Time for Concurrency *

Zohar Manna †   Amir Pnueli ‡

October 6, 1992

Abstract. A hierarchy of models that capture realistic aspects of reactive and real-time systems is introduced. On the most abstract level, the *qualitative* (non-quantitative) model captures the temporal precedence aspect of time. A more refined model is that of *timed transition systems*, representing the metric aspects of time. The third and most detailed model is that of *hybrid systems*. This model allows the incorporation of *continuous* components into the studied reactive system.

Keywords: Temporal logic, real-time, specification, verification, hybrid systems, Statecharts, proof rules.

1 Introduction

As our ability to specify and develop programs for reactive systems increases, there is a growing interest in the representation of more realistic features of such systems in the formal models and languages used for their specification, verification, and development.

As is the case with the application of mathematics to other scientific and engineering disciplines, no single model can fully capture the physical phenomenon under study. This paper presents a hierarchy of three models for the specification and verification of reactive systems:

- A *qualitative* (non-quantitative) model that captures the *temporal precedence* aspect of time. This model can only identify that one event precedes the other but not by how much.
- A *real-time* model that captures the *metric* aspect of time. This model can measure the time elapsing between two events.
- A *hybrid system* model that allows the inclusion of *continuous* components in a reactive system. Such continuous components may cause continuous change in the values of some state variables according to some physical or control law.

*This research was supported in part by the National Science Foundation under grants CCR-89-11512 and CCR-89-13641, by the Defense Advanced Research Projects Agency under contract NAG2-703, by the United States Air Force Office of Scientific Research under contract AFOSR-90-0057, by the European Community ESPRIT Basic Research Action Project 6021 (REACT) and by the France-Israel project for cooperation in Computer Science.

†Department of Computer Science, Stanford University, Stanford, CA 94305

‡Department of Applied Mathematics and Computer Science, Weizmann Institute, Rehovot, Israel

This paper omits many details of the models and their verification. A fuller account of these missing elements is provided in [MP92].

2 The Qualitative Model

The qualitative model uses an abstract notion of time, based on the ordering of events during an observed computation. This is the main model used, for example, in [MP91].

2.1 Computational Model: Fair Transition System

The computational model for the qualitative level is that of *fair transition systems*. Such a system consists of the following components.

- $V = \{u_1, ..., u_n\}$: A finite set of *state variables.*

  We define a *state* s to be an interpretation of V, assigning to each variable $u \in V$ a value $s[u]$ over its domain. We denote by Σ the set of all states.

- Θ : The *initial condition.* This is an assertion characterizing all the *initial states*, i.e., states at which a computation of the program can start. It is required that Θ be satisfiable, i.e., there exists at least one state satisfying Θ.

- $\mathcal{T}$: A finite set of *transitions.* Each transition $\tau \in \mathcal{T}$ is a function $\tau : \Sigma \mapsto 2^{\Sigma}$, mapping each state $s \in \Sigma$ into a (possibly empty) set of τ-successor states $\tau(s) \subseteq \Sigma$.

  A transition τ is *enabled* on s iff $\tau(s) \neq \phi$. Otherwise τ is *disabled* on s.

  The function associated with a transition τ is represented by an assertion $\rho_\tau(V, V')$, called the *transition relation*, which relates a state $s \in \Sigma$ to its τ-successor $s' \in \tau(s)$ by referring to both unprimed and primed versions of the state variables. An unprimed version of a state variable refers to its value in s, while a primed version of the same variable refers to its value in s'. For example, the assertion $x' = x + 1$ states that the value of x in s' is greater by 1 than its value in s.

- $\mathcal{J} \subseteq \mathcal{T}$: A set of *just* transitions (also called *weakly fair* transitions). Intuitively, the requirement of justice for $\tau \in \mathcal{J}$ disallows a computation in which τ is continually enabled beyond a certain point but taken only finitely many times.

- $\mathcal{C} \subseteq \mathcal{T}$: A set of *compassionate* transitions (also called *strongly fair* transitions). Intuitively, the requirement of compassion for $\tau \in \mathcal{C}$ disallows a computation in which τ is enabled infinitely many times but taken only finitely many times.

The transition relation $\rho_\tau(V, V')$ identifies state s' as a τ-*successor* of state s if

$$\langle s, s' \rangle \models \rho_\tau(V, V'),$$

where $\langle s, s' \rangle$ is the joint interpretation which interprets $x \in V$ as $s[x]$, and interprets x' as $s'[x]$.

The enabledness of a transition τ can be expressed by the formula

$$En(\tau) : \; (\exists V')\rho_\tau(V, V'),$$

which is true in s iff s has some τ-successor. We require that every state $s \in \Sigma$ has at least one transition enabled on it.

Let $\mathcal{S}$ be a transition system for which the above components have been identified. We define a *computation* of $\mathcal{S}$ to be an infinite sequence of states $\sigma : s_0, s_1, s_2, ...,$ satisfying the following requirements:

- *Initiation:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j = 0, 1, ...,$ the state s_{j+1} is a τ-successor of the state s_j, i.e., $s_{j+1} \in \tau(s_j)$, for some $\tau \in \mathcal{T}$. In this case, we say that the transition τ is *taken* at position j in σ.
- *Justice:* For each $\tau \in \mathcal{J}$ it is not the case that τ is continually enabled beyond some point in σ but taken at only finitely many positions in σ.
- *Compassion:* For each $\tau \in \mathcal{C}$ it is not the case that τ is enabled on infinitely many states of σ but taken at only finitely many positions in σ.

For a system $\mathcal{S}$, we denote by $\mathcal{C}omp(\mathcal{S})$ the set of all computations of $\mathcal{S}$.

2.2 A Simple Programming Language

To present programs, we introduce a simple concurrent programming language (SPL) in which processes communicate by shared variables. The reader is referred to [MP91] for a description of the full language. Here, we will illustrate its syntax and semantics through an example.

Fig. 1 presents a simple program consisting of two processes communicating by the shared variable x, initially set to 0. Process P_1 keeps incrementing variable y as long as $x = 0$. Process P_2 has only one statement, which sets x to 1. Obviously, once x is set to 1, process P_2 terminates and some time later so does P_1, as soon as it observes that $x \neq 0$.

$$x,\ y\text{: } \textbf{integer where } x = y = 0$$

$$P_1 :: \left[\begin{array}{l} \ell_0 : \ \ \textbf{while } x = 0 \textbf{ do} \\ \qquad \left[\ell_1 :\ y := y + 1\right] \\ \ell_2 : \end{array}\right] \quad \Big\| \quad P_2 :: \left[\begin{array}{ll} m_0 : & x := 1 \\ m_1 : & \end{array}\right]$$

Figure 1: Program PARA: A Simple Concurrent Program

We will show how to define a fair transition system $\mathcal{S}_P$ corresponding to program P.

• The *state variables* of system $\mathcal{S}_P$ are $V : \{\pi, x, y\}$. Variable π is a control variable ranging over subsets of the locations $\{\ell_0, \ell_1, \ell_2, m_0, m_1\}$. The value of π in a state denotes all the locations of the program in which control currently resides.

As states we take all possible interpretations that assign to the state variables values over their respective domains. For example, the initial state of program PARA is

$$\langle \pi : \{\ell_0, m_0\},\ x : 0,\ y : 0 \rangle.$$

• The *transitions* of system $\mathcal{S}_P$ are $\mathcal{T} : \{\tau_I, \ell_0, \ell_1, m_0\}$, where τ_I is the *idling transition*, and ℓ_0, ℓ_1, m_0 are transitions corresponding to the program statements.

Their transition relations are given by:

$$\begin{aligned}
\rho_I &: \ \pi' = \pi \ \wedge \ x' = x \ \wedge \ y' = y \\
\rho_{\ell_0} &: \ \ell_0 \in \pi \ \wedge \left(\begin{array}{c} x = 0 \ \wedge \ \pi' = \pi - \{\ell_0\} \cup \{\ell_1\} \\ \vee \\ x \neq 0 \ \wedge \ \pi' = \pi - \{\ell_0\} \cup \{\ell_2\} \end{array} \right) \wedge \ x' = x \ \wedge \ y' = y \\
\rho_{\ell_1} &: \ \ell_1 \in \pi \ \wedge \ \pi' = \pi - \{\ell_1\} \cup \{\ell_0\} \ \wedge \ x' = x \ \wedge \ y' = y + 1 \\
\rho_{m_0} &: \ m_0 \in \pi \ \wedge \ \pi' = \pi - \{m_0\} \cup \{m_1\} \ \wedge \ x' = 1 \ \wedge \ y' = y
\end{aligned}$$

• The *initial condition* of system $\mathcal{S}_P$ is Θ: $\quad \pi = \{\ell_0, m_0\} \ \wedge \ x = 0 \ \wedge \ y = 0.$

• The *justice set* of system $\mathcal{S}_P$ is $\mathcal{J} : \{\ell_0, \ell_1, m_0\}$.

• The *compassion set* of system $\mathcal{S}_P$ is $\mathcal{C} : \phi$, the empty set.

Requirement Specification Language: Temporal Logic

As a requirement specification language for reactive systems (under the qualitative model) we take *temporal logic* [MP91].

