Liveness in Timed and Untimed Systems *

Rainer Gawlick[1] Roberto Segala[1] Jørgen Søgaard-Andersen[1,2]
Nancy Lynch[1]

[1] Laboratory for Computer Science, MIT
[2] Department of Computer Science, Technical University of Denmark

Abstract. We present a coordinated pair of general labeled transition system models for describing timed and untimed concurrent systems. Both of the models incorporate liveness properties as well as safety properties. The models are related via an *embedding* of the untimed model into the timed model, which preserves all the interesting attributes of the untimed model. Both models include notions of *environment-freedom*, which express the idea that the liveness properties can be guaranteed by the system, independently of the behavior of the environment in which it operates. These environment-freedom conditions are used to prove compositionality results for both models. This pair of models, which generalize several existing models, is intended to comprise a general formalism for the verification of timed and untimed concurrent systems.

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 the verification of both *safety* and *liveness* properties [3]. Roughly speaking, a liveness property specifies that certain desirable events will eventually occur, while a safety property specifies that certain undesirable events will never occur.

In this paper, we present a *coordinated formalism* that permits modeling and verification of safety and liveness properties for both timed and untimed systems. The formalism 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 formalism is intended to support a variety of verification techniques, including *simulation methods*, *compositional reasoning*, *algebraic methods*, and *temporal logic methods*.

The Input/Output (I/O) automaton model of Lynch and Tuttle [10] and it timed extension by Merritt, Modugno and Tuttle [13], have been used successfully in the past as a formalism for verification. 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 *input* and *output* actions, which characterize the events under the control of the environment and those under the control

* Full report appears as [8]. Supported by NSF grant CCR-92-25124, DARPA contract N00014-92-J-4033, ONR contract N00014-91-J-1046, and by the Danish Technical Research Council.

of the automaton, respectively. I/O automata include a special type of liveness property called *fairness*, also known as *weak fairness*. An I/O automaton behaves fairly if it gives infinitely many turns to each of its subcomponents; a fair trace is a sequence of actions that occur during a fair execution. The distinction between input and output is used to justify the use of the simple notion of *fair trace inclusion* as a notion of implementation, which in turn is important to the simulation based proof methods [10–12]. The generalization to the timed case adds upper and lower time bounds for some of the subcomponents and augments the traces to include time information.

Unfortunately, I/O automata do not quite meet our needs. The problem is that there are some liveness properties that cannot be expressed naturally using just the simple notion of I/O automaton fairness; see [15] for an example. This motivates the attempt to generalize the I/O automaton model to handle more general liveness properties, while retaining an implementation notion based on some form of trace inclusion.

A simple and natural generalization is suggested by the work of Abadi and Lamport [2], which models a machine as a pair (A, L) consisting of an automaton A and a subset L of its executions 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. Unfortunately, if L is not restricted, simple examples show that live trace inclusion is not compositional (c.f. Example 3).

In this paper, we identify the appropriate restrictions on L, in the untimed 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 may not constrain its environment. The environment-freedom property is defined, using ideas from Dill [7], by means of a two-person game between a live (timed) I/O automaton and its environment. Specifically, the environment provides any input, while the system tries to react so that it behaves according to its liveness property L. A live (timed) I/O automaton has a *winning strategy* against its environment if it has a way to behave according to L independently of its environment. If a live (timed) I/O automaton has a winning strategy it is said to be environment-free.

The environment-freedom property for the timed model is a natural extension of the one for the untimed model up to some technical details involving the so-called *Zeno executions*. The close relation between the two definitions allows the timed and untimed models to be tied together, thus (for example) permitting the verification for timed implementations of untimed specifications. Specifically, we define an embedding, similar to the *patient* transducer of [17], that converts live I/O automata into live timed I/O automata without timing constraints. The embedding, which is omitted from this abstract (see [8]), preserves the environment-freedom property and the trace preorder relations of the untimed model. Furthermore, it commutes with the parallel composition operator.

Our model is closely related to several others in the literature. It captures the I/O automata of [10], the failure free complete trace structures of [7], and the timed I/O automata of [13]. It generalizes the notion of *strong I/O feasibility* introduced in [17]. The untimed model is similar to the model of [2]. However, the generalization of [2] to the timed case [1] is very complex, possibly because of the absence of a clear role for *time* in the interaction between the automaton and its environment. In contrast, our generalization to the timed case is simple, and follows naturally from the untimed case.

