as "in the absence of crashes, each message will eventually be delivered". However, the liveness of the system depends on liveness assumptions on the channels connecting the sender and the receiver: "if a packet is sent infinitely often then it will be received infinitely often". This liveness assumption must hold even though the channels are allowed to lose packets. However, if the channel is cut, then correctness as defined

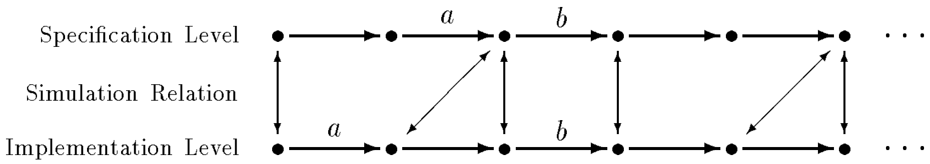


Figure 2: Example of a simulation. The actions a and b are external actions. The remaining steps represent internal actions.

by the live preorder is no longer ensured. Fortunately, since the safety property is independent of the liveness of the channel, safety is still guaranteed, i.e., no message is delivered more than once.

6.1 Untimed Proof Techniques

Section 6.1.1 defines a number of simulation relations taken from [LV93a]. Section 6.1.2 presents the Execution Correspondence Theorem. Finally, Sections 6.1.3 and 6.1.4 deal with proving the safe and live preorders, respectively.

6.1.1 Simulation Proof Techniques

This section presents simulation relations taken from [LV93a]. For the purpose of generality, the definitions are stated in terms of automata. All results are also valid for the special case of safe I/O automata.

A *simulation relation* is a relation between the states of one automaton, called the *concrete*, *low-level*, or *implementation* automaton, and the states of another automaton, called the *abstract*, *high-level*, or *specification* automaton, such that certain properties hold. The exact properties depend on the type of simulation (forward, backward, etc.) but they generally consist of two properties. First, the start states of the two automata must be related in a certain way, and, second, each step of the implementation automaton must "correspond" to a sequence of steps of the specification automaton.

The second property is depicted in Figure 2. For each step of the implementation automaton, i.e., for each concrete step, there must exist a sequence of (abstract) steps of the specification automaton between states related—by the simulation relation—to the pre- and post-state of the considered concrete step, such that the sequence of abstract steps contains exactly the same external actions as the concrete step. How the sequence of abstract steps is selected depends on what type of simulation is considered.

Below forward simulations, refinement mappings, backward simulations, history relations, and prophecy relations are defined. The definitions are similar to the definitions given in [LV93a] where combinations of forward and backward simulations are also considered. The reader is referred to [LV93a] for details about, e.g., partial completeness of the simulation techniques.

The simulation techniques use *invariants* (and are thus called *weak* in [LV93a]) of the implementation and specification automata to restrict the steps which need to be considered. Define an invariant of an automaton to be any set of states of the automaton that is a superset of the reachable states of the automaton. Equivalently, an invariant could be defined to be a state predicate that is satisfied for all reachable states of the automaton.

We use the following notational convention: if R is a relation over $S_1 \times S_2$ and $s_1 \in S_1$, then $R[s_1]$ denotes the set $\{s_2 \in S_2 \mid (s_1, s_2) \in R\}$.

Definition 6.1 (Forward simulation)

Let A and B be automata with the same external actions and with invariants I_A and I_B, respectively. A *forward simulation* from A to B, with respect to I_A and I_B, is a relation f over $states(A) \times states(B)$ that satisfies:

1. If $s \in start(A)$ then $f[s] \cap start(B) \neq \emptyset$.

2. If $(s, a, s') \in steps(A)$, $s, s' \in I_A$, and $u \in f[s] \cap I_B$, then there exists an $\alpha \in frag^*(B)$ with $fstate(\alpha) = u$, $lstate(\alpha) \in f[s']$, and $trace(\alpha) = trace(a)$.

Write $A \leq_F B$ if there exists a forward simulation from A to B with respect to some invariants I_A and I_B. If f is a forward simulation from A to B with respect to some invariants I_A and I_B, write $A \leq_F B$ via f. ■

A refinement mapping is a special case of a forward simulation where the relation is a function. Because of its practical importance (cf. [AL91a]) an explicit definition is given.

Definition 6.2 (Refinement mapping)

Let A and B be automata with the same external actions and with invariants I_A and I_B, respectively. A *refinement mapping* from A to B, with respect to I_A and I_B, is a function r from $states(A)$ to $states(B)$ that satisfies:

1. If $s \in start(A)$ then $r(s) \in start(B)$.

2. If $(s, a, s') \in steps(A)$, $s, s' \in I_A$, and $r(s) \in I_B$, then there exists an $\alpha \in frag^*(B)$ with $fstate(\alpha) = r(s)$, $lstate(\alpha) = r(s')$, and $trace(\alpha) = trace(a)$.

Write $A \leq_R B$ if there exists a refinement mapping from A to B with respect to some invariants I_A and I_B. If r is a refinement mapping from A to B with respect to some invariants I_A and I_B, write $A \leq_R B$ via r. ■

In a forward simulation there has to be a sequence of abstract steps starting from *any* of the abstract states related to the concrete pre-state (restricted to the invariant) and ending in *some* state related to the concrete post-state. The word "forward" thus refers to the fact that the abstract sequence of steps is constructed from any possible pre-state in a forward direction toward the set of possible post-states.

In a backward simulation, on the other hand, there has to be a sequence of abstract steps *ending* in *any* state related to the concrete post-state (restricted to the invariant) and starting in *some* state related to the concrete pre-state. In other words, the sequence of abstract steps is constructed given a post-state rather than a pre-state as in the forward simulation. Thus, in a backward simulation the steps are constructed in a backward direction.

We need the following definition of image-finiteness for the definition of a backwards simulation. A relation R over $S_1 \times S_2$ is *image-finite* if for each $s_1 \in S_1$, $R[s_1]$ is a finite set.

Definition 6.3 (Backward simulation)

Let A and B be automata with the same external actions and with invariants I_A and I_B, respectively. A *backward simulation* from A to B, with respect to I_A and I_B, is a relation b over $states(A) \times states(B)$ that satisfies:

1. If $s \in I_A$ then $b[s] \cap I_B \neq \emptyset$.
2. If $s \in start(A)$ then $b[s] \cap I_B \subseteq start(B)$.
3. If $(s, a, s') \in steps(A)$, $s, s' \in I_A$, and $u' \in b[s'] \cap I_B$, then there exists an $\alpha \in frag^*(B)$ with $lstate(\alpha) = u'$, $fstate(\alpha) \in b[s] \cap I_B$, and $trace(\alpha) = trace(a)$.

Write $A \leq_B B$ if there exists a backward simulation from A to B with respect to some invariants I_A and I_B. Furthermore, if the backward simulation is image-finite, write $A \leq_{iB} B$. If b is a backward simulation from A to B with respect to some invariants I_A and I_B, write $A \leq_B B$ (or $A \leq_{iB} B$ when b is image-finite) via b. ■

In [LV93a] abstract notions of *history variables* [OG76] and *prophecy variables* [AL91a] are given in terms of *history relations* and *prophecy relations*.