In this paper we use the temporal operators $\Box$ and $\Diamond$ and the entailment $p \Rightarrow q$ which is an abbreviation for the formula $\Box(p \rightarrow q)$. In the underlying assertion language we use the location predicate at_ℓ_i, which is an abbreviation for the formula $\ell_i \in \pi$, stating that control is currently at location ℓ_i. We also use the expression $at\_\ell_{i,j}$ as an abbreviation for the disjunction $at\_\ell_i \vee at\_\ell_j$. We refer to a formula in the assertion language as a *state formula*, or simply as an *assertion*.

Specification of Properties

A temporal formula φ that is valid over a program P specifies a property of P, i.e., states a condition that is satisfied by all computations of P. As is explained in [MP91], the properties expressible by temporal logic can be arranged in a hierarchy that identifies different classes of properties according to the form of formulas expressing them.

Here we will consider only properties falling into the two most important classes.

- *Safety properties* are those that can be expressed by the formula $\Box\psi$, for some past formula ψ. Here, we only consider *invariance formulas*, i.e., the case where ψ is an assertion. For example, the formula $\Box(y \geq 0)$ states that y is nonnegative in all states appearing in computations of PARA.

- *Response properties* are those that can be expressed by the formula $p \Rightarrow \Diamond q$, for past formulas p and q. Here, we only consider the case that p and q are assertions. For example, the response formula $\Theta \Rightarrow \Diamond(at\_\ell_2 \wedge at\_m_1)$ is valid over program PARA. It claims that every state satisfying the initial condition of PARA is followed by a terminal state characterized by $at\_\ell_2 \wedge at\_m_1$. This implies that all computations of PARA eventually terminate.

2.3 Verifying Safety Properties

We present a proof rule for establishing the validity of a safety formula over a given transition system $P : \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$.

Verification Conditions

For a transition τ and state formulas p and q, we define the *verification condition* of τ, relative to p and q, denoted $\{p\}\tau\{q\}$, to be the implication:

$$(\rho_\tau \wedge p) \rightarrow q',$$

where ρ_τ is the transition relation corresponding to τ, and q', the *primed version* of the assertion q, is obtained from q by replacing each variable occurring in q by its primed version. Since ρ_τ holds for two states s and s' iff s' is a τ-successor of s, and q' states that q holds on s', it is not difficult to see that if the verification condition $\{p\}\tau\{q\}$ is valid, then every τ-successor of a p-state is a q-state. For a set of transitions $T \subseteq \mathcal{T}$, we denote by $\{p\}T\{q\}$ the conjunction of verification conditions, containing the conjunct $\{p\}\tau\{q\}$ for each $\tau \in T$.

In the context of program PARA, consider for example the verification condition of ℓ_1 with respect to assertions $y \geq 0$ and $y > 0$, i.e., $\{y \geq 0\}\ \ell_1\ \{y > 0\}$. Expanding the definition of the verification condition, this yields

$$\underbrace{\ell_1 \in \pi \wedge \pi' = \pi - \{\ell_1\} \cup \{\ell_0\} \wedge x' = x \wedge y' = y + 1}_{\rho_{\ell_1}} \ \wedge\ \underbrace{y \geq 0}_{p} \ \rightarrow\ \underbrace{y' > 0}_{q'}$$

which is valid. This shows that every ℓ_1-successor of a state satisfying $y \geq 0$ satisfies $y > 0$.

An Invariance Rule

The following rule can be used to establish the validity of the invariance formula $\Box p$.

INV	I1.	$\Theta \rightarrow \varphi$
	I2.	$\{\varphi\}\mathcal{T}\{\varphi\}$
	I3.	$\varphi \rightarrow p$
		$\Box p$

Premises I1 and I2 establish that the auxiliary assertion φ is initially true and is preserved by every transition of P. It follows that all states in the computations of P satisfy φ. By premise I3, they also satisfy p.

It often happens that assertion φ appearing in rule INV naturally splits into a disjunction $\varphi = \bigvee_{i \in M} \varphi_i$, where M is some finite index range such as $1, \ldots, m$. In this case, it may be easier to prove premises I1 and I2 of the rule in the form:

$$\begin{array}{lll} \text{I1.} & \Theta \ \rightarrow \ \varphi_i & \text{for some } i \in M \\ \text{I2.} & \{\varphi_i\}\ \mathcal{T}\ \{\varphi\} & \text{for every } i \in M \end{array}$$

Let us illustrate the application of rule INV for proving the property $\Box(y \geq 0)$ for program PARA. We take φ and p to be $y \geq 0$. Premise I1 for this case is

$$\underbrace{\cdots \wedge y = 0}_{\Theta} \quad \rightarrow \quad y \geq 0$$

which is obviously valid. For premise I2, we consider first the verification condition for transition ℓ_1

$$\underbrace{\cdots \wedge y' = y + 1}_{\rho_{\ell_1}} \wedge \; y \geq 0 \quad \rightarrow \quad y' \geq 0$$

which is also valid. All other transitions preserve the value of y and therefore trivially preserve $\varphi : \; y \geq 0$. Premise I3 holds since $\varphi = p$.

Proof Diagrams

As we become more experienced in conducting proofs according to rule INV, proofs need not be presented with full formal detail. However, identification of the structure of the verification conditions and how transitions may lead from a state satisfying some φ_i into a state satisfying some φ_j is helpful in increasing confidence in the correctness of the proof. Here we propose to represent this information by proof diagrams.

A *proof diagram* is a directed graph consisting of a finite set of nodes N and a set of directed edges E connecting the nodes. Each node n_i is labeled by an assertion φ_i, and each edge is labeled with a name of a transition. A subset of the nodes $I \subseteq N$ is identified as the set of *initial nodes*. An example of a proof diagram is presented in Fig. 2.

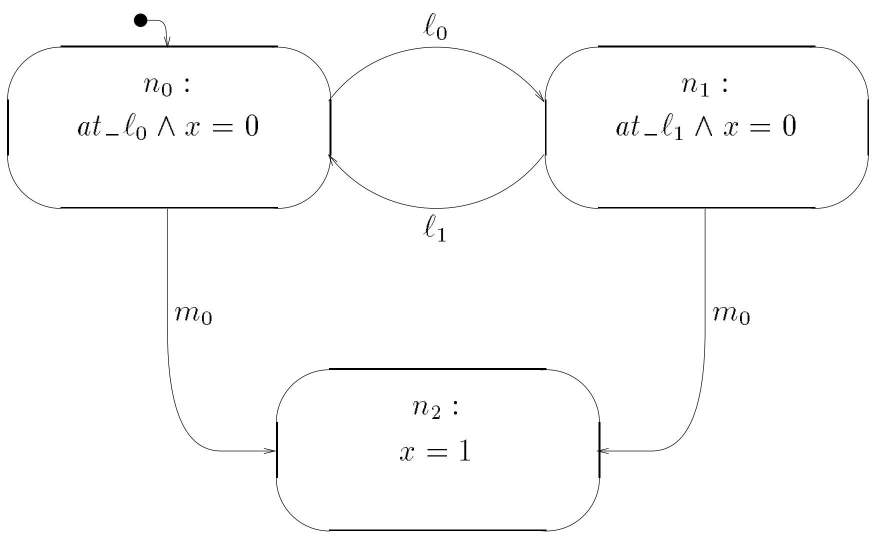

Figure 2: A Proof Diagram

If node n_i is connected to node n_j by an edge labeled by τ, we say that n_j is a τ-successor of n_i.

A proof diagram is defined to be *sound* if, for every nonterminal node $n \in M$ labeled by assertion φ and every transition $\tau \in \mathcal{T}$:

- If $n_1, \ldots n_k$ are the τ-successors of n for some $k > 0$, then the verification condition $\{\varphi\}\ \tau\ \{\bigvee_{i=1}^{k} \varphi_k\}$ is valid.
- If n has no τ-successor then the verification condition $\{\varphi\}\ \tau\ \{\varphi\}$ is valid.

It is obvious that a sound proof diagram identifies a set of verification conditions that can serve as premise I2 of rule INV. To account for premises I1 and I3, we define a diagram D to be *valid with respect to* assertion p if D is sound and the following implications are valid:

$$\Theta \rightarrow \varphi_I, \qquad \varphi_N \rightarrow p,$$

using the notation φ_K as an abbreviation for the disjunction $\bigvee_{i \in K} \varphi_i$, where K is either I or N[1].

Claim 1 *A diagram that is valid with respect to assertion p establishes the validity of the formula $\Box p$ over program P.*

Taking p to be $x = 0 \rightarrow at\_\ell_{0,1}$, it is not difficult to show that the proof diagram of Fig. 2 is valid with respect to p. This establishes the validity of the safety property $\Box(x = 0 \rightarrow at\_\ell_{0,1})$ over program PARA.

Statechart Conventions

There are several conventions inspired by the visual language of Statecharts [Har87] that improve the presentation and readability of proof diagrams. We extend the notion of a directed graph into a structured directed graph by allowing *compound nodes* that may encapsulate other nodes, and edges that may depart or arrive at compound nodes. A node that does not encapsulate other nodes is called a *basic node*.