It is already clear that our formalism supports a wide range of proof methods, including *simulation methods* as described in [11, 12] (and extended to handle liveness in [8, 16]), *compositional reasoning* as justified by the theorems of this paper, and *tempo-*

ral logic methods, as described in [16]. An extensive verification project that uses the formalism described in this paper can be found in [9, 15, 16]; in fact, that verification project provided a major impetus to the development of our formalism.

The paper is divided in two main sections, dealing with the untimed and timed model, respectively. The two sections have a similar structure: first they present the basic safe models, taken from [10] and [12], respectively, then they present the live model along with the main theorems and some examples showing that our environment-freedom condition cannot easily be relaxed. Finally, there is a comparison with existing work.

2 The Untimed Setting

2.1 Safe Automata

A *safe automaton* is simply a state machine with labeled transitions. A safe 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 $(ext(A), int(A))$, where $ext(A)$ and $int(A)$ are disjoint sets of external and internal actions, and a transition relation $steps(A) \subseteq states(A) \times acts(A) \times states(A)$, where $acts(A)$ denotes the set $ext(A) \cup int(A)$ of actions of A. An action a of safe automaton A is said to be *enabled* in state s iff there exists some state s' such that the step (s, a, s') is an element of $steps(A)$.

An *execution fragment* $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$ of a safe automaton A is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the sequence finite, ending in a state, where each $(s_i, a_{i+1}, s_{i+1}) \in steps(A)$. An *execution* is an execution fragment whose first state is a start state. Denote by $exec^*(A)$ and $exec(A)$ the sets of finite and all execution of A, respectively.

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*. 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 $\alpha_2 = \alpha_1 \frown \alpha_1'$.

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., $trace(\alpha) = \alpha \restriction ext(A)$. 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 is an execution α of A with $trace(\alpha) = \beta$. Denote by $traces(A)$ the set of traces of A.

Safe automata $A_1, \ldots, A_N$ are *compatible* if for all $1 \leq i, j \leq N$, with $i \neq j$, $int(A_i) \cap acts(A_j) = \emptyset$. The *parallel composition* $A_1 \parallel \cdots \parallel A_N$ of compatible safe automata $A_1, \ldots, A_N$ is the safe automaton A whose states and start states are the cross product of the states and start states, respectively, of its components, whose internal and external actions are the union of the internal and external actions, respectively, of its components, and whose transition relation is such that $((s_1, \ldots, s_N), a, (s_1', \ldots, s_N')) \in steps(A)$ iff for all $1 \leq i \leq N$, if $a \in acts(A_i)$ then $(s_i, a, s_i') \in steps(A_i)$, and if $a \notin acts(A_i)$ then $s_i = s_i'$.

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 $s_i \in states(A)$ and $a_i \in acts(A)$ for all i, and let $\alpha \lceil A_i$ be the sequence obtained from α by projecting each state into its i^{th} component and by removing each subsequence $a_j s_j$ whenever $a_j \notin acts(A_i)$. Then it is easy to show that $\alpha \lceil A_i \in exec(A_i)$ for all A_i iff $\alpha \in exec(A)$. This property will be used to compose live automata in parallel.

2.2 Live Automata

A safe automaton A can be thought of as expressing *safety* properties. *Liveness* properties can be expressed by a subset L of its executions, as suggested in [3]. In order to ensure that the set L of executions does not introduce additional safety restrictions, it should not be possible to violate L in a finite number of steps. Thus, any finite execution of A must be extendible to an execution in L. This requirement is closely related to the *liveness relative to a safety property* of [6] and to the *machine-closure* of [2].

Definition 1. A *liveness condition* L for an automaton A is a subset of the executions of A such that any finite execution of A is a prefix of an execution 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. ■

Informally, a liveness condition can be used to express (at least) two intuitively different requirements. First, a liveness condition can specify assumptions about the execution of a system that are based on its physical structure, e.g., that individual physical processors continue to operate. Second, a liveness condition can specify additional properties that a system is required to satisfy, e.g., that every message sent is eventually delivered. We simply think of a liveness condition as representing the set of executions that a system can exhibit whenever it is "working properly".