Definition 6.4 (History relation)

Let A and B be automata with the same external actions and with invariants I_A and I_B, respectively. A relation h over $states(A) \times states(B)$ is a *history relation* from A to B, with respect to I_A and I_B, if h is a forward simulation from A to B with respect to I_A and I_B, and h^{-1} is a refinement mapping from B to A, with respect to I_B and I_A.

Write $A \leq_H B$ if there exists a history relation from A to B with respect to some invariants I_A and I_B. If h is a history relation from A to B with respect to some invariants I_A and I_B, write $A \leq_H B$ via h. ■

Definition 6.5 (Prophecy relation)

Let A and B be automata with the same external actions and with invariants I_A and I_B, respectively. A relation p over $states(A) \times states(B)$ is a *prophecy relation* from A to B, with respect to I_A and I_B, if p is a backward simulation from A to B with respect to I_A and I_B, and p^{-1} is a refinement mapping from B to A, with respect to I_A and I_B.

Write $A \leq_P B$ if there exists a prophecy relation from A to B with respect to some invariants I_A and I_B. Furthermore, if the prophecy relation is image-finite, write $A \leq_{iP} B$. If p is a prophecy relation from A to B with respect to some invariants I_A and I_B, write $A \leq_P B$ (or $A \leq_{iP} B$ when p is image-finite) via p. ■

6.1.2 Execution Correspondence

This subsection introduces and proves the Execution Correspondence Theorem (ECT). The ECT states that if any of the simulations defined in the previous section has been proven from an implementation automaton to a specification automaton, then for any execution of the implementation automaton, there is a "corresponding" execution of the specification automaton. In order to formalize this notion of correspondence, the notions of *R-relation* and *index mapping* are introduced.

Definition 6.6 (R-relation and index mappings)

Let A and B be automata with the same external actions and let R be a relation over $states(A) \times states(B)$. Furthermore, let α and α' be executions of A and B, respectively:

$$\begin{aligned} \alpha &= s_0 a_1 s_1 a_2 s_2 \cdots \\ \alpha' &= u_0 b_1 u_1 b_2 u_2 \cdots \end{aligned}$$

Say that α and α' are *R-related*, written $(\alpha, \alpha') \in R$, if there exists a total, nondecreasing mapping $m : \{0, 1, \ldots, |\alpha|\} \to \{0, 1, \ldots, |\alpha'|\}$ such that

1. $m(0) = 0$,
2. $(s_i, u_{m(i)}) \in R$ for all $0 \leq i \leq |\alpha|$,
3. $trace(b_{m(i-1)+1} \cdots b_{m(i)}) = trace(a_i)$ for all $0 < i \leq |\alpha|$, and
4. for all j, $0 \leq j \leq |\alpha'|$, there exists an i, $0 \leq i \leq |\alpha|$, such that $m(i) \geq j$.

The mapping m is referred to as an *index mapping* from α to α' with respect to R.

Write $(A, B) \in R$ if for every execution α of A, there exists an execution α' of B such that $(\alpha, \alpha') \in R$. ■

Thus, an index mapping maps indices of states in the concrete execution to indices of states in the abstract execution. Effectively, an index mapping maps concrete states to corresponding abstract states in such a way that the start states correspond (Condition 1), corresponding states are related by R (Condition 2), and the external actions between two consecutive pairs of corresponding states are the same at the concrete and the abstract level (Condition 3). Condition 4 ensures that the abstract execution (α') is not "too long", i.e., α' must not extend beyond the last state of α' corresponding to some state in α (if such a state exists). Note, that if α is finite, then α' must also be finite. However, even if α is infinite, α' can be finite if the index mapping is constant for indices above some bound.

In order to prove the ECT, two auxiliary lemmas are needed. The first, Lemma 6.7, deals with forward simulations; the second, Lemma 6.10, deals with backward simulations.

Lemma 6.7

Let A and B be automata with the same external actions and assume $A \leq_F B$ via f. Furthermore, let α be an arbitrary execution of A. Then there exists a collection $(\alpha'_i, m_i)_{0 \leq i \leq |\alpha|}$ of finite executions of B and mappings such that

1. *m_i is an index mapping from $\alpha|_i$ to α'_i with respect to f, for all $0 \leq i \leq |\alpha|$, and*
2. *$\alpha'_{i-1} \leq \alpha'_i$ and $m_{i-1} = m_i \restriction \{0, \ldots, i-1\}$, for all $0 < i \leq |\alpha|$.*

Proof. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ and let I_A and I_B be invariants of A and B, respectively, such that f is a forward simulation from A to B with respect to I_A and I_B. Construct α'_i and m_i inductively.

Since $s_0 \in start(A)$, Condition 1 of Definition 6.1 of a forward simulation gives the existence of a state $u_0 \in start(B)$ such that $(s_0, u_0) \in f$. Let $\alpha'_0 = u_0$ and let m_0 be the mapping that maps 0 to 0. Then clearly m_0 is an index mapping from $\alpha|_0$ to α'_0 with respect to f.

Now assume m_{i-1} (for $0 < i \leq |\alpha|$) is an index mapping from $\alpha|_{i-1}$ to α'_{i-1} with respect to f. Let $u = lstate(\alpha'_{i-1})$. Then, by definition of m_{i-1}, $m_{i-1}(i-1) = |\alpha'_{i-1}|$ and $(s_{i-1}, u) \in f$. Since $(s_{i-1}, a_i, s_i) \in steps(A)$, and s_{i-1}, s_i, and u are are reachable (by definition since they occur in an execution) and therefore satisfy their respective invariants, Condition 2 of Definition 6.1 (Forward simulation) gives the existence of a finite execution fragment α'' of B which starts in u and ends in a state u' with $(s_i, u') \in f$, such that $trace(\alpha'') = trace(a_i)$. Now define $\alpha'_i = \alpha'_{i-1} \frown \alpha''$ and define m_i to be the mapping such that $m_i(j) = m_{i-1}(j)$ for all $0 \leq j \leq i-1$ and $m_i(i) = |\alpha'_i|$. Then, m_i is trivially an index mapping from $\alpha|_i$ to α'_i with respect to f, and Part 2 of the lemma holds by construction. ∎

In order to state an analogous lemma for backward simulations, the notion of *induced digraph* is introduced along with some lemmas giving properties of the induced digraph.

Definition 6.8 (Induced digraph)

Let A and B be automata with the same external actions and assume $A \leq_{iB} B$ via b with respect to some invariants I_A and I_B. For any execution $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ of A, let the digraph *induced* by α, b, and I_B be the digraph G constructed as follows:

- The nodes of G are the pairs (u, i) where $0 \leq i \leq |\alpha|$ and $u \in b[s_i] \cap I_B$.