We use the following conventions:

- Labels of compound nodes: a diagram containing a compound node n, labeled by an assertion φ and encapsulating nodes $n_1, \ldots, n_k$ with assertions $\varphi_1, \ldots, \varphi_k$, is equivalent to a diagram in which n is unlabeled and nodes $n_1, \ldots, n_k$ are labeled by $\varphi_1 \wedge \varphi, \ldots, \varphi_k \wedge \varphi$.
- Edges entering and exiting compound nodes: a diagram containing an edge e connecting node A to a compound node n encapsulating nodes $n_1, \ldots, n_k$ is equivalent to a diagram in which there is an edge connecting A to each n_i, $i = 1, \ldots, k$, with the same label as e. Similarly, an edge e connecting the compound node n to node B is the same as having a separate edge connecting each n_i, $i = 1, \ldots, k$, to B with the same label as e.

With these conventions we can redraw the proof diagram of Fig. 2 as shown in Fig. 3. Note that the common conjunct $x = 0$ has been factored out of nodes n_0 and n_1 and now appears as the label of the compound node encapsulating them.

[1] Note the abuse of notation by which we use I and N to denote sets of nodes as well as sets of the indices of these nodes. We hope that the ensuing ambiguity can always be resolved by the context.

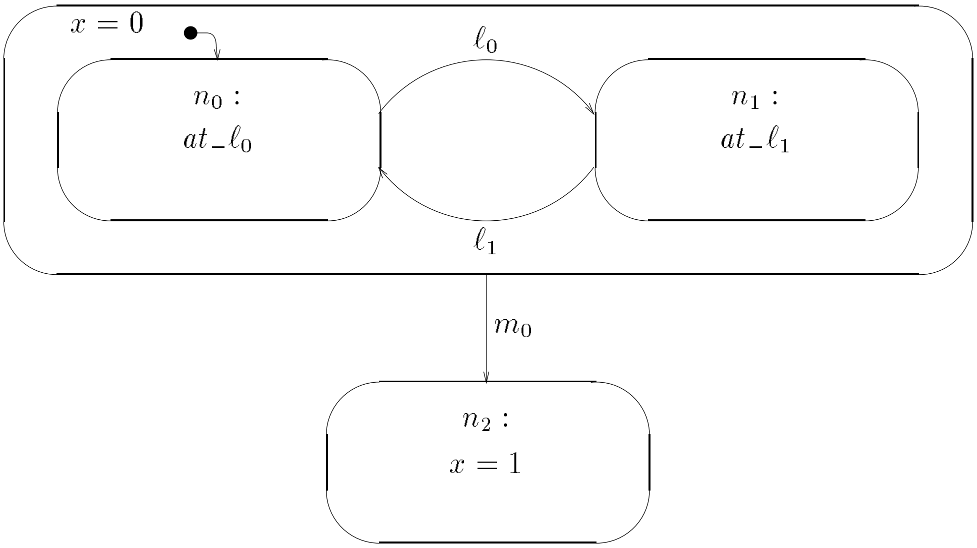

Figure 3: A Structured Proof Diagram

3 Real-Time Systems

The next model we consider introduces the metric aspect of time, and provides a measure for the time-distance between events as well as for the duration of activities in the system.

The specific model we present here was introduced and discussed in [HMP90],[HMP91], [HMP92]. A closely related model was presented in [AL92]. Many of the Process Algebra extensions to real-time, such as [NSY92], [MT90], and many others listed in [Sif91], are based on very similar assumptions.

3.1 Computational Model: Timed Transition System

As the time domain we take the nonnegative reals R^+. In some cases, we also need its extension $\mathsf{R}^\infty = \mathsf{R}^+ \cup \{\infty\}$.

A *timed transition system* (TTS) $S = \langle V, \Theta, \mathcal{T}, l, u \rangle$ consists of the components V, Θ, and $\mathcal{T}$, defined the same as for a fair transition system, and the two new components:

- A *minimal delay* $l_\tau \in \mathsf{R}^+$ (also called *lower bound*) for every transition $\tau \in \mathcal{T}$.
- A *maximal delay* $u_\tau \in \mathsf{R}^\infty$ (also called *upper bound*) for every transition $\tau \in \mathcal{T}$. It is required that $u_\tau \geq l_\tau$ for all $\tau \in \mathcal{T}$.

Note that the components $\mathcal{J}$ and $\mathcal{C}$ are eliminated.

We introduce a special variable T, sometimes called the *clock variable*. At any point in an execution of a system, T has a value over R^+ representing the current time. The set of variables $V_T = V \cup \{T\}$ is called the set of *situation variables*. A type consistent interpretation of V_T is called a *situation*, and the set of all situations is denoted by Σ_T.

Often, we represent a situation as a pair $\langle s, t\rangle$ where s is a state and $t \in \mathsf{R}^+$ is the interpretation of the clock T.

To simplify the formalism, we assume that all transitions are *self disabling*. This means that no transition $\tau \in \mathcal{T}$ can be applied twice in succession to any state, implying that τ is disabled on any τ-successor of any state, i.e., $\tau(\tau(s)) = \phi$ for any s.

Computations

A *computation* of a timed transition system is an infinite sequence of situations

$$\sigma \;:\; \langle s_0, t_0\rangle,\ \langle s_1, t_1\rangle,\ \langle s_2, t_2\rangle,\ \ldots\ ,$$

satisfying:

- *Initiation:* $s_0 \models \Theta$ and $t_0 = 0$.
- *Consecution:* For each $j = 0, 1, \ldots$,
  - Either $t_j = t_{j+1}$ and $s_{j+1} \in \tau(s_j)$ for some transition $\tau \in \mathcal{T}$, or
  - $s_j = s_{j+1}$ and $t_j < t_{j+1}$. We refer to this step as a *tick step*, implying that time has progressed.
- *Lower bound:* For every transition $\tau \in \mathcal{T}$ and position $j \geq 0$, if τ is taken at j, there exists a position i, $i \leq j$, such that $t_i + l_\tau \leq t_j$ and τ is enabled on $s_i, s_{i+1}, \ldots, s_j$. This implies that τ must be continuously enabled for at least l_τ time units before it can be taken.
- *Upper bound:* For every transition $\tau \in \mathcal{T}$ and position $i \geq 0$, if τ is enabled at position i, there exists a position j, $i \leq j$, such that $t_i + u_\tau \geq t_j$ and

  either τ is not enabled at j,
  or τ is taken at j.

  In other words, τ cannot be continuously enabled for more than u_τ time units without being taken.
- *Time Divergence:* As i increases, t_i grows beyond any bound.

Unlike the untimed case, it is not necessary to require that every state has at least one transition enabled on it. This is because, even if all transitions are disabled, we can always take tick steps which ensures that all computations are infinite. Consequently, we no longer need the *idling transition* τ_I and omit it from the systems corresponding to programs.
The requirement of time divergence excludes *Zeno computations* in which there are infinitely many state-changes within a finite time interval [AL92].

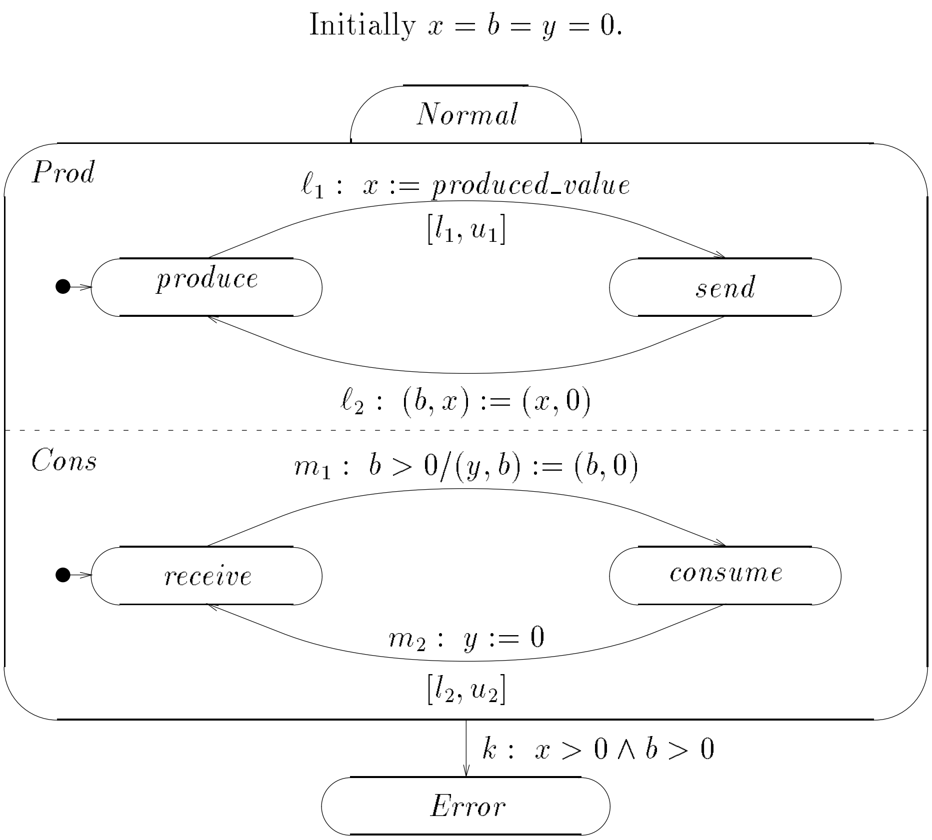

Figure 4: PROD-CONS a Producer Consumer system.