The most natural extensions of the notion of implementation used for I/O automata, *fair trace inclusion*, are the following preorders, where the safe preorder coincides with the unfair preorder of [10], and the live preorder generalizes the fair preorder of [10].

Definition 2. Given two live automata (A_1, L_1) and (A_2, L_2) with the same external action signature, 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 $traces(L_1) \subseteq traces(L_2)$ ■

2.3 Safe I/O Automata

A *safe I/O automaton* A is an automaton augmented with an *external action signature*, $(in(A), out(A))$, which partitions $ext(A)$ into input and output actions. A must be *input enabled*, i.e., each input action is enabled from each state. The internal and output actions of A are referred to as the *locally-controlled* actions of A, written $local(A)$. The compatibility requirement for safe I/O automata is strengthened by forbidding common output actions; the output actions of the parallel composition of safe I/O automata are given by the union of the output actions of each component.

2.4 Live I/O Automata

For I/O automata, input enabling achieves the independence from the environment required for the compositionality result. However, for general liveness, the following example shows that input enabling is not enough for such independence.

Example 1. Let A be a safe I/O automaton with a unique state s, a unique input action i, a unique output action o, and a self-loop step on s for i and o. Let L be the set of executions of A containing at least five occurrences of action i. L is trivially a liveness condition for A. However, the live automaton (A, L) would not behave properly if the environment does not provide more than four i actions. ■

A similar problem is noted in [2,7], leading to the notion of *receptiveness*. Intuitively, a system is receptive if it behaves properly independently of the inputs provided by 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, and the system moves consist of performing at most one local 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 contain at most finitely many 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 setting, a strategy consists of a pair of functions (g, f). Function g specifies which state the system reaches in response to any given input action; function f determines the next move of the system, which can be a local step or no step ($\perp$).

Definition 3. Let A be a safe I/O automaton. 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 $\alpha a s \in exec^*(A)$ ■

The strategies of [2,7] consist of function f only, since in [7] there is no notion of state, and in [2] the environment moves by changing the state of the system.

The moves of the environment can be represented as an infinite sequence $\mathcal{I}$, called an *environment sequence*, of input actions interleaved with infinitely many λ symbols, where λ represents the points at which the system is allowed to move. The occurrence of infinitely many λ symbols in $\mathcal{I}$ guarantees that each environment move consists of only finitely many input actions. Let the game start 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 4. Let A be a safe I/O automaton, α a finite execution of A, (g, f) a strategy defined on A, and let $\mathcal{I}$ be an environment sequence for A. Consider the chain of executions $(\alpha^n)_{n \geq 0}$, ordered by prefix, constructed inductively as follows: $(\alpha^0, \mathcal{I}^0) = (\alpha, \mathcal{I})$ and for each $n \geq 0$

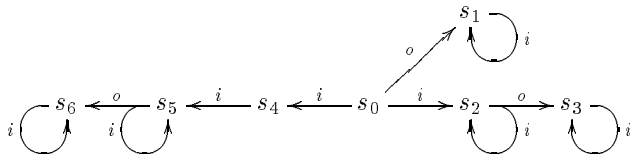
$$(\alpha^{n+1}, \mathcal{I}^{n+1}) = \begin{cases} (\alpha^n a s, \mathcal{I}') \text{ if } \mathcal{I}^n = \lambda \mathcal{I}',\ f(\alpha^n) = (a, s) \\ (\alpha^n, \mathcal{I}') \quad \text{if } \mathcal{I}^n = \lambda \mathcal{I}',\ f(\alpha^n) = \perp \\ (\alpha^n b s, \mathcal{I}') \text{ if } \mathcal{I}^n = b \mathcal{I}',\ g(\alpha^n, b) = s \end{cases}$$

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

Definition 5. 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 $\mathcal{O}_{(g,f)}(\alpha, \mathcal{I}) \in L$. The strategy (g, f) is called an *environment-free strategy* for (A, L). ■

Definition 6. 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 2. Consider the safe I/O automaton A described by the transition diagram



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 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 [2] 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 [2] 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 $\alpha \in exec^*(A)$ which ensures that L does not introduce any new safety properties. ■

Definition 7. N live I/O automata are compatible iff their safe components 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 $L = \{\alpha \in exec(A) \mid \alpha\lceil A_1 \in L_1, \ldots, \alpha\lceil A_N \in L_N\}$. ■

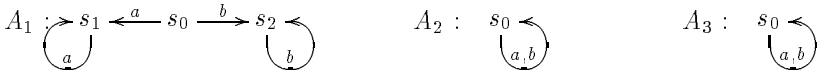
Our key theorems about live I/O automata are the following compositionality results.