- There is an edge from (u, i) to (u', i') exactly if $i' = i+1$ and there exists a finite execution fragment α' of B such that $fstate(\alpha') = u$, $lstate(\alpha') = u'$, and $trace(\alpha') = trace(a_{i+1})$. ■

Lemma 6.9

Let A and B be automata with the same external actions and assume $A \leq_{iB} B$ via b with respect to some I_A and I_B. Furthermore, let α be any execution of A. Then the digraph G induced by α, b, and I_B satisfies:

1. *For each $0 \leq i \leq |\alpha|$, there are nodes of the form (u, i).*
2. *Exactly all nodes of the form $(u, 0)$ are roots.*
3. *G has finitely many roots.*
4. *Each node of G has finite outdegree.*
5. *Each node of G is reachable from some root of G.*

Proof. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$.

1. Since each state s_i in α is reachable (by definition) and thus belongs to I_A, Condition 1 of Definition 6.3 (Backward simulation) gives us that $b[s_i] \cap I_B \neq \emptyset$. Thus, by Definition 6.8, G has nodes of the form (u, i).
2. Any node $(u, 0)$ is a root in G. Consider any node (u, i) with $i > 0$. Then since $u \in b[s_i] \cap I_B$, $s_{i-1}, s_i \in I_A$, and $(s_{i-1}, a_i, s_i) \in steps(A)$, Definition 6.3 implies the existence of a finite execution fragment α' of B with $lstate(\alpha) = u$, $trace(\alpha') = trace(a_i)$, and $fstate(\alpha) \in b[s_{i-1}] \cap I_B$. Then by Definition 6.8 there is an edge in G from $(fstate(\alpha), i-1)$ to (u, i). Thus, (u, i) is not a root in G.
3. Since b is image-finite, the set $b[s_0] \cap I_B$ is finite and the result follows.
4. From any node (u, i), there can only be edges to nodes of the form $(u', i+1)$. Again, since b is image-finite, there are only finitely many such nodes.
5. Any node of the form $(u, 0)$ is reachable. Assume all nodes of the form (u, i) are reachable (for some $0 \leq i < |\alpha|$). By an argument similar to Point 2 above, it is seen that to any node of the form $(u', i+1)$, there is an edge from a node of the form (u, i). Thus, any node of the form $(u', i+1)$ is reachable. By induction all nodes of G are reachable. ■

Lemma 6.10

Let A and B be automata with the same external actions and assume $A \leq_{iB} B$ via b. Furthermore, let α be an arbitrary execution of A. Then there exists a collection $(\alpha'_i, m_i)_{0 \leq i \leq |\alpha|}$ of finite executions of B and mappings such that

1. *m_i is an index mapping from $\alpha|_i$ to α'_i with respect to b, for all $0 \leq i \leq |\alpha|$, and*

2. $\alpha'_{i-1} \leq \alpha'_i$ *and* $m_{i-1} = m_i \restriction \{0, \ldots, i-1\}$*, for all* $0 < i \leq |\alpha|$*.*

Proof. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ and let I_A and I_B be invariants of A and B, respectively, such that b is an image-finite backward simulation from A to B with respect to I_A and I_B. Furthermore, let G be the digraph induced by α, b, and I_B. If α is finite, fix p to be any path $(u_0, 0)(u_1, 1) \cdots (u_n, n)$, where $n = |\alpha|$. Such a path exists by Condition 5 of Lemma 6.9. If α is infinite, then G is infinite and Lemmas 6.9 and 2.1 (König's Lemma) imply the existence of an infinite path in G. Fix $p = (u_0, 0)(u_1, 1) \cdots$ to be any such path. Now construct α'_i and m_i inductively. At the same time prove that $\mathit{lstate}(\alpha'_i) = u_i$.

Since $s_0 \in \mathit{start}(A)$ and $u_0 \in b[s_0] \cap I_B$, Condition 2 of Definition 6.3 of a backward simulation implies that $u_0 \in \mathit{start}(B)$. Let $\alpha'_0 = u_0$ and let m_0 be the mapping that maps 0 to 0. Then clearly m_0 is an index mapping from $\alpha|_0$ to α'_0 with respect to b, and $\mathit{lstate}(\alpha'_0) = u_0$.

Now assume m_{i-1} (for $0 < i \leq |\alpha|$) is an index mapping from $\alpha|_{i-1}$ to α'_{i-1} with respect to b and assume that $\mathit{lstate}(\alpha'_{i-1}) = u_{i-1}$. Since there is an edge in G from $(u_{i-1}, i-1)$ to (u_i, i), there exists, by Definition 6.8, a finite execution fragment α'' of B such that $\mathit{fstate}(\alpha'') = u_{i-1}$, $\mathit{lstate}(\alpha'') = u_i$, and $\mathit{trace}(\alpha'') = \mathit{trace}(a_i)$. Now define $\alpha'_i = \alpha'_{i-1} \frown \alpha''$ and define m_i to be the mapping such that $m_i(j) = m_{i-1}(j)$ for all $0 \leq j \leq i-1$ and $m_i(i) = |\alpha'_i|$. Then, trivially m_i is an index mapping from $\alpha|_i$ to α'_i with respect to b, and Point 2 of the lemma holds by construction. Also, $\mathit{lstate}(\alpha'_i) = u_i$ as required.

If α is finite, then the lemma holds by construction; if α is infinite, then the lemma holds by induction. ∎

Finally, the Execution Correspondence Theorem can be stated and proven. The theorem states that if a relation S is a forward simulation, refinement mapping, image-finite backward simulation, history relation, or image-finite prophecy relation from A to B, then for any execution of A, there exists an S-related execution of B.

Theorem 6.11 (Execution Correspondence Theorem)

Let A and B be automata with the same external actions. Assume for $X \in \{F, R, iB, H, iP\}$ that $A \leq_X B$ via S. Then $(A, B) \in S$.

Proof. One must show that for all $\alpha \in \mathit{exec}(A)$ there exists an $\alpha' \in \mathit{exec}(B)$ such that $(\alpha, \alpha') \in S$. Consider cases.

1. $A \leq_F B$ via S.

   Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ be an arbitrary execution of A, and let $(\alpha'_i, m_i)_{0 \leq i \leq |\alpha|}$ be a collection of finite executions of B and mappings as defined in Lemma 6.7.

   First assume α is finite. Then $\alpha = \alpha|_{|\alpha|}$, and according to Lemma 6.7 $m_{|\alpha|}$ is an index mapping from $\alpha|_{|\alpha|}$ to $\alpha'_{|\alpha|}$. That suffices since $\alpha'_{|\alpha|} = \alpha'$ by Condition 4 of Definition 6.6

   Now, assume α is infinite. Then let m be the unique mapping over the natural numbers defined by $m(i) = m_i(i)$, and let α' be the limit of α'_i under the prefix ordering. Thus,

α' is the unique execution of B defined by $\alpha'|_{m(i)} = \alpha'_i$ with the restriction that for any index j of α' there exists an i such that $\alpha'|_j \leq \alpha'_i$.