System Description by Timed Statecharts

A very convenient specification of timed systems can be obtained by extending the visual notation of Statecharts [Har87] by annotating each transition with a pair of numbers $[l, u]$, denoting the lower and upper time bounds of that transition. As an example, we present in Fig. 4 a timed specification of a producer-consumer system.

The diagram consists of two processes (automata): *Prod* and *Cons*, which operate concurrently. Process *Prod* represents a producer that produces a positive value in x and places it in the buffer variable b. Process *Cons* waits for b to become positive and then copies b to its working variable y while resetting b to 0.

A label of a transition in this Statechart specification has the form

$$name : c/assignment,$$

where *name* is an optional name of the transition (with no semantic meaning), c is a *triggering condition* which causes the transition to become enabled, and *assignment* is an optional assignment which is executed when the transition is taken. When the transition has the trivial triggering condition T, such as transition ℓ_1 in the diagram, we omit the separator '/' from the label. In this case, the transition is enabled whenever the state from which it departs (state *produce* in the diagram) is active.

In addition, each transition is optionally labeled by a pair of real numbers $[l, u]$, which specifies the minimal and maximal delays of the transition. Transitions that are not explicitly labeled are considered to be immediate, i.e., to have the time bounds $[0, 0]$.

An interesting analysis question one would like to address in this situation is the conditions under which state *Error* is guaranteed to be unreachable. A simple calculation implies that $l_1 > u_2$ is a sufficient condition for the unreachability of *Error*. In the sequel we discuss an approach to the verification of such statements.

Timed Statecharts as a TTS

While we refer the reader to [KP92] for a full definition of the semantics of timed Statecharts, we show here how statechart PROD-CONS of Fig. 4 can be viewed as a timed transition system.

As we see in the diagram, a Statechart contains *basic states* which do not contain other states and *compound states* which do. We refer to the direct descendants of a compound states as its *children*. Thus, the children of state *Prod* are *produce* and *send*, the children of *Cons* are *receive* and *consume*, and the children of *Normal* are *Prod* and *Cons*. States *Prod* and *Cons* are (exclusive) *or-states*. A basic state is considered active if the system is currently at this state. An or-state is active if precisely one of its children is active. State *Normal*, on the other hand, is an *and-state*. An and-state is active if all of its children are active. Or- and and-states correspond to sequential and parallel composition of their children, respectively.

The control variable π in the TTS representation of Statecharts ranges over sets of basic states. We adopt the notation by which a reference to a compound state names the set of basic states that are its descendants. For example, *Prod* refers to the set $\{produce, consume\}$.

Following is the identification of the constituents of the timed transition system corresponding to statechart PROD-CONS.

- *State Variables*: $\{\pi, x, b, y\}$, where π ranges over subsets of $\{produce, send, receive, consume, Error\}$.
- *Initial Condition*: Given by

  $$\Theta: \ \pi = \{produce, receive\} \wedge x = b = y = 0.$$

- *Transitions, lower and upper bounds*: Listed in the following table. For simplicity, we omitted all conjuncts of the form $u' = u$ for any state variable u.

τ	ρ_τ	l_τ	u_τ
ℓ_1	$produce \in \pi \ \wedge \ \pi' = \pi - \{produce\} \cup \{send\} \ \wedge \ x' > 0$	l_1	u_1
ℓ_2	$send \in \pi \ \wedge \ \pi' = \pi - \{send\} \cup \{produce\} \ \wedge \ b' = x \ \wedge \ x' = 0$	0	0
m_1	$receive \in \pi \ \wedge \ \pi' = \pi - \{receive\} \cup \{consume\} \ \wedge \ b > 0 \ \wedge \ y' = b \ \wedge \ b' = 0$	0	0
m_2	$consume \in \pi \ \wedge \ \pi' = \pi - \{consume\} \cup \{receive\} \ \wedge \ y' = 0$	l_2	u_2
k	$\pi \cap Normal \neq \phi \ \wedge \ x > 0 \ \wedge \ b > 0 \ \wedge \ \pi' = \pi - Normal \cup \{Error\}$	0	0

Note that transition ℓ_1 takes any positive value as a "produced value." The reference to *Normal* in the relation for transition k, is equivalent to a reference to the set $\{produce, send, receive, consume\}$.

Timed Extension of the Textual Language

In the previous section (subsection 2.2), we introduced the simple programming language SPL for the qualitative model. What extensions, if any, are necessary to deal with real-time?

On the lowest level, very few extensions are necessary. At the minimum, it is only necessary to assign time bounds to the transitions associated with statements of the program. For example, we can assign uniform time bounds $l_\tau = L$ and $u_\tau = U$ to every transition.

It is obvious that with this time bounds assignment each SPL program can be viewed as a TTS.

With this timing assignment, we may reconsider a program such as PARA and claim for it some stronger properties. For example, the property of termination can now be quantified by saying that the program terminates within $3 \cdot U$ time units. In the following subsections we will show how such properties are specified and verified.

To distinguish between the interpretation of a program P as a fair transition system and its interpretation as a timed transition system (when provided time bounds for its transitions), we denote the latter as P_T.

3.2 Requirement Specification Languages

To specify properties of timed systems, we use the language of temporal logic with appropriate extensions. There have been several proposals for such extensions. Here we present only two of them.

One approach to the specification of timing properties presented in [HMP91] introduces a bounded version of most temporal operators, obtained by subscripting the operator by a timing constraint. For example, the formula $\Diamond_{\leq d}\ q$ holds at position i of a timed computation $\sigma :\ (s_0, t_0), (s_1, t_1), \ldots$ iff there exists a j, $i \leq j$, such that $t_j \leq t_i + d$ and q holds at j. With this notation, we can write the formula $p \Rightarrow \Diamond_{\leq d}\ q$, which expresses the *bounded response* property stating that every p should be followed by an occurrence of a q, after no more than d time units.

This language for the specification of timing properties, which we refer to as *metric temporal logic* (MTL), has been advocated in [KVdR83], [KdR85], and [Koy90], although an early proposal in [BH81] can be viewed as a precursor to this specification style.

Another approach to the specification of timed properties introduces a temporal function $\Gamma(\varphi)$, called the *age* of the formula φ. The age function measures the length of the largest interval, extending through the past to the present, in which φ has been continuously true. More precisely, the value of $\Gamma(\varphi)$ at position j in a computation σ is defined to be

> the largest t such that, for some $i \leq j$, $t = t_j - t_i$ and φ holds at all positions $i, \ldots, j$, or 0 if $(\sigma, j) \not\models \varphi$.

We denote by TL_Γ the logic obtained by extending temporal logic with the age function. Note that the value of $\Gamma(\mathit{true})$ at situation $\langle s_i, t_i \rangle$ is always t_i, the current value of the clock variable T. Consequently, we allow formulas in TL_Γ to refer explicitly to the clock variable T. In this respect, TL_Γ can be viewed as an extension of the *Explicit Clock Temporal Logic* considered, for example, in [PH88], [HLP90], and [Ost90].

Referring to the clock T, we can specify the bounded response property by the formula $p \wedge T = t_0 \Rightarrow \Diamond(q \wedge T \leq t_0 + d)$.

An assertion that may refer to the clock variable T or contain age expressions of the form $\Gamma(\psi)$, where ψ is an assertion, is called a *timed assertion*.

3.3 Verification of Age Formulas

For verifying properties specified by TL_Γ, we develop an extended version of rule INV.

Before presenting the rule itself, we present some axioms governing the behavior of age expressions:

AGE-RANGE :	$0 \leq \Gamma(\psi) \leq T$	for every formula ψ
AGE-FALSE :	$\neg\psi \ \rightarrow \ \Gamma(\psi) = 0$	for every formula ψ
UPPER-BOUND :	$\Gamma(En(\tau)) \leq u_\tau$	for every transition $\tau \in \mathcal{T}$

We may use these axioms freely in any reasoning step. Note that a consequence of AGE-RANGE is that $T = 0$ implies $\Gamma(\psi) = 0$ for every ψ.

Verification Conditions

In preparation for rule INV, we introduced the verification condition $\{p\}\,\tau\,\{q\}$ whose validity ensures that every τ-successor of a p-state satisfies q. When considering computations of timed transition system, there are two ways to get from a situation to its successor: by taking a transition or by letting time progress (a tick step). Consequently, we introduce two verification conditions.

- The condition $\{p\}\,\tau\,\{q\}_T$ is given by $\rho^*_\tau \ \wedge \ p \ \rightarrow \ q'$, where ρ^*_τ stands for

  $\rho_\tau \ \wedge \ T' = T \ \wedge \ l_\tau \leq \Gamma(En(\tau)) \leq u_\tau$.

- The condition $\{p\}\,tick\,\{q\}$ is given by $\rho_{tick} \ \wedge \ p \ \rightarrow \ q'$, where ρ_{tick} stands for

  $V' = V \ \wedge \ T' > T \ \wedge \ \bigwedge_{\tau\in\mathcal{T}}(\Gamma'(En(\tau)) \leq u_\tau)$.