Theorem 8. *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.* ■

Theorem 9. *Define $(A_1, L_1), \ldots, (A_N, L_N)$ and $(A'_1, L'_1), \ldots, (A'_N, L'_N)$, to be N-tuples of compatible live I/O automata, and let $\sqsubseteq_X$ be either $\sqsubseteq_S$ or $\sqsubseteq_L$. If, for each i, $(A_i, L_i) \sqsubseteq_X (A'_i, L'_i)$, then $(A_1, L_1) \| \cdots \| (A_N, L_N) \sqsubseteq_X (A'_1, L'_1) \| \cdots \| (A'_N, L'_N)$.* ■

Environment-freedom and I/O distinction are crucial properties of live I/O automata. If a pair (A, L) is not environment-free, the parallel composition operator may generate pairs that are not even live automata, and the compositionality of the live preorder would fail. Clearly, the same problems arise if the I/O distinction is removed.

Example 3. Let A_1, A_2 and A_3 be the safe I/O automata



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 occurrence of a or one occurrence of b 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.

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. However, $(A_1, L_1) \| (A_3, L_3)$ is not a live I/O automaton since A_1 can never perform an action a immediately followed by an action b. This provides an example where in absence of environment-freedom an implementation (A_1, L_1) cannot be safely substituted for its specification (A_2, L_2). ■

3 The Timed Setting

3.1 Safe Timed Automata

A *safe timed automaton* A [12] is a safe automaton whose set of external actions contains a special *time-passage* action ν. Define the set $vis(A)$ of *visible* actions to be $ext(A) \setminus \{\nu\}$. As an additional component, a safe timed automaton contains a mapping $.now_A : states(A) \to \mathbb{R}^{\geq 0}$ (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 state the last axiom, the following auxiliary definition is needed. Let I be an interval of $\mathbb{R}^{\geq 0}$. Then an *A-trajectory* is a function $\omega : I \to states(A)$, such that

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 span from state $fstate(\omega)$ to state $lstate(\omega)$. A trajectory ω whose domain $dom(\omega)$ is a singleton set $[t, t]$ is also denoted by the set $\{\omega(t)\}$.

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

Axioms **S1-S4** are self-explanatory; axiom **S5** says that if time can pass from t to t', then it is possible to associate states with all times in interval $[t, t']$ in a consistent way.

A *timed execution fragment* $\Sigma = \omega_0 a_1 \omega_1 a_2 \omega_2 \cdots$ of a timed automaton A is a (finite or infinite) sequence of alternating trajectories and actions in $vis(A) \cup int(A)$, starting in a trajectory and, if the sequence is finite, ending in a trajectory, 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 $(lstate(\omega_i), a_{i+1}, fstate(\omega_{i+1})) \in steps(A)$.

A *timed execution* is a timed execution fragment whose first state is a start state.

If $\Sigma = \omega_0 a_1 \omega_1 \cdots$ is a timed execution fragment, then define $ftime(\Sigma)$ and $fstate(\Sigma)$ to be $ftime(\omega_0)$ and $fstate(\omega_0)$, respectively. Also, define $ltime(\Sigma)$ to be the supremum of the times of the states in Σ. If Σ is a finite sequence and the domain of the last trajectory ω is closed, then define $lstate(\Sigma)$ to be $lstate(\omega)$.

A timed execution (fragment) Σ is *finite*, if it is a finite sequence and the domain of the last trajectory is closed. 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. Note that Zeno timed executions can be of two types: those containing infinitely many occurrences of non-time-passage actions in a finite amount of time, and those containing finitely many occurrences of non-time-passage actions and for which the domain of the last trajectory is right-open and bounded. Denote by $t\text{-}exec^*(A)$, $t\text{-}exec^\infty(A)$, and $t\text{-}exec(A)$ the sets of finite, admissible, 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 $\omega \frown \omega'(t)$ is defined to be $\omega(t)$ if t is in $dom(\omega)$, and $\omega'(t)$ if t is in $dom(\omega') \backslash dom(\omega)$. 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 $\Sigma_2 = \Sigma'_1 \frown \Sigma_1$. Define $\Sigma \lhd t$, read "Σ before t", for all $t \geq ftime(\Sigma)$, to be the t-prefix of Σ that includes exactly all states with times not bigger than t. Likewise, define $\Sigma \rhd t$, read "Σ after t", for all $t < ltime(\Sigma)$ or all $t \leq ltime(\Sigma)$ when Σ is finite, to be the t-suffix of Σ that includes exactly all states with times not smaller than t.