Now the claim is that m is an index mapping from α to α' with respect to S. First note that m is total and nondecreasing. The latter is seen by contradiction. Assume m is not nondecreasing. Then there exists an i such that $m(i) < m(i-1)$, but since $m(i) = m_i(i)$ and $m(i-1) = m_{i-1}(i-1) = m_i(i-1)$ this contradicts the fact that m_i is an index mapping and thus is nondecreasing. Similarly, it can be seen that the range of m is within $\{0, \ldots, |\alpha'|\}$.
Now the four conditions of Definition 6.6 must be checked. Condition 1 holds since m_0 is an index mapping and thus satisfies $m_0(0) = 0$. Assume Condition 2 or 3 does not hold. Then there must exist an i such that the condition is invalidated. However, this contradicts the fact that for any i, m_i is an index mapping from $\alpha|_i$ to α'_i with respect to S. Finally, assume Condition 4 does not hold. Thus, assume the existence of an index j in α' such that for all i, $m(i) < j$. By definition of α' there exists an i such that $\alpha'|_j \leq \alpha'_i$. Now, Lemma 6.7 gives that $m_i(i) = |\alpha'_i| \geq j$. Thus, $m(i) \geq j$ which contradicts the assumption that $m(i) < j$.

2. $A \leq_R B$ via S.

   A refinement mapping is a forward simulation, so the result follows from the previous case.

3. $A \leq_{iB} B$ via S.

   Same as case 1, by using Lemma 6.10 instead of Lemma 6.7.

4. $A \leq_H B$ via S.

   By Definition 6.4 S is a forward simulation from A to B, so the result follows from case 1 above.

5. $A \leq_{iP} B$ via S.

   By Definition 6.5 S is an image-finite backward simulation from A to B, so the result follows from case 3 above. ■

6.1.3 Proving the Safe Preorder

This subsection proves the soundness of the simulation proof techniques with respect to the safe preorder. This is a well-known result, see, e.g., [LV93a], however, instead of proving the result directly as in [LV93a], the ECT is used. We start with some preliminary definitions and technical lemmas needed for the proof.

Define the ith *step* of α, for all $0 < i \leq |\alpha|$, as $step(\alpha, i) \triangleq {}_{i-1}|\alpha|_i = (s_{i-1}, a_i, s_i)$. Also, let m be a total, nondecreasing mapping $m : \{0, 1, \ldots, N\} \to \{0, 1, \ldots, |\alpha|\}$, where $N \in \mathbb{N}_0 \cup \{\infty\}$. Then, define the ith *m-step* of α, for all $0 < i \leq N$, as $step_m(\alpha, i) \triangleq {}_{m(i-1)}|\alpha|_{m(i)}$.

Lemma 6.12

Let α be an execution fragment.

1. *Then, for all $0 \leq i < j \leq |\alpha|$,*

$$_i|\alpha|_j = \mathit{step}(\alpha, i+1) \frown \mathit{step}(\alpha, i+2) \frown \cdots \frown \mathit{step}(\alpha, j)$$

2. *Let m be a total, nondecreasing mapping $m : \{0, 1, \ldots, N\} \to \{0, 1, \ldots, |\alpha|\}$, where $N \in \mathbb{N}_0 \cup \{\infty\}$. Then, for all $0 \leq i < j \leq N$,*

$$_{m(i)}|\alpha|_{m(j)} = \mathit{step}_m(\alpha, i+1) \frown \mathit{step}_m(\alpha, i+2) \frown \cdots \frown \mathit{step}_m(\alpha, j)$$

Proof. Trivial by explicit construction. ■

Lemma 6.13

Let α be an execution fragment.

1. *Then, for all $0 \leq i < |\alpha|$,*

$$_i|\alpha = \begin{cases} \mathit{step}(\alpha, i+1) \frown \mathit{step}(\alpha, i+2) \frown \cdots \frown \mathit{step}(\alpha, |\alpha|) & \text{if } \alpha \text{ is finite} \\ \mathit{step}(\alpha, i+1) \frown \mathit{step}(\alpha, i+2) \frown \cdots & \text{otherwise} \end{cases}$$

2. *Let m be a total, nondecreasing mapping $m : \{0, 1, \ldots, N\} \to \{0, 1, \ldots, |\alpha|\}$, where $N \in \mathbb{N}_0 \cup \{\infty\}$, such that for all $0 \leq j \leq |\alpha|$ there exists an $i \in \mathit{dom}(m)$ with $m(i) \geq j$. Then, for all $0 \leq i < N$,*

$$_{m(i)}|\alpha = \begin{cases} \mathit{step}_m(\alpha, i+1) \frown \mathit{step}_m(\alpha, i+2) \frown \cdots \frown \mathit{step}_m(\alpha, N) & \text{if } N \text{ is finite} \\ \mathit{step}_m(\alpha, i+1) \frown \mathit{step}_m(\alpha, i+2) \frown \cdots & \text{otherwise} \end{cases}$$

Proof. The lemma follows from Lemma 6.12 ■

The following lemma is used to show that any two related executions have the same trace.

Lemma 6.14

Let A and B be automata with the same external actions and let R be a relation over $\mathit{states}(A) \times \mathit{states}(B)$. Assume that $(\alpha, \alpha') \in R$ and let m be any index mapping from α to α' with respect to R. Then, for all $0 \leq i \leq |\alpha|$, $\mathit{trace}({}_i|\alpha) = \mathit{trace}({}_{m(i)}|\alpha')$.

Proof. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ and $\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$. If $i = |\alpha|$ (in the case where α is finite), $_{|\alpha|}|\alpha = s_{|\alpha|}$ and Condition 4 of Definition 6.6 gives $_{m(|\alpha|)}|\alpha' = u_{m(|\alpha|)}$. Thus, obviously $\mathit{trace}({}_i|\alpha) = \mathit{trace}({}_{m(i)}|\alpha')$ (the empty list). If $0 \leq i < |\alpha|$, then from Lemma 6.13

$$\begin{aligned} {}_i|\alpha \quad &= \quad \mathit{step}(\alpha, i+1) \frown \mathit{step}(\alpha, i+2) \frown \cdots \\ {}_{m(i)}|\alpha' \quad &= \quad \mathit{step}_m(\alpha', i+1) \frown \mathit{step}_m(\alpha', i+2) \frown \cdots \end{aligned}$$

where the concatenations are finite (and end in $step(\alpha, |\alpha|)$ and $step_m(\alpha', |\alpha|)$, respectively) if and only if α is finite.

Now use the obvious fact that restricting an execution to a set of actions, distributes over concatenation. This gives us:

$$\begin{aligned}
trace({}_i|\alpha) &= ({}_i|\alpha) \restriction ext(A) \\
&= (step(\alpha, i+1) \restriction ext(A))\,\hat{}\,(step(\alpha, i+2) \restriction ext(A))\,\hat{}\cdots \\
&= trace(step(\alpha, i+1))\,\hat{}\,trace(step(\alpha, i+2))\,\hat{}\cdots
\end{aligned}$$

$$\begin{aligned}
trace({}_{m(i)}|\alpha') &= ({}_{m(i)}|\alpha') \restriction ext(B) \\
&= (step_m(\alpha', i+1) \restriction ext(B))\,\hat{}\,(step_m(\alpha', i+2) \restriction ext(B))\,\hat{}\cdots \\
&= trace(step_m(\alpha', i+1))\,\hat{}\,trace(step_m(\alpha', i+2))\,\hat{}\cdots
\end{aligned}$$