In any of these formulas we may need to evaluate the primed version of $\Gamma(r)$, denoted by $\Gamma'(r)$ for some assertion r. This is given by

$$\Gamma'(r) \quad = \quad \textbf{if } r' \textbf{ then } \Gamma(r) + T' - T \textbf{ else } 0.$$

A Rule for Timed Invariances

The following rule can be used to establish the validity of the formula $\Box p$ for a (possibly timed) assertion p over a timed transition system P_T.

$$\begin{array}{|llccc|}
\hline
\text{T-INV} & \text{I1.} & \Theta_T & \rightarrow & \varphi \\
 & \text{I2.} & \{\varphi\} & \mathcal{T} & \{\varphi\}_T \\
 & \text{I3.} & \{\varphi\} & tick & \{\varphi\} \\
 & \text{I3.} & \varphi & \rightarrow & p \\
\cline{2-5}
 & & & \Box p & \\
\hline
\end{array}$$

Premise I1 refers to Θ_T, the timed version of the initial condition, which is defined as $\Theta \ \wedge \ T = 0$. Rule T-INV is complete for proving timed invariance formulas.

To present elaborate proofs, we again recommend proof diagrams. The only difference between the timed and untimed case is that, in the timed case, assertions may refer to the clock variable T and contain age expressions.

Moving to the timed case, we add to the definition of a sound (and valid) proof diagram the requirement

- For every $n_i \in N$ it is required that the tick verification condition $\{\varphi_i\}\ \mathit{tick}\ \{\varphi_i\}$ is valid, implying that each of the assertions is preserved under the progress of time.

Let us consider the property stating that program PARA$_T$ terminates within 15 time units. In MTL, this can be specified by the formula $\Theta \Rightarrow \Diamond_{\leq 15}(at\_\ell_2 \wedge at\_m_1)$. In this form, this property appears to be a response property. However, this property is actually a *safety* property. We refer the reader to [Hen91] and [Pnu92] where it is pointed out that many liveness properties become safety properties when we consider their bounded version.

We will use rule T-INV via proof diagrams to establish the same property, expressed by the invariance

$$\Box(T \leq 15 \ \vee \ (at\_\ell_2 \ \wedge \ at\_m_1)).$$

This formula states that if more than 15 time units have elapsed, then the program must have terminated.

The proof diagram presented in Fig. 5 establishes the validity of this formula over program PARA$_T$.

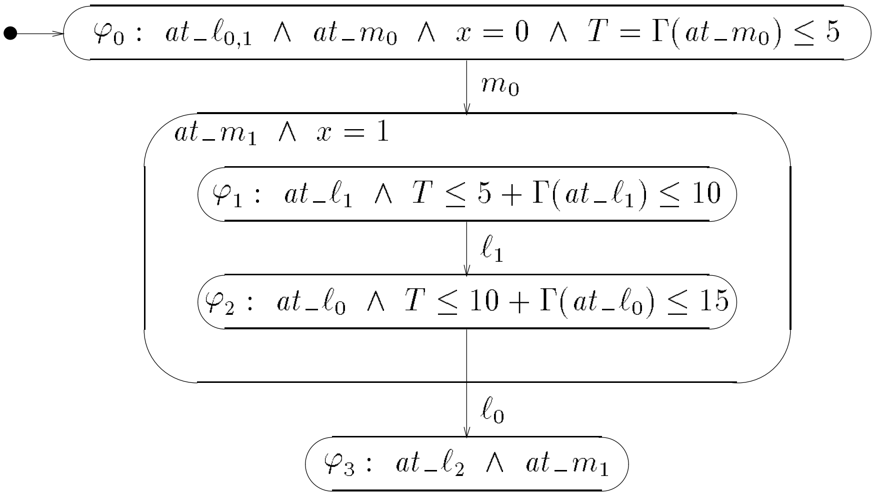

Figure 5: A Timed Proof Diagram

It is not difficult to show that the diagram is valid with respect to the assertion $T \leq 15 \ \vee \ (at\_\ell_2 \wedge at\_m_1)$.

Proving Untimed Properties of Timed Systems

The previous example concentrated on proving timed properties, i.e., properties in which time is explicitly mentioned. Another interesting class of properties consists of properties that do not refer to time directly but whose validity over a program P_T is a consequence of the timing constraints satisfied by the computations of P_T.

For example, the property $\square(y \leq 3)$ is valid over all computations of program PARA_T with uniform time bounds $[1, 5]$. It is certainly not valid for the untimed computations of PARA_T.

In Fig. 6 we present a proof diagram that is valid with respect to the assertion $y \leq 3$.

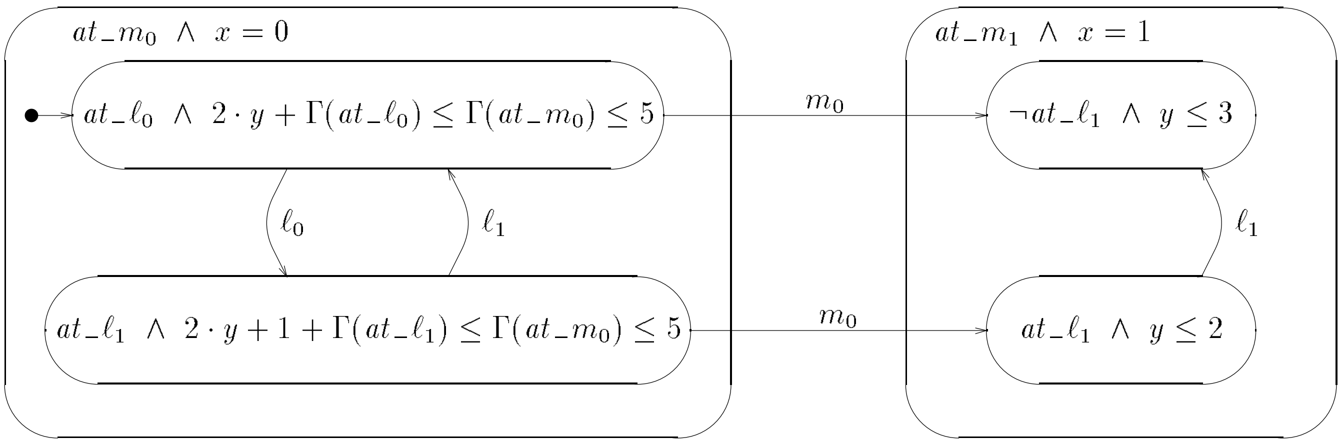

Figure 6: A Timed Invariance Proof Diagram

4 Hybrid Systems

The last model presented here is that of *hybrid systems*. This model allows some components of the system to modify their variables continuously, in addition to discrete components that modify their variables by transitions that change the values of variables in zero time.

4.1 Computational Model: Phase Transition System

A *phase transition system* (PTS) is given by $\Phi : \langle V, \Theta, \mathcal{T}, \mathcal{A}, l, u \rangle$. The components V, Θ, $\mathcal{T}$, l, and u are defined identically to their definition for a timed transition system. The set $V = V_d \cup V_c$ is partitioned into V_d the set of *discrete variables* and V_c the set of *continuous variables*. Continuous variables have always the type *real*. The new component is

- $\mathcal{A}$: A finite set of *activities*. Each activity $\alpha \in \mathcal{A}$ is a conditional differential equation of the form:

  $$p \ \rightarrow \ \dot{x} = e,$$

where p is a predicate over V_d called the *activation condition* of α, $x \in V_c$ is a continuous state variable, and e is an expression over V. We say that the activity α *governs* variable x. Activity α is said to be *active* in state s if its activation condition p holds on s. Otherwise, α is said to be *passive*. Let G_x denote the set of activities that govern variable x.

It is required that the activation conditions of the activities that govern the same variable x be exhaustive and exclusive, i.e., at least one of them holds on any state but no two of them can hold simultaneously.

For simplicity, we require that transitions whose enabling condition depends on a continuous variable be immediate, i.e., have the time bounds $[0, 0]$.

Activity Successors

Consider a phase transition system Φ. An *activity selection* is a mapping $g : V \mapsto \mathcal{A}$, assigning to each continuous variable $x \in V_c$ an activity $g(x) \in G_x$ in its governing set.

Let $\langle s_1, t_1\rangle$ and $\langle s_2, t_2\rangle$ be two situations of Φ with $t_1 < t_2$. An *evolution* $\langle g, F\rangle$ from $\langle s_1, t_1\rangle$ to $\langle s_2, t_2\rangle$ consists of an activity selection g and a set of functions $F : \{f_x(t) \mid x \in V\}$, one for each variable $x \in V$, that are differentiable in the interval $[t_1, t_2]$ and satisfy the following requirements for every $x \in V$:

- $f_x(t_1) = s_1[x]$ and $f_x(t_2) = s_2[x]$. Thus, the values of $f_x(t)$ at the boundaries of the interval $[t_1, t_2]$ agree with the interpretation of x by s_1 and s_2, respectively.