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 $ltime(\omega_{i-1})$, or equivalently, $ftime(\omega_i)$. Then, define $t\text{-}seq(\Sigma) = (a_1, t_1)(a_2, t_2) \cdots$ to be the sequence consisting of the actions in Σ paired with their time of occurrence. Then $t\text{-}trace(\Sigma)$, the *timed trace* of Σ, is defined to be the pair $(t\text{-}seq(\Sigma) \restriction (vis(A) \times \mathbb{R}^{\geq 0}), ltime(\Sigma))$. Thus, $t\text{-}trace(\Sigma)$ records the occurrences of *visible* actions together with their time of occurrence, and the limit time of the timed execution fragment. Denote by $t\text{-}traces(A)$ the set of timed traces of A.

The parallel composition operator for safe timed automata is defined similarly to the corresponding operator for the untimed model. In the composition, time is allowed to pass by a certain amount only if all component automata allow the same amount of time to pass. Also, at each state of the composition all 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. The timed executions of the parallel composition $A = A_1 \| \cdots \| A_N$ can be characterized by means of projections as in the untimed case. 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. 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. Then, $\Sigma \lceil A_i \in t\text{-}exec(A_i)$, for all A_i, iff $\Sigma \in t\text{-}exec(A)$.

As for the untimed model, two preorder relations are defined. The definition of a live timed automaton is given in the same way as for live automata.

Definition 10. Given two live timed automata (A_1, L_1) and (A_2, L_2) with the same external action signature, define:

Safe: $(A_1, L_1) \sqsubseteq_{\mathrm{St}} (A_2, L_2)$ iff $t\text{-}traces(A_1) \subseteq t\text{-}traces(A_2)$.
Live: $(A_1, L_1) \sqsubseteq_{\mathrm{Lt}} (A_2, L_2)$ iff $t\text{-}traces(L_1) \subseteq t\text{-}traces(L_2)$. ■

3.2 Safe Timed I/O Automata

A *safe timed I/O automaton* is defined similarly to the untimed case. This time the pair $(in(A), out(A))$ is a partition of $vis(A)$, and is called the *visible action signature* of A.

3.3 Live Timed I/O Automata

The definition of live timed I/O automata, is considerably more complicated than the definition of live I/O automata, because the presence of time in the model has a strong impact on the type of interactions that can occur between a timed automaton and its environment. In the untimed model, the relative speed of the system with respect to its environment is determined by the environment moves; in the timed model the relative speed is determined by the explicit time associated with each action. 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. In this way the current state of the system will always be known should the time-passage step be interrupted by an input action.

A strategy in the timed model is again a pair of function (g, f). Function f takes a finite timed execution and specifies how the system behaves until its next locally-controlled action, assuming that no input is received in the meantime. Function g specifies what state is reached whenever some input is received.

Definition 11. Let A be a any safe timed I/O automaton. A *strategy* defined on A is a pair of functions (g, f) where $g : t\text{-}exec^*(A) \times in(A) \to states(A)$ and $f : t\text{-}exec^*(A) \to (traj(A) \times local(A) \times states(A)) \cup traj(A)$, where $traj(A)$ denotes the set of trajectories of A, such that

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

Let $f(\Sigma).trj$ denote the trajectory part of $f(\Sigma)$. ■

The consistency condition of Definition 11 is needed for technical reasons; it has the intuitive meaning that a strategy's decision cannot change in the absence of inputs.

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. 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 uses function g to compute the state reached as a result of the input. A problem arises when the system decides to perform an action at the same real time as the environment is providing some input. Such a situation is modeled as a nondeterministic choice. As a consequence, the *outcome* for a timed strategy is a *set* of timed executions rather than just a single execution.