Now, from the definitions of $step$ and $step_m$ and Condition 3 of Definition 6.6

$$trace(step(\alpha, j)) = trace(step_m(\alpha', j))$$

for all $0 < j \leq |\alpha|$. So, if $|\alpha| \neq \infty$, $trace({}_i|\alpha) = trace({}_{m(i)}|\alpha')$ by construction. If $|\alpha| = \infty$, assume that $trace({}_i|\alpha) \neq trace({}_{m(i)}|\alpha')$. Then there must be a finite prefix β of $trace({}_i|\alpha)$ such that $\beta \not\leq trace({}_{m(i)}|\alpha')$. Also, there must exist a finite number $j > i$ such that

$$\beta \leq \beta_1 = trace(step(\alpha, i+1))\,\hat{}\cdots\hat{}\,trace(step(\alpha, j))$$

Since $\beta \not\leq trace({}_{m(i)}|\alpha')$, it must also be the case that $\beta_1 \not\leq trace({}_{m(i)}|\alpha')$. Now, let

$$\beta' = trace(step_m(\alpha', i+1))\,\hat{}\cdots\hat{}\,trace(step_m(\alpha', j))$$

Then, $\beta' \leq trace({}_{m(i)}|\alpha')$ and by distributivity of restriction over concatenation $\beta_1 = \beta'$. Thus $\beta_1 \leq trace({}_{m(i)}|\alpha')$, which contradicts the assumption. So, also if $|\alpha| = \infty$ conclude that $trace({}_i|\alpha) = trace({}_{m(i)}|\alpha')$. ■

Lemma 6.15

Let A and B be automata with the same external actions and let R be a relation over $states(A) \times states(B)$. If $(\alpha, \alpha') \in R$, then $trace(\alpha) = trace(\alpha')$.

Proof. Immediate from Lemma 6.14 since for any execution α_1, ${}_0|\alpha_1 = \alpha_1$, and any index mapping maps 0 to 0 (cf. Condition 1 of Definition 6.6). ■

The soundness of the simulation relations with respect to trace inclusion can now be shown.

Lemma 6.16 (Soundness of simulations w.r.t. trace inclusion)

Let A and B be automata with the same external actions and assume for $X \in \{F, R, iB, H, iP\}$ that $A \leq_X B$ via S. Then $traces(A) \subseteq traces(B)$.

Proof. Let $\beta \in traces(A)$ be an arbitrary trace of A and let α be an execution of A such that $trace(\alpha) = \beta$. Then, by Theorem 6.11 (ECT), there exists an execution α' of B such that $(\alpha, \alpha') \in S$. By Lemma 6.15, $trace(\alpha') = trace(\alpha) = \beta$. Thus, $\beta \in traces(B)$ as required. ∎

Finally, it follows immediately from the fact that the simulation relations are sound with respect to trace inclusion (Lemma 6.16) and the definition of the safe preorder (Definition 3.30) that the simulation relations are sound with respect to the safe preorder

Theorem 6.17 (Soundness of simulations w.r.t. the safe preorder)
Let (A, L) and (B, M) be live I/O automata with $esig(A) = esig(B)$, and assume for some $X \in \{F, R, iB, H, iP\}$ that $A \leq_X B$. Then $(A, L) \sqsubseteq_{\mathrm{S}} (B, M)$. ∎

6.1.4 Proving the Live Preorder

A proof strategy for proving that one live I/O automaton implements another live I/O automaton via the live preorder is now described. First consider the following lemma.

Lemma 6.18
Suppose (A, L) and (B, M) are live I/O automata with $esig(A) = esig(B)$, and assume for some $X \in \{F, R, iB, H, iB\}$ that $A \leq_X B$ via S. If

$$\forall\ (\alpha, \alpha') \in S : (\alpha \in L \Longrightarrow \alpha' \in M)$$

then $(A, L) \sqsubseteq_{\mathrm{L}} (B, M)$.

Proof. It is enough to show that $traces(L) \subseteq traces(M)$. Let $\beta \in traces(L)$. By definition of trace there is an execution α of L such that $trace(\alpha) = \beta$. By definition of live I/O automaton α is an execution of A. From Theorem 6.11 (ECT) there exists an execution α' of B such that $(\alpha, \alpha') \in S$. From the hypothesis of this lemma α' is an execution of M. Moreover, from Lemma 6.15 we have $trace(\alpha) = trace(\alpha')$. Thus, $\beta \in traces(M)$. ∎

Based on Lemma 6.18, the following proof strategy proves that a live I/O automaton (A, L) is a correct implementation of another live I/O automaton (B, M):

1. Prove a simulation S from A to B with respect to some invariants.
2. Assume α and α' are arbitrary executions of A and B, respectively, and assume that α is live (i.e., $\alpha \in L$).

   Prove that α' is also live (i.e., $\alpha' \in M$).

   This will usually be a proof by contradiction. That is, assume that α' is not live and show that this leads to a contradiction. This strategy gives a nice way of splitting the proof into cases since being live usually means satisfying a conjunction of condition such that not being live means not satisfying one (at least) of these conditions. Thus, each of the conditions can be considered separately.

The reader is referred to [SLL93, Lyn93] for extensive applications of the proof techniques.

6.2 Timed Proof Techniques

Since liveness conditions in the timed model are expressed in terms of *timed* executions, the obvious generalization of the approach taken in the untimed model would be to develop simulation techniques that give a correspondence between the timed executions of timed automata. This suggests that the simulation techniques in the timed model should for every "timed step" (ω, a, ω') of a low-level timed automaton, where ω and ω' are trajectories, find a corresponding timed execution fragment of the high-level timed automaton. On the other hand, the fact that the transition relation of a timed automaton determines ordinary steps of the form (s, a, s'), rather than steps of timed executions of the form (ω, a, ω'), suggests simulation techniques that for each ordinary step, (s, a, s'), of the low-level timed automaton find a corresponding (ordinary) execution fragment of the high-level timed automaton. We pursue this latter type of simulation.

In particular, this section shows that the existence of such a simulation, based on ordinary steps, between two timed automata implies all four of the timed safe preorders of the timed model (cf. Definition 4.36). Also, (timed) liveness conditions can be *characterized* by sets of ordinary (sampled) executions some of which are *minimal*. These characterizations by sets of ordinary (sampled) executions form the basis of a lemma similar to Lemma 6.18 on which proofs of the timed live preorder can be based.

The structure of this section parallel that of the untimed model. First a number of (timed) simulation techniques are defined. Then, the *execution correspondence theorem* for the timed model is proven, and finally the use of the timed simulation techniques to prove the timed safe and live preorders is discussed.

6.2.1 Timed Simulation Proof Techniques

The timed simulations presented here are similar to the ones defined in [LV91] except for our use of invariants. Recall, that an invariant is any set of states of an automaton that is a superset of the reachable states (reachability coincides with t-reachability).