- If x is a continuous variable, and $p \to \dot{x} = e$ is the activity selected for x by g, then p holds at s_1 and f_x satisfies the differential equation

  $$\frac{d}{dt} f_x(t) \quad = \quad e(F)$$

  holds in the interval $[t_1, t_2]$, where the expression $e(F)$ is obtained from e by replacing each occurrence of a variable y by the function $f_y(t)$.

- If $x \in V_d$ is a discrete variable, then $f_x(t) = s_1[x]$ for every $t \in [t_1, t_2]$.

- For every t, $t_1 \le t < t_2$, there is no immediate transition which is enabled on the state $s(t)$ defined as the interpretation assigning to each variable $y \in V$ the value $f_y(t)$.

In such a case we say that the situation $\langle s_2, t_2\rangle$ is an *activity successor* of the situation $\langle s_1, t_1\rangle$. Note that there exists at most one evolution from $\langle s_1, t_1\rangle$ to $\langle s_2, t_2\rangle$, for a given activity selection g. In fact, the functions F are uniquely determined by the selection g and the situation $\langle s_1, t_1\rangle$.

Sampling Computations

A *sampling computation* of a phase transition system $\Phi : \langle V, \Theta, \mathcal{T}, \mathcal{A}, l, u\rangle$ is an infinite sequence of situations

$$\sigma \;:\; \langle s_0, t_0\rangle,\ \langle s_1, t_1\rangle,\ \langle s_2, t_2\rangle,\ \ldots\ ,$$

satisfying:

- *Initiation:* $s_0 \models \Theta$ and $t_0 = 0$.
- *Consecution:* For each $j = 0, 1, ...,$
  - Either $t_j = t_{j+1}$ and $s_{j+1} \in \tau(s_j)$ for some transition $\tau \in \mathcal{T}$ — transition τ taken at j, or
  - $\langle s_{j+1}, t_{j+1}\rangle$ is an activity successor of $\langle s_j, t_j\rangle$ (implying $t_j < t_{j+1}$) — a continuous phase is executed at step j.
- *Bounds:* Each transition $\tau \in \mathcal{T}$,
  - *can* be taken only after being continuously enabled for at least l_τ time units, and
  - *cannot* be continuously enabled for more than u_τ time units without being taken.
- *Time Divergence:* As i increases, t_i grows beyond any bound.

System Description by Hybrid Statecharts

Hybrid systems can be conveniently described by an extension of timed Statecharts called *hybrid Statecharts*. The main extension is that states may be labeled by (unconditional) differential equations. The implication is that the activity associated with the differential equation is active precisely when the state it labels is active.

We illustrate this form of description by the example of *Cat and Mouse* taken from [MMP92]. At time $T = 0$, a mouse starts running from a certain position on the floor in a straight line towards a hole in the wall, which is at a distance X_0 from the initial position. The mouse runs at a constant velocity v_m. After a delay of Δ time units, a cat is released at the same initial position and chases the mouse at velocity v_c along the same path. Will the cat catch the mouse, or will the mouse find sanctuary while the cat crashes against the wall?

The Statechart in Fig. 7 describes the possible scenarios.

This diagram illustrates the typical interleaving between continuous activities and discrete state changes, which in this example only involves changes of control.

The specification (and underlying phase transition system) uses the continuous state variables x_m and x_c, measuring the distance of the mouse and the cat, respectively, from the wall. It refers to the constants X_0, v_m, v_c, and Δ.

The idea of using Statecharts with continuous activities associated with certain states (usually basic ones) was already suggested in [Har84].

The Underlying Phase Transition System

Following the graphical representation, we will now identify the phase transition system underlying the picture of Fig. 7.

- *State Variables:* Given by $V_c = \{x_c, x_m\}$ and $V_d = \{\pi\}$. The variable π is a control variable whose value is a set of basic states of the Statechart.

Initially $x_c = x_m = X_0$.

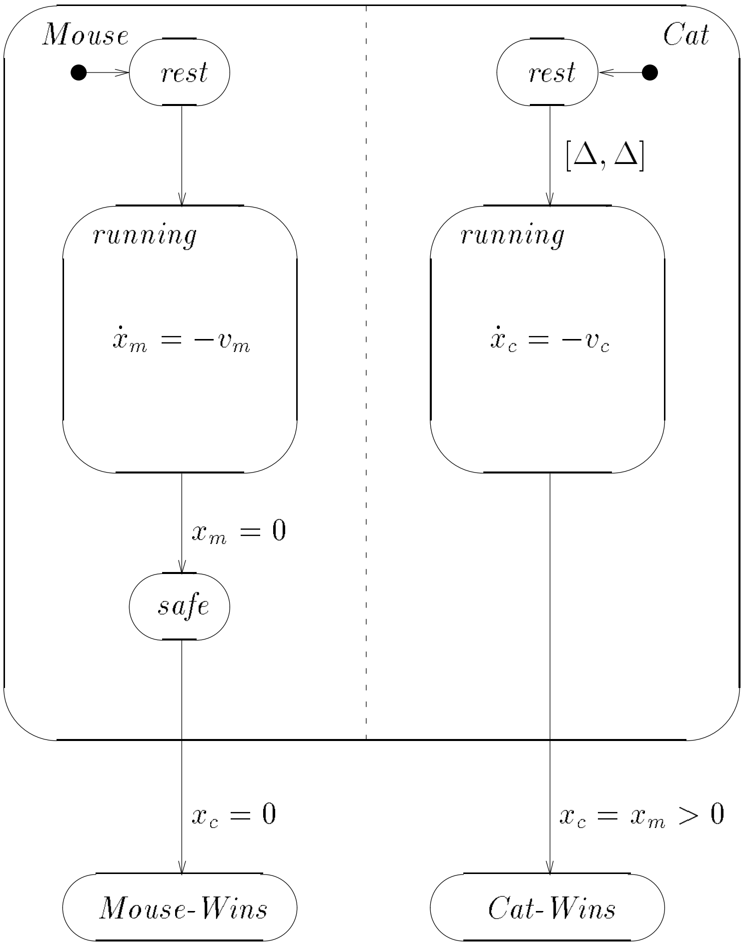

Figure 7: Specification of Cat and Mouse.

- *Initial Condition:* Given by

$$\Theta \;:\; \pi = \{Mouse.rest, Cat.rest\} \;\wedge\; x_c = x_m = X_0$$

- *Transitions:* Listed together with the transition relations associated with them. The transitions are named according to the states from which they depart.

$$\begin{array}{rcl}
Mouse.rest & : & Mouse.rest \in \pi \;\wedge\; \pi' = \pi - \{Mouse.rest\} \cup \{Mouse.running\} \\
Cat.rest & : & Cat.rest \in \pi \;\wedge\; \pi' = \pi - \{Cat.rest\} \cup \{Cat.running\} \\
Mouse.running & : & Mouse.running \in \pi \;\wedge\; x_m = 0 \;\wedge \\
& & \qquad \pi' = \pi - \{Mouse.running\} \cup \{Mouse.safe\} \\
Mouse.safe & : & Mouse.safe \in \pi \;\wedge\; x_c = 0 \;\wedge\; \pi' = Mouse\text{-}Wins \\
Cat.running & : & Cat.running \in \pi \;\wedge\; x_c = x_m > 0 \;\wedge\; \pi' = Cat\text{-}Wins
\end{array}$$

- *Activities:* Four activities α_m and α_c represent the running activities of the two participants. Their equations sets are given by

$$\begin{array}{lcrcl}
\alpha_m^{on} & : & \mathit{Mouse.running} \in \pi & \rightarrow & \dot{x}_m = -v_m \\
\alpha_m^{off} & : & \mathit{Mouse.running} \notin \pi & \rightarrow & \dot{x}_m = 0 \\
\alpha_c^{on} & : & \mathit{Cat.running} \in \pi & \rightarrow & \dot{x}_c = -v_c \\
\alpha_c^{off} & : & \mathit{Cat.running} \notin \pi & \rightarrow & \dot{x}_c = 0
\end{array}$$

- *Time Bounds:* For transition *Cat.rest* they are $[\Delta, \Delta]$. All other transitions are immediate.

System Description by Textual Programs

It is possible to extend the simple programming language SPL to represent timed and hybrid behaviors as well. The resulting language is a subset of the language *Statext* introduced in [KP92], which is shown there to have expressive power equal to that of hybrid Statecharts.

The main extensions are to allow a differential equation as an SPL statement, and the *preemption* statement $S_1 \mathsf{U} S_2$. The intended meaning of the *preemption* statement is that S_1 is executed until a first step of S_2 is taken. When the first step of S_2 is taken, the remainder of statement S_1 is discarded.

We refer the reader to [KP92] for a sampling-computation semantics of this extension of SPL.