Definition 12. Let A be a safe timed I/O automaton, Σ be a finite timed execution of A, and (g, f) be a strategy defined on A. Let $\mathcal{I} = (a_1, t_1), (a_2, t_2), \cdots$ be a sequence of input actions of A paired with non-decreasing times such that either $\mathcal{I}$ is empty or $ltime(\Sigma) \leq t_1$. $\mathcal{I}$ is called a *timed environment sequence* for A compatible with Σ.

Consider the set $\mathcal{S}$ of chains (ordered by *t-prefix*) of timed executions $(\Sigma^n)_{n\geq 0}$ such that $(\Sigma^0, \mathcal{I}^0) = (\Sigma, \mathcal{I})$, and for each $n \geq 0$ one of the following conditions is satisfied:

$$(\Sigma^{n+1}, \mathcal{I}^{n+1}) = \begin{cases} (\Sigma^n \frown \omega a\{s\}, \mathcal{I}^n) & \text{if } \mathcal{I}^n = \varepsilon,\ f(\Sigma^n) = (\omega, a, s) \\ (\Sigma^n \frown \omega, \mathcal{I}^n) & \text{if } \mathcal{I}^n = \varepsilon,\ f(\Sigma^n) = \omega \\ (\Sigma^n \frown \omega a\{s\}, \mathcal{I}^n) & \text{if } \mathcal{I}^n = (b,t)\mathcal{I}',\ f(\Sigma^n) = (\omega, a, s),\ ltime(\omega) \leq t \\ (\Sigma^n \frown \omega' b\{s'\}, \mathcal{I}') & \text{if } \mathcal{I}^n = (b,t)\mathcal{I}',\ f(\Sigma^n).trj = \omega, \\ & \quad ltime(\omega) \geq t,\ \omega' = \omega \lhd t,\ g(\Sigma^n \frown \omega', b) = s'. \\ (\Sigma^n, \mathcal{I}^n) & \text{if } \Sigma^n \text{ is not finite} \end{cases}$$

Note, that Σ^n is finite in the first four cases. The *outcome* $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I})$ of the strategy (g, f) applied to Σ and $\mathcal{I}$ is the set of timed executions Σ' for which there exists $(\Sigma^n)_{n\geq 0} \in \mathcal{S}$ such that $\Sigma' = \lim_{n\to\infty} \Sigma^n$. ■

The first, second, and third cases of the above inductive definition deal with different situations in which no input occurs during the system move chosen by f. The fourth case takes care of the situation where inputs do occur during the system move chosen by f. Note that the third and fourth cases are both applicable whenever the next input action of $\mathcal{I}$ and the local action chosen by f occur at the same time. Finally, the fifth case of the inductive definition is needed for technical convenience, since the second case generates an admissible timed execution.

A 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 Abadi, Zeno timed executions cannot be ignored completely.

Example 4. 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 execution, 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 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 13. Let Σ be a timed execution of a safe timed automaton A.

- Σ is *environment-Zeno* if Σ is a Zeno timed execution that contains infinitely many input actions;
- Σ is *system-Zeno* if Σ is a Zeno timed execution that either contains infinitely many locally-controlled actions or contains finitely many actions;
- Σ is *Zeno-tolerant* if it is an environment-Zeno, non-system-Zeno timed execution. Denote by $t\text{-}exec^{Zt}(A)$ the set of Zeno-tolerant timed executions of A. ■

Definition 14. A strategy (g, f) defined on a safe timed I/O automaton A is *Zeno-tolerant* if, for each finite $\Sigma \in t\text{-}exec^*(A)$ and each timed environment sequence $\mathcal{I}$ for A compatible with Σ, $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq t\text{-}exec^\infty(A) \cup t\text{-}exec^{Zt}(A)$. ■

Definition 15. 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, $\mathcal{O}_{(g,f)}(\Sigma, \mathcal{I}) \subseteq L$. ■

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 L. Note that the environment-freedom of (A, L) does not depend on the finite or system-Zeno timed executions of L. Also, A may never generate any finite or system-Zeno timed execution, since this would constrain its environment. Thus, it is reasonable to exclude system-Zeno timed executions from liveness conditions. Similarly, we could exclude Zeno-tolerant timed executions, except that they are needed to handle illegal interactions. 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 16. A *live timed I/O automaton* is a pair (A, L), where A is a safe timed I/O automaton, $L \subseteq t\text{-}exec^\infty(A)$, and $(A, L \cup t\text{-}exec^{Zt}(A))$ is environment-free. ■

The parallel composition for live timed I/O automata is defined in the same way as for the untimed case. Having built up all the requisite machinery, we obtain the compositionality and substitutivity theorems for the timed case, just as for the untimed case. The proofs are long, but no more difficult, conceptually, than for the untimed case.