There are only two minor differences between the simulation relations presented here and the simulation relations from the untimed model. First, states related by a simulation relation must have the same time. Second, since the *trace* operator on execution fragments does not adequately abstract from time-passage actions, the simulation techniques below use a notion of *visible trace*. For any timed automaton A and any execution fragment α of A, define the visible trace of α, written $\textit{vis-trace}_A(\alpha)$, or just $\textit{vis-trace}(\alpha)$ when A is clear from context, to be $\alpha \restriction \textit{vis}(A)$. Similarly, given any sequence of actions β, define the visible trace of β, written $\textit{vis-trace}_A(\beta)$, or just $\textit{vis-trace}(\beta)$ if A is clear from context, to be $\beta \restriction \textit{vis}(A)$.

Definition 6.19 (Timed forward simulation)

Let A and B be timed automata with the same visible actions and with invariants I_A and I_B, respectively. A *timed forward simulation* from A to B, with respect to I_A and I_B, is a relation

f over $states(A) \times states(B)$ that satisfies:

1. If $u \in f[s]$ then $u.now = s.now$.
2. If $s \in start(A)$ then $f[s] \cap start(B) \neq \emptyset$.
3. If $(s, a, s') \in steps(A)$, $s, s' \in I_A$, and $u \in f[s] \cap I_B$, then there exists an $\alpha \in frag^*(B)$ with $fstate(\alpha) = u$, $lstate(\alpha) \in f[s']$, and $vis\text{-}trace(\alpha) = vis\text{-}trace(a)$.

Write $A \leq_{tF} B$ if there exists a timed forward simulation from A to B with respect to some invariants I_A and I_B. If f is a timed forward simulation from A to B with respect to some invariants I_A and I_B, write $A \leq_{tF} B$ via f. ■

Definition 6.20 (Timed refinement mapping)

Let A and B be timed automata with the same visible actions and with invariants I_A and I_B, respectively. A *timed refinement mapping* from A to B, with respect to I_A and I_B, is a function r from $states(A)$ to $states(B)$ that satisfies:

1. $r(s).now = s.now$.
2. If $s \in start(A)$ then $r(s) \in start(B)$.
3. If $(s, a, s') \in steps(A)$, $s, s' \in I_A$, and $r(s) \in I_B$, then there exists an $\alpha \in frag^*(B)$ with $fstate(\alpha) = r(s)$, $lstate(\alpha) = r(s')$, and $vis\text{-}trace(\alpha) = vis\text{-}trace(a)$.

Write $A \leq_{tR} B$ if there exists a timed refinement mapping from A to B with respect to some invariants I_A and I_B. If r is a timed refinement mapping from A to B with respect to some invariants I_A and I_B, write $A \leq_{tR} B$ via r. ■

Definition 6.21 (Timed backward simulation)

Let A and B be timed automata with the same visible actions and with invariants I_A and I_B, respectively. A *timed backward simulation* from A to B, with respect to I_A and I_B, is a relation b over $states(A) \times states(B)$ that satisfies:

1. If $u \in b[s]$ then $u.now = s.now$.
2. If $s \in I_A$ then $b[s] \cap I_B \neq \emptyset$.
3. If $s \in start(A)$ then $b[s] \cap I_B \subseteq start(B)$.
4. If $(s, a, s') \in steps(A)$, $s, s' \in I_A$, and $u' \in b[s'] \cap I_B$, then there exists an $\alpha \in frag^*(B)$ with $lstate(\alpha) = u'$, $fstate(\alpha) \in b[s] \cap I_B$, and $vis\text{-}trace(\alpha) = vis\text{-}trace(a)$.

Write $A \leq_{tB} B$ if there exists a timed backward simulation from A to B with respect to some invariants I_A and I_B. If furthermore the timed backward simulation is image-finite, write $A \leq_{itB} B$. If b is a timed backward simulation from A to B with respect to some invariants I_A and I_B, write $A \leq_{tB} B$ (or $A \leq_{itB} B$ when b is image-finite) via b. ■

Definition 6.22 (Timed history relation)

Let A and B be timed automata with the same visible actions and with invariants I_A and I_B, respectively. A relation h over $states(A) \times states(B)$ is a *timed history relation* from A to B, with respect to I_A and I_B, if h is a timed forward simulation from A to B with respect to I_A and I_B, and h^{-1} is a timed refinement mapping from B to A, with respect to I_B and I_A.

Write $A \leq_{tH} B$ if there exists a timed history relation from A to B with respect to some invariants I_A and I_B. If h is a timed history relation from A to B with respect to some invariants I_A and I_B, write $A \leq_{tH} B$ via h. ■

Definition 6.23 (Timed prophecy relation)

Let A and B be timed automata with the same visible actions and with invariants I_A and I_B, respectively. A relation p over $states(A) \times states(B)$ is a *timed prophecy relation* from A to B, with respect to I_A and I_B, if p is a timed backward simulation from A to B with respect to I_A and I_B, and p^{-1} is a timed refinement mapping from B to A, with respect to I_B and I_A.

Write $A \leq_{tP} B$ if there exists a timed prophecy relation from A to B with respect to some invariants I_A and I_B. If furthermore the timed prophecy relation is image-finite, write $A \leq_{itP} B$. If p is a prophecy relation from A to B with respect to some invariants I_A and I_B, write $A \leq_{tP} B$ (or $A \leq_{itP} B$ when p is image-finite) via p. ■

6.2.2 Execution Correspondence

As in the untimed model, the simulation relations imply a certain correspondence between the ordinary executions of the involved timed automata. The following definition formalizes this correspondence, called *timed R-relation*, and defines a notion of *timed index mapping*. The definition is similar to Definition 6.6 in the untimed model; the only differences are that the R relation must relate states with the same time and that the definition deals with visible traces as opposed to traces, i.e., the same differences as in the simulation relations.

Definition 6.24 (Timed R-relation and timed index mappings)

Let A and B be timed automata with the same external actions and let R be a relation over $states(A) \times states(B)$ such that if $(s, u) \in R$ then $s.now = u.now$. Furthermore, let α and α' be (ordinary) executions of A and B, respectively.

$$\begin{aligned} \alpha &= s_0 a_1 s_1 a_2 s_2 \cdots \\ \alpha' &= u_0 b_1 u_1 b_2 u_2 \cdots \end{aligned}$$

Let α and α' be *timed R-related*, written $(\alpha, \alpha') \in_t R$, if there exists a total, nondecreasing mapping $m : \{0, 1, \ldots, |\alpha|\} \to \{0, 1, \ldots, |\alpha'|\}$ such that

1. $m(0) = 0$,
2. $(s_i, u_{m(i)}) \in R$ for all $0 \leq i \leq |\alpha|$,
3. $\textit{vis-trace}(b_{m(i-1)+1} \cdots b_{m(i)}) = \textit{vis-trace}(a_i)$ for all $0 < i \leq |\alpha|$, and
4. for all j, $0 \leq j \leq |\alpha'|$, there exists an i, $0 \leq i \leq |\alpha|$, such that $m(i) \geq j$.

The mapping m is referred to as a *timed index mapping* from α to α' with respect to R.

Write $(A, B) \in_t R$ if for every execution α of A, there exists an execution α' of B such that $(\alpha, \alpha') \in_t R$. ■

The following lemma shows that timed R-related executions have the same limit time and that there is a correspondence with respect to finiteness, admissibility, and Zenoness.