In Fig. 8, we present an extended SPL program that can be viewed as a hybrid version of program PARA.

$$\begin{array}{c}
y\text{: }\textbf{integer where } y = 0 \\
x\text{: }\textbf{real where } x = 0 \\[1em]
\left[\begin{array}{l}
\ell_0 : \ \textbf{while } x < 1 \textbf{ do} \\
\qquad \left[\ell_1 : \ y := y + 1\right] \\
\ell_2 :
\end{array}\right]
\quad \Big\| \quad
\left[\begin{array}{l}
[m_0 : \ \dot{x} = 0.2] \quad \mathsf{U} \quad [m_1 : \ \textbf{await } x \geq 1] \\
m_2 :
\end{array}\right] \\[1em]
- \ P_1 \ - \qquad\qquad\qquad\qquad\qquad - \ P_2 \ -
\end{array}$$

Figure 8: Program PARA_H: A Hybrid textual Program

In this program, process P_2 represents a continuous component that lets x grows linearly from 0 until it reaches a value $x \geq 1$. At that point, statement m_1 intervenes and shuts off the continuous process. Process P_1 is very similar to process P_1 in program PARA. It loops, incrementing y, as long as $x < 1$. Once process P_1 detects that $x \geq 1$, it terminates.

The following interprets program PARA_H as a phase transition system.

$$\begin{array}{lcl}
V & = & V_c : \{x\} \ \cup \ V_d : \{\pi, y\} \\
\Theta & : & \pi = \{\ell_0, m_0, m_1\} \ \wedge \ x = y = 0
\end{array}$$

$\mathcal{T}$: $\{\ell_0, \ell_1, m_1\}$ with transition relations:

$$\rho_{\ell_0} \;:\; \ell_0 \in \pi \;\wedge\; \left(\begin{array}{c} x = 0 \;\wedge\; \pi' = \pi - \{\ell_0\} \cup \{\ell_1\} \\ \vee \\ x \neq 0 \;\wedge\; \pi' = \pi - \{\ell_0\} \cup \{\ell_2\} \end{array}\right) \;\wedge\; x' = x \;\wedge\; y' = y$$

$$\rho_{\ell_1} \;:\; \ell_1 \in \pi \;\wedge\; \pi' = \pi - \{\ell_1\} \cup \{\ell_0\} \;\wedge\; x' = x \;\wedge\; y' = y + 1$$

$$\rho_{m_1} \;:\; m_1 \in \pi \;\wedge\; x \geq 1 \;\wedge\; \pi' = \pi - \{m_0, m_1\} \cup \{m_2\} \;\wedge\; x' = 1 \;\wedge\; y' = y$$

$\mathcal{A}$: $\{\alpha_1, \alpha_2\}$ given by:

$\alpha_1 \;:\; m_0 \in \pi \;\rightarrow\; \dot{x} = 0.2$

$\alpha_2 \;:\; m_0 \notin \pi \;\rightarrow\; \dot{x} = 0$

$l = 1$ and $u = 5$ for ℓ_0 and ℓ_1; m_1 is immediate.

Requirement Specification Languages

At present, no special extensions to the requirement specification languages have been identified for hybrid systems. As in the case of the real-time model, we use either MTL or TL_Γ for specifying properties of hybrid systems.

4.2 Verification of Age Formulas

Here we only consider verification of TL_Γ formulas over hybrid systems. The major extension relative to the real-time model is in the verification conditions.

The verification condition $\{p\}\,\tau\,\{q\}_{\mathcal{T}}$ remains with no change. However, instead of using the verification condition $\{p\}\;\mathit{tick}\;\{q\}$, we define a new condition $\{p\}\;\mathit{cont}\;\{q\}$ which is intended to ensure that every continuous phase leads from a p-situation to a q-situation.

The condition $\{p\}\;\mathit{cont}\;\{q\}$ is given by a conjunction of implications $\rho^g_{cont} \;\wedge\; p \;\rightarrow\; q'$, one for each activity selection g. The relation ρ^g_{cont} stands for

$$\begin{array}{l} F^g(T) = V \;\wedge\; F^g(T') = V' \;\wedge\; T' > T \;\wedge\; \displaystyle\bigwedge_{\tau \in \mathcal{T}} (\Gamma'(En(\tau)) \leq u_\tau) \\ \wedge \\ \displaystyle\bigwedge_{x \in V_c} (p^g_x(V) \;\wedge\; \bigwedge_{\tau \in \mathcal{T}_0} \neg(\exists t : (T < t < T') : En(\tau)(F^g)) \end{array}$$

where $\mathcal{T}_0$ consists of the immediate transitions. We assume that, for a given g, we can solve the differential equations for the activities selected by g, and express the solutions as a formula involving t, V – the initial value of the variables at the beginning of the continuous step, and T – the initial value of t at the beginning of the continuous step.

For example, if the activity selected is $\dot{x} = 2$, then the solution for this selection is given by $f_x(t) = x + 2 \cdot (t - T)$. Let F^g denote the set of these solutions.

The expression $e(F^g)$ and the formula $En(\tau)(F^g)$ are obtained by replacing every occurrence of $x \in V$ by $f_x(t)$.

The first line in the formula states that the boundary values V and V' agree with the values of F^g at T and T', respectively. This line also states that $T' > T$ and that no transition becomes over-ripe as a result of the progress of time.

The second line states that the activation conditions selected by g all hold at V, and that no immediate transition becomes enabled in the middle of an evolution.

In both q' and ρ_{cont}, we need to evaluate $\Gamma'(r)$ for several assertions r. This is given by

$$\Gamma'(r) \quad = \quad \textbf{if } r \wedge r' \textbf{ then } \Gamma(r) + T' - T \textbf{ else } 0.$$

Note that, in the real-time case, it was sufficient to evaluate r at s'. This is because all state variables preserve their values in a tick step. In the hybrid case, the continuous state variables may change their values from s to s'. Consequently, $\Gamma(r)$ is incremented only if r holds at both s and s'.

In order to ensure that this definition actually measures the age of an assertion, we restrict our use of the expression $\Gamma(r)$ to assertions r that cannot change their truth value in the middle of a continuous step.

A Rule for Invariances

Having defined the two verification conditions, we can immediately formulate a rule for proving invariance formulas over hybrid systems.

$$\begin{array}{lll} \text{H-INV} & \text{I1.} & \Theta_T \rightarrow \varphi \\ & \text{I2.} & \{\varphi\}\, \mathcal{T}\, \{\varphi\}_T \\ & \text{I3.} & \{\varphi\}\, cont\, \{\varphi\} \\ & \text{I3.} & \varphi \rightarrow p \\ \hline & & \Box p \end{array}$$

We will illustrate the use of this rule on several examples.

For the first example, we prove that program PARA$_H$ also terminates within 15 time units. The proof diagram of Fig. 9 is valid with respect to the assertion $T \leq 15 \vee (at\_\ell_2 \wedge at\_m_1)$.

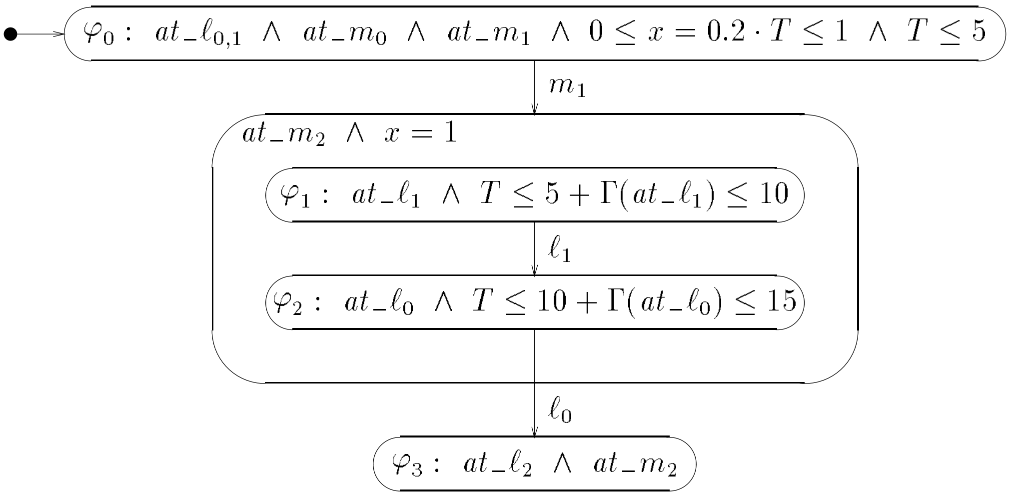

Figure 9: A Proof Diagram for Termination of PARA$_H$

The only new element in establishing this validity is checking that assertion φ_0 is preserved under a continuous step. The only relevant activity allocation function g is the one that picks α_1 for x, i.e., $g(x) = 1$. For this choice of g, the evolution of x is given by $f_x(t) = x + 0.2 \cdot (t - T)$. Consequently, the appropriate verification condition (after some simplifications) is

$$\underbrace{\pi' = \pi \ \wedge \ T' > T \ \wedge \ x' = x + 0.2 \cdot (T' - T) \ \wedge \ (\forall t : T < t < T' : x + 0.2 \cdot (t - T) < 1)}_{\rho^g_{cont}}$$

$$\wedge \quad \underbrace{at\_\ell_{0,1} \ \wedge \ at\_m_0 \ \wedge \ at\_m_1 \ \wedge \ 0 \leq x = 0.2 \cdot T \leq 1 \ \wedge \ T \leq 5}_{p}$$

$$\rightarrow$$

$$\underbrace{at\_\ell'_{0,1} \ \wedge \ at\_m'_0 \ \wedge \ at\_m'_1 \ \wedge \ 0 \leq x' = 0.2 \cdot T' \leq 1 \ \wedge \ T' \leq 5}_{q'}$$

It is not difficult to see that this implication is valid.