Theorem 17. *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.* ■

Theorem 18. *Define $(A_1, L_1), \ldots, (A_N, L_N)$ and $(A'_1, L'_1), \ldots, (A'_N, L'_N)$, to be N-tuples of compatible live timed I/O automata, and let $\sqsubseteq_X$ be either $\sqsubseteq_{\mathrm{St}}$ or $\sqsubseteq_{\mathrm{Lt}}$. If, for each i, $(A_i, L_i) \sqsubseteq_X (A'_i, L'_i)$, then $(A_1, L_1) \| \cdots \| (A_N, L_N) \sqsubseteq_X (A'_1, L'_1) \| \cdots \| (A'_N, L'_N)$.* ■

4 Related Work

An I/O automaton M of [10] can be represented in our model as the environment-free pair (A, L) where A is 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) simply gives turns (say in a round robin way) to all the components of M that are continuously willing to perform some locally-controlled action. In a similar way a timed I/O automaton of [13] can be represented in our timed model.

The failure free complete trace structures of [7] are a special case of our model, where the state structure of a machine is not considered. However, they are not adequate to describe systems whenever their state structure is important.

The model of [2] is closely related to our model (c.f. Example 2). However, our timed model departs from the key ideas of [1], leading to a more natural treatment of time.

The work in [17] does not deal with general liveness, and uses finite and admissible timed traces inclusion as an implementation relation. The automata of [17] need not be environment-free, however, to avoid trivial implementations and guarantee closure under composition, [17] assumes some form of I/O distinction and some more restrictive form of environment-freedom, called *strong I/O feasibility*, at the lower level of implementation. Our notion of environment-freedom solves the same problem in a more general way.

It is easy to show, given our definition of environment-freedom, that the set of live traces of any live automaton is union-game realizable according to [14], and thus describable by means of a standard I/O automaton of [10]. However in general the I/O automaton description would be extremely unnatural.

Acknowledgments: We thank Hans Henrik Løvengreen and Frits Vaandrager for their valuable criticism and useful comments.

References

1. M. Abadi and L. Lamport. An old-fashioned recipe for real time. In [5], pages 1–27.
2. M. Abadi and L. Lamport. Composing specifications. *TOPLAS*, 15(1):73–132, 1993.
3. B. Alpern and F. Schneider. Defining liveness. *IPL*, 21(4):181–185, 1985.
4. *Proceedings of CONCUR 92,* Stony Brook, NY, USA, LNCS 630, 1992.
5. *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, LNCS 600, 1991.
6. F. Dederichs and R. Weber. Safety and liveness from a methodological point of view. *Information Processing Letters*, 36(1):25–30, 1990.
7. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1988.
8. R. Gawlick, R. Segala, J. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. Technical Report MIT/LCS/TR-587, November 1993.
9. Butler Lampson. Principles for computer system design, 1993. Turing Award Talk.
10. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. PODC*, 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.
11. N. Lynch and F. Vaandrager. Forward and backward simulations – part I: Untimed systems. Technical Report MIT/LCS/TM-486, May 1993. Preliminary version in [5].
12. N. Lynch and F. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Technical Report MIT/LCS/TM-487, September 1993. Preliminary version in [5].
13. M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *Proceedings CONCUR 91,* Amsterdam, LNCS 527, 1991.
14. N. Reingold, D. Wang, and L. Zuck. Games I/O automata play. In [4], pages 325–339.
15. J. Søgaard-Andersen, B. Lampson, and N. Lynch. Correctness of at-most-once message delivery protocols. In *Proc. FORTE 93*, 1993.
16. J. Søgaard-Andersen, N. Lynch, and B. Lampson. Correctness of communication protocols, a case study. Technical Report MIT/LCS/TR-589, November 1993.
17. F. Vaandrager and N. Lynch. Action transducers and timed automata. In [4].