Lemma 6.25

Let A and B be timed automata with the same external actions and let R be a relation over $states(A) \times states(B)$ such that if $(s, u) \in R$ then $s.now = u.now$. Furthermore, let α and α' be executions of A and B, respectively. Then, if $(\alpha, \alpha') \in_t R$

1. *$ltime(\alpha) = ltime(\alpha')$,*
2. *if α is finite then α' is finite,*
3. *α is admissible iff α' is admissible, and*
4. *if α' is Zeno then α is Zeno.*

Proof. Let $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ and $\alpha' = u_0 b_1 u_1 b_2 u_2 \cdots$, and assume $(\alpha, \alpha') \in_t R$. Let m be a timed index mapping from α to α with respect to R. The four parts of the lemma are considered separately.

1. For any state s in α (and thus any time in α) there exists, by Condition 2 of Definition 6.24, a state u in α' with $(s, u) \in R$, and thus $u.now = s.now$. This proves that $ltime(\alpha) \leq ltime(\alpha')$. Similarly Condition 4 of Definition 6.24 implies that $ltime(\alpha) \geq ltime(\alpha')$. Thus, $ltime(\alpha) = ltime(\alpha')$.
2. Assume α is finite. Now, assume that α' is not finite. Let $m' = m(|\alpha|)$. Then, since α' is not finite and thus infinite, the state $u_{m'+1}$ exists in α'. Then Condition 4 of Definition 6.24 implies the existence of an index $0 \leq i \leq |\alpha|$ such that $m(i) > m'+1 > m'$, but this contradicts the fact that $m' = m(|\alpha|)$, and m is nondecreasing. Thus, α' is finite.
3. This result follows directly from Part 1 of this lemma.

4. Assume α' is Zeno. Then Part 2 of this lemma implies that α is not finite. Furthermore, Part 3 of this lemma implies that α is not admissible. Thus, α is Zeno.

This concludes the proof. ■

Note that in Part 2 of Lemma 6.25 the converse is not true: even though α' is finite, α could be Zeno by having a suffix containing only internal actions and having m be constant for all indices in that suffix. This argument shows that the converse of Part 4 is also not true in general.

Now the Execution Correspondence Theorem can be stated for the timed model.

Theorem 6.26 (Execution Correspondence Theorem)

Let A and B be timed automata with the same visible actions. Assume for $X \in \{tF, tR, itB, tH, itP\}$ that $A \leq_X B$ via S. Then $(A, B) \in_t S$.

Proof. Similar to the ECT proof in the untimed model (Theorem 6.11). ■

6.2.3 Proving the Timed Safe Preorders

Due to the fact that timed R-related executions have the same time in related states and have a correspondence between the their visible traces, it is possible to prove that timed R-related executions have the same timed traces.

Lemma 6.27

Let A and B be timed automata with the same external actions and let R be a relation over $states(A) \times states(B)$ such that if $(s, u) \in R$ then $s.now = u.now$. Then, if $(\alpha, \alpha') \in_t R$, then $t\text{-}trace(\alpha) = t\text{-}trace(\alpha')$.

Proof. Similar to the proofs of Lemmas 6.14 and 6.15. ■

Soundness of the timed simulations with respect to timed trace inclusion now follows.

Lemma 6.28 (Soundness of timed simulations w.r.t. timed trace inclusion)

Let A and B be timed automata with the same external actions. Assume for $X \in \{tF, tR, itB, tH, itP\}$ that $A \leq_X B$. Then

1. $t\text{-}traces(A) \subseteq t\text{-}traces(B)$
2. $t\text{-}traces^*(A) \subseteq t\text{-}traces^*(B)$
3. $t\text{-}traces^\infty(A) \subseteq t\text{-}traces^\infty(B)$

Proof. Consider the three parts separately.

1. Suppose $\gamma \in$ *t-traces*(A). Then by definition there exists a timed execution $\Sigma \in$ *t-exec*(A) such that *t-trace*$(\Sigma) = \gamma$. Now, the sampling results of Lemmas 4.4 and 4.7 imply the existence of an execution $\alpha \in exec(A)$ with *t-trace*$(\alpha) =$ *t-trace*(Σ). Then ECT (Theorem 6.26) and Lemma 6.27 imply the existence of an execution $\alpha' \in exec(B)$ such that *t-trace*$(\alpha') =$ *t-trace*(α). Finally, the sampling results of Lemmas 4.3 and 4.7 give the existence of a timed execution $\Sigma' \in$ *t-exec*(B) with *t-trace*$(\Sigma') =$ *t-trace*(α').

   Thus, *t-trace*$(\Sigma') =$ *t-trace*$(\alpha') =$ *t-trace*$(\alpha) =$ *t-trace*$(\Sigma) = \gamma$. Therefore $\gamma \in$ *t-traces*(B). That suffices.

2. Similar to Part 1. Also use Lemma 6.25 Part 2 and Lemma 4.6 Part 1 to prove the following: if Σ is finite then α, α', and Σ' are also finite. Then the result follows.

3. Similar to Part 2. Use Lemma 6.25 Part 3 and Lemma 4.6 Part 2. ■

Based on this lemma, the soundness of the timed simulations with respect to the timed safe preorders can be shown.

Theorem 6.29 (Soundness of timed simulations w.r.t. the timed safe preorders)
Let (A, L) and (B, M) be live timed I/O automata with $vsig(A) = vsig(B)$, and assume for some $X \in \{tF, tR, itB, tH, itP\}$ that $A \leq_X B$. Then

1. $A \sqsubseteq_{\mathrm{St}} B$
2. $A \sqsubseteq^{*}_{\mathrm{St}} B$
3. $A \sqsubseteq^{\infty}_{\mathrm{St}} B$
4. $A \sqsubseteq^{\mathrm{nz}}_{\mathrm{St}} B$

Proof. Parts 1–3 follow directly from Lemma 6.28 Parts 1–3 and the definition of the timed safe preorders (Definition 4.36). Part 4 follows, by Definition 4.36, from Parts 2 and 3. ■

6.2.4 Proving the Timed Live Preorder

It is possible to characterize timed liveness conditions by a set of ordinary executions such that a lemma like Lemma 6.18 (based on the timed simulation techniques above) can be stated. Start by defining such *(minimal) sampling characterizations* of liveness conditions.

Definition 6.30 ((Minimal) sampling characterizations)
Let (A, L) be a live timed I/O automaton. A set $L_0 \subseteq exec(A)$ is a *sampling characterization* of L if $L = \{\Sigma \in \textit{t-exec}^{\infty}(A) \mid \text{for all } \alpha \in exec(A) \text{ where } \alpha \text{ samples } \Sigma,\ \alpha \in L_0\}$.

Furthermore, L_0 is said to be *minimal* if it equals the set of all samplings of all timed executions in L. ■

For any live timed I/O automaton (A, L), L has a minimal sampling characterization L_0, namely the one containing all samplings of the timed executions in L.

Lemma 6.31

Let (A, L) and (B, M) be live timed I/O automata with $vsig(A) = vsig(B)$. Assume that L_0 and M_0 are sampling characterizations of L and M, respectively, and assume that M_0 is minimal. Assume for some $X \in \{tF, tR, itB, tH, itP\}$ that $A \leq_X B$ via S. If

$$\forall\ (\alpha, \alpha') \in_t S : (\alpha \in L_0 \Longrightarrow \alpha' \in M_0)$$

then $(A, L) \sqsubseteq_{\mathrm{Lt}} (B, M)$.