This establishes that program PARA_H terminates within 15 time units.

Verifying a Property of the Cat and Mouse System

Consider the property that, under the assumption

$$\frac{X_0}{v_m} \ < \ \Delta + \frac{X_0}{v_c} \tag{1}$$

all computations of the Cat and Mouse system satisfy

$$\Box(Cat.running \wedge (x_c = x_m) \ \rightarrow \ x_m = 0).$$

In Fig. 10, we present a proof diagram of this invariance property. In this diagram we use shorter names for the states of the Statechart. For example, *C.run* stands for *Cat.running*. Transitions are identified by the names of the state from which they depart. For example, *M.rest* refers to the transition connecting state *Mouse.rest* to state *Mouse.running* in the Statechart. We also use t_m for $\frac{X_0}{v_m}$, the time it takes the mouse to run the distance X_0.

It is not difficult to verify that the diagram is sound, including the preservation of all assertions under a continuous step. The only part that requires some more attention is showing that the φ_2 conjunct

$$X_0 - v_c \cdot (T - \Delta) \ > \ x_m = X_0 - v_m \cdot T$$

is maintained as long as x_m is nonnegative, which implies $T \leq t_m$. To show this, it is sufficient to show $v_c \cdot (T - \Delta) \ < \ v_m \cdot T$ which is equivalent to

$$\frac{v_m}{v_c} \ > \ 1 - \frac{\Delta}{T} \tag{2}$$

From inequality (1), we can obtain

$$\frac{v_m}{v_c} \ > \ 1 - \Delta \cdot \frac{v_m}{X_0}$$

which, using the definition of $t_m = \dfrac{X_0}{v_m}$, gives

$$\frac{v_m}{v_c} \ > \ 1 - \frac{\Delta}{t_m}. \tag{3}$$

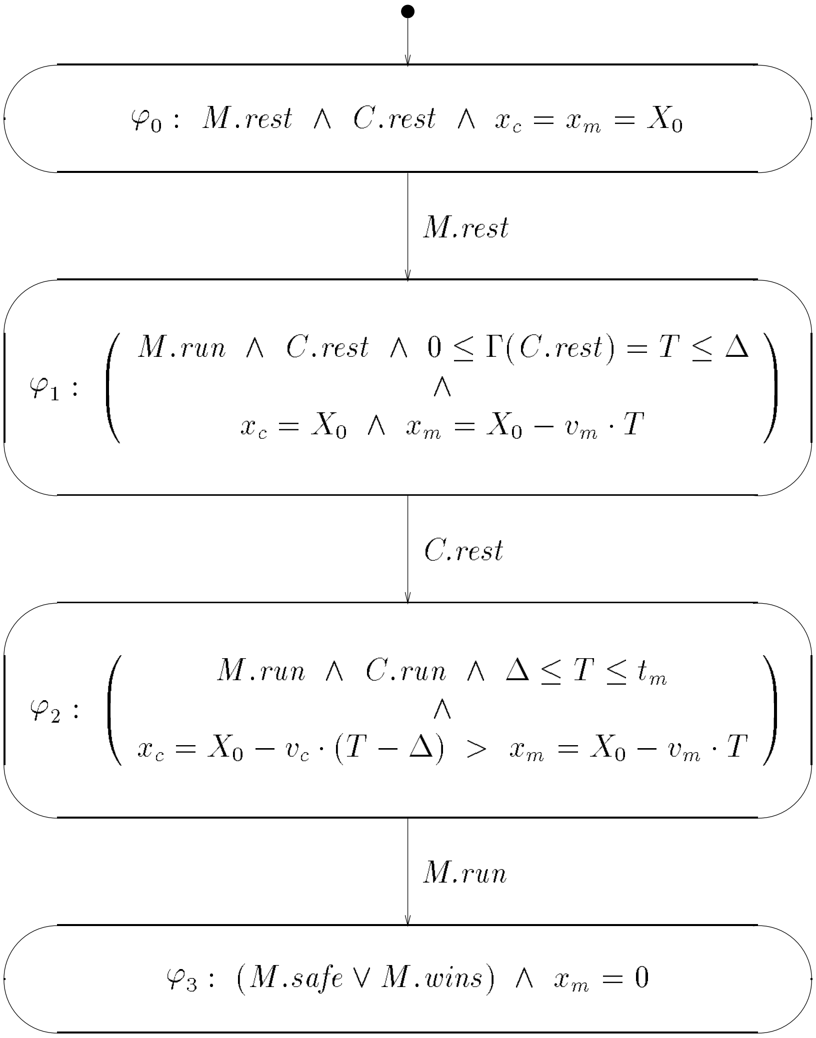

Figure 10: A Proof Diagram for the Safety of the Mouse

Since $T \le t_m$, the right hand side of (3) is not smaller than $1 - \dfrac{\Delta}{T}$ establishing (2).

It remains to show that

$$\underbrace{M.rest \;\wedge\; C.rest \;\wedge\; x_c = x_m = X_0}_{\Theta_T} \;\rightarrow\; \underbrace{M.rest \;\wedge\; C.rest \;\wedge\; x_c = x_m = X_0}_{\varphi_0} \qquad (4)$$

$$\varphi_0 \;\vee\; \cdots \;\vee\; \varphi_3 \;\rightarrow\; \big(C.run \;\wedge\; x_c = x_m \;\rightarrow\; x_m = 0\big). \qquad (5)$$

Note that, to simplify matters, we consider the initial state of all computations to be the one in which the two competitors are in $M.rest$ and $C.rest$, respectively, and that x_c and x_m are already set to X_0. Implication (4) is obviously valid. To check implication (5), we observe that both φ_0 and φ_1 imply $\neg C.run$, φ_2 implies $x_c > x_m$, and φ_3 implies $x_m = 0$.

This shows that under the assumption (1), property

$$\Box(Cat.running \wedge (x_c = x_m) \;\rightarrow\; x_m = 0)$$

is valid for the Cat and Mouse system.

References

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In 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 *Lect. Notes in Comp. Sci.* Springer-Verlag, 1992.

[BH81] A. Bernstein and P. K. Harter. Proving real time properties of programs with temporal logic. In *Proceedings of the Eighth Symposium on Operating Systems Principles*, pages 1–11. ACM, 1981.

[Har84] D. Harel. Statecharts: A visual approach to complex systems. Technical report, Dept. of Applied Mathematics, Weizmann Institute of Science CS84-05, 1984.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comp. Prog.*, 8:231–274, 1987.

[Hen91] T.A. Henzinger. Sooner is safer than later. Technical report, Stanford University, 1991.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proc. 5th IEEE Symp. Logic in Comp. Sci.*, pages 402–413, 1990.

[HMP90] T. Henzinger, Z. Manna, and A. Pnueli. An interleaving model for real time. In *5th Jerusalem Conference on Information Technology*, pages 717–730, 1990.

[HMP91] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for real-time systems. In *Proc. 18th ACM Symp. Princ. of Prog. Lang.*, pages 353–366, 1991.

[HMP92] T. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In 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 *Lect. Notes in Comp. Sci.* Springer-Verlag, 1992.

[KdR85] R. Koymans and W.-P. de Roever. Examples of a real-time temporal logic specifications. In B.D. Denvir, W.T. Harwood, M.I. Jackson, and M.J. Wray, editors, *The Analysis of Concurrent Systems*, volume 207 of *Lect. Notes in Comp. Sci.*, pages 231–252. Springer-Verlag, 1985.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.

[KP92] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lect. Notes in Comp. Sci.*, pages 591–619. Springer-Verlag, 1992.

[KVdR83] R. Koymans, J. Vytopyl, and W.-P. de Roever. Real-time programming and asynchronous message passing. In *Proc. 2nd ACM Symp. Princ. of Dist. Comp.*, pages 187–197, 1983.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In 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 *Lect. Notes in Comp. Sci.* Springer-Verlag, 1992.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[MP92] Z. Manna and A. Pnueli. Models for reactive systems. Technical report, Dept. of Comp. Sci., Stanford University, 1992.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of Concur'90*, volume 458 of *Lect. Notes in Comp. Sci.*, pages 401–415. Springer-Verlag, 1990.

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In 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 *Lect. Notes in Comp. Sci.* Springer-Verlag, 1992.

[Ost90] J.S. Ostroff. *Temporal Logic of Real-Time Systems.* Advanced Software Development Series. Research Studies Press (John Wiley & Sons), Taunton, England, 1990.

[PH88] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real time systems. In M. Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *Lect. Notes in Comp. Sci.*, pages 84–98. Springer-Verlag, 1988.

[Pnu92] A. Pnueli. How vital is liveness? In W.R. Cleaveland, editor, *Proceedings of Concur'92*, volume 630 of *Lecture Notes in Computer Science*, pages 162–175. Springer-Verlag, 1992.

[Sif91] J. Sifakis. An overview and synthesis on timed process algebra. In K.G. Larsen and A. Skou, editors, *3rd Computer Aided Verification Workshop*, volume 575 of *Lect. Notes in Comp. Sci.*, pages 376–398. Springer-Verlag, 1991.