Proof. Let $\gamma \in \textit{t-trace}(L)$ be an arbitrary timed trace of L and let $\Sigma \in L$ with $\textit{t-trace}(\Sigma) = \gamma$. Based on the sampling result of Lemma 4.4 and the fact that L_0 is a sampling characterization of L, there exists an execution $\alpha \in exec(A)$ such that α samples Σ and $\alpha \in L_0$. Based on the sampling results of Lemmas 4.6 and 4.7, α is admissible and $\textit{t-trace}(\alpha) = \gamma$. Then by ECT (Theorem 6.26) there exists an $\alpha' \in exec(B)$ such that $(\alpha, \alpha') \in_t S$. Lemmas 6.25 and 6.27 imply that α' is admissible and $\textit{t-trace}(\alpha') = \gamma$. By the hypothesis in this lemma, $\alpha' \in M_0$. Then, based on the sampling results of Lemmas 4.3, 4.6, and 4.7, there exists a $\Sigma' \in \textit{t-exec}^{\infty}(B)$ with $\textit{t-trace}(\Sigma') = \gamma$. Now, since M_0 is a minimal sampling characterization of M, $\Sigma \in M$ and thus $\gamma \in \textit{t-traces}(M)$. By definition of $\sqsubseteq_{\mathrm{Lt}}$ (Definition 4.36) this suffices. ∎

Lemma 6.31 can be used to prove the live preorder between two live timed I/O automata in a manner similar to the way Lemma 6.18 is used in the untimed model. However, one must first find sampling characterizations of the liveness conditions. Furthermore, the sampling characterization for the high-level liveness condition must be minimal. In practice the liveness condition L of a live timed I/O automaton is often *defined* as those timed executions that have all their samplings in some set of ordinary executions L_0, which, in turn, could be those executions that satisfy some formula in a temporal logic. In this case L_0 is, by definition, a sampling characterization of L. Then, the only remaining proof obligation is to show that the sampling characterization of the high-level live timed I/O automaton is minimal. In [SLL93] there is an example of the use of Lemma 6.31

7 Concluding Remarks

This paper extends I/O automata [LT87, MMT91] to handle general liveness properties in both the timed and untimed model, and creates a coordinate framework where timed and untimed systems can be analyzed. A key aspect of the models is the notion of *environment-freedom*, which expresses the fact that a live (timed) I/O automaton does not constrain its environment. Moreover, the simulation method of [AL91a, LV91, LV93a, LV93b, Jon91] is extended to our model, making the results of this paper immediately applicable in practice. A substantial verification project using the model appears in [SLL93, LLS93]. In addition to generalizing the I/O automaton model [LT87] and its timed version [MMT91], our model generalizes the failure free complete trace structures of [Dil88] and the strong I/O feasibility notion of [VL92].

People familiar with process algebras might object to our model, arguing that environment-freedom is too restrictive since it rules out several systems that might be of interest at a high level of abstraction. We recognize this objection and regard the generalization of the model as future work. In fact, our model is closer to the classical models of the process algebraic community than the models of [AL93, AL91b], and thus may represent a natural starting point for possible generalizations. Some promising results come from [Seg93], which shows that there is a strong connection between the trace semantics of I/O automata and the MUST preorder of the theory of testing [DH84].

Another line of research consists of extending the current model to handle systems with probabilistic behaviors. The ultimate goal would be a model where probabilistic behaviors, timing constraints, safety properties, and liveness properties can be integrated together.

Acknowledgements

We thank Hans Henrik Løvengreen and Frits Vaandrager for their valuable criticism and useful comments on this paper.

References

[AL91a] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 2(82):253–284, 1991.

[AL91b] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In de Bakker et al. [dBHRR91], pages 1–27.

[AL93] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993. A preliminary version appeared in *Proceedings REX Workshop on Stepwise Refinement of Distributed Systems*, pages 1–41, Mook, The Netherlands, May 1989, LNCS 430, 1990, Springer-Verlag.

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[Bel76] D. Belsnes. Single message communication. *IEEE Transactions on Communications*, Com-24(2), February 1976.

[BG91] J.C.M. Baeten and J.F. Groote, editors. *Proceedings CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[Cle92] W.R. Cleaveland, editor. *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[dBHRR91] J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors. *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Dil88] D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1988.

[Fis85] M. Fischer. Re: Where are you? E-mail message to Leslie Lamport. Arpanet message number 8506252257.AA07636@YALE-BULLDOG.YALE/ARPA (47 lines), June 25 1985.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs, 1985.

[Jon91] B. Jonsson. Simulations between specifications of distributed systems. In Baeten and Groote [BG91], pages 346–360.

[Kön26] D. König. Sur les correspondances multivoques des ensembles. *Fundamenta Mathematicae*, 8:114–134, 1926.

[Lam91] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Equipment Corporation, Systems Research Center, December 1991.

[LLS93] Butler W. Lampson, Nancy A. Lynch, and Jørgen .F Søgaard-Andersen. Correctness of at-most-once message delivery protocols. In *Formal Description Techniques VI*. North-Holland Elsevier, 1993.

[LS89] N.A. Lynch and E.W. Stark. A proof of the kahn principle for Input/Output automata. *Information and Computation*, 82(1):81–92, 1989.

[LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6^{th} Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, Canada, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[LV91] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations for timing-based systems. In de Bakker et al. [dBHRR91], pages 397–446.

[LV93a] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part I: Untimed systems, 1993. Submitted for publication.

[LV93b] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems, 1993. Submitted for publication.

[Lyn93] N.A. Lynch. Simulation techniques for proving properties of real-time systems. In *Proceedings of the REX Workshop "A Decade of Concurrency"*, Lecture Notes in Computer Science. Springer-Verlag, 1993. To appear.

[MMP91] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In de Bakker et al. [dBHRR91], pages 447–484.

[MMT91] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In Baeten and Groote [BG91], pages 408–423.

[NS92] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K.G. Larsen and A. Skou, editors, *Proceedings of the Third Workshop on Computer Aided Verification,* Aalborg, Denmark, July 1991, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer-Verlag, 1992.

[OG76] S. Owicki and D. Gries. An axiomatic technique for parallel programs i. *Acta Informatica*, 6:319–340, 1976.

[RWZ92] N. Reingold, D.W. Wang, and L.D. Zuck. Games I/O automata play. In Cleaveland [Cle92], pages 325–339.

[Seg93] R. Segala. Quiescence, fairness, testing and the notion of implementation. In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[SLL93] J.F. Søgaard-Andersen, N.A. Lynch, and B.W. Lampson. Correctness of communication protocols. a case study. Technical Report MIT/LCS/TR-589, Laboratory for Computer Science, Massachusetts Institute of Technology, November 1993.

[VL92] F.W. Vaandrager and N.A. Lynch. Action transducers and timed automata. In Cleaveland [Cle92], pages 436–455.