From Timed to Hybrid Systems *

Oded Maler
INRIA/IRISA[†]

Zohar Manna
Stanford University[‡]and Weizmann Institute of Science[§]

Amir Pnueli
Weizmann Institute of Science[§]

Abstract. We propose a framework for the formal specification and verification of *timed* and *hybrid* systems. For timed systems we propose a specification language that refers to time only through *age* functions which measure the length of the most recent time interval in which a given formula has been continuously true.

We then consider hybrid systems, which are systems consisting of a non-trivial mixture of discrete and continuous components, such as a digital controller that controls a continuous environment. The proposed framework extends the temporal logic approach which has proven useful for the formal analysis of discrete systems such as reactive programs. The new framework consists of a semantic model for hybrid time, the notion of *phase transition systems*, which extends the formalism of discrete transition systems, an extended version of Statecharts for the specification of hybrid behaviors, and an extended version of temporal logic that enables reasoning about continuous change.

Keywords: Real-time, timed transitions system, hybrid systems, discrete and continuous systems, Statecharts.

Contents

*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 France-Israel project for cooperation in Computer Science, and by the European Community ESPRIT Basic Research Action Project 3096 (SPEC).

[†]INRIA/IRISA, Campus de Beaulieu, Rennes 35042, France

[‡]Department of Computer Science, Stanford University, Stanford, CA 94305

[§]Department of Applied Mathematics, Weizmann Institute, Rehovot, Israel

1 Introduction

This paper concerns the development of formal approaches for the specification, verification, and systematic construction of reliable *reactive* systems. These are systems whose role is to maintain some ongoing interaction with their environment rather than to compute some final result on termination. The correct and reliable construction of reactive programs is particularly challenging. Typical examples of reactive programs are concurrent and real-time programs, embedded systems, communication networks, air-traffic control systems, avionic and process control programs, operating systems, and many others.

There is by now a general agreement that formal specification of reactive systems contributes significantly to a better understanding of the expected functionality of the contemplated system at an early stage, thereby leading to a more reliable and efficient construction of such systems. One of the promising and widely considered approaches to the specification of reactive systems is that of *temporal logic*, which provides a natural and abstract way to describe the behavior of a reactive system.

Traditionally, temporal logic (and similar formalisms) use a *discrete events* approach to model a reactive system. This means that the behavior of a reactive system is described as a sequence of discrete events that cause abrupt changes (taking no time) in the state of the system, separated by intervals in which the system's state remains unchanged. This approach has proven effective for describing the behavior of programs and other digital systems. We refer the reader to [MP81], [Pnu86], [Lam83], and [EC82] for examples of applications of the temporal approach.

The discrete event approach is justified by an assumption that the environment, similar to the system itself, can be faithfully modeled as a digital (discrete) process. This assumption is very useful, since it allows a completely symmetrical treatment of the system and its environment and encourages modular analysis of systems, where what is considered an environment in one stage of the analysis may be considered a component of the system in the next stage.

While this assumption is justified for systems such as communication networks, where all members of the network are computers, there are certainly many important contexts in which modeling the environment as a discrete process greatly distorts reality, and may lead to unreliable conclusions. For example, a control program driving a robot within a maze or controlling a fast train must take into account that the environment with which it interacts follows continuous rules of change.

This paper suggests an extension to the temporal logic framework that will enable it to deal with continuous processes. This extension leads to an integrated approach to *hybrid* systems, i.e., systems consisting of a non-trivial mixture of discrete and continuous components, such as a digital controller that controls a continuous environment, control of process and manufacturing plants, guidance of transport systems, robot planning, and many similar applications. Such an extension may enlarge the domain of systems that yield to formal and rigorous development approaches to include such important practical applications.

Recent Extensions to Real Time

An interesting step towards more realistic modeling and analysis of programs that interact with a continuous environments has been made by various extensions of the temporal framework to deal with *real time* [KKZ87], [RR87], [PH88], [Ost89], [AH89], [NRSV90], [HMP91]. Many other approaches are represented in this volume. It is interesting to note that, while some of these extensions are based on a dense model of time, the general structure of the model is still that of interleaving transitions, each of which causes an abrupt change of state. Some of the models even introduce a special time-passage (a *tick*) transition, which is the only transition causing the clock to progress.

This extension of the methodology allows representation of many additional phenomena in the world of programming [HMP91]. We can now take into consideration the fact that instructions take a nonzero time to complete, and represent the effect of *delay* commands, as well as the phenomena of task scheduling according to time and priorities, and so on.

However, this extension can be characterized as describing the interaction of a digital system with a *single* continuously varying factor — the real time clock. When we have several continuously varying parameters with more complex rules of change, the simple interleaving model is no longer satisfactory.

Extending the Framework

The discrete temporal framework as described, for example, in [MP89] includes the following components:

- An *underlying semantic model*. This captures the notion of a possible behavior of a system. In the case of discrete systems, this will be an infinite sequence of states or events that may occur in a possible run of the system.

- A generic *computational model*. This provides an effective representation of realizable systems. In the discrete case we take a *fair transition system* (a *timed transition system* [HMP91] for the real-time case) and show how concrete programming languages can be mapped onto this generic model. Transition systems provide an abstract representation of reactive programs and systems.

- A *specification language*. In the temporal framework we use temporal logic for specifying properties of reactive programs. In some cases we may use equivalent automata-based formalisms such as Statecharts [Har87] for specifying the detailed behavior of a system.

- A *verification methodology*, providing rules and axioms for formally proving that a proposed system meets its specification.

In the extension of this framework to hybrid systems we propose to extend (or replace) each of the discrete-framework components as follows:

- As an underlying semantic model we take *hybrid traces*, which are a mapping from extended continuous time to system states.

- As a generic computational model we take *phase transition systems* which represent the behavior of a system as a sequence of phases alternating between continuous and discrete changes. A continuous phase takes positive time and allows continuous change of variables governed, for example, by a set of differential equations. The discrete phase consists of a finite number of discrete transitions, each of which causes a (possibly) discontinuous change in the value of the variables.

- We consider two specification formalisms. For describing the detailed behavior of a system, we use an extension of Statecharts in which basic (unstructured) states may be labeled by a set of differential equations, used to describe a continuous change that occurs as long as the system is in that state. For describing properties and requirements of the system, we propose a modest extension of temporal logic, enabling it to refer to continuous change and to time.

- In this preliminary work on hybrid systems, we present only a partial proposal for an appropriate verification methodology, consisting of a rule for safety properties, and leave a more thorough investigation of the subject to subsequent research.

Related Work

The interest in formal treatment of systems that interact with continuous environments is certainly not new. It received a new impetus by the extension of formalisms to deal with real time. Indeed, several papers consider the specification of such systems, some of which are [MSB91], [CHR92], and [HRR91]. While all of these works recommend extensions to the specification language, they do not propose changes to the basic underlying semantic model, considering instead a discrete sequence of points which correspond to the points at which the discrete system samples the continuous environment.

To the best of our knowledge, the paper which comes closest to our semantic model is [San89], which proposes a *piecewise smooth* modeling of physical systems for the purpose of qualitative reasoning about physics.

Our interest in hybrid systems was triggered by a presentation of Fred Schneider at the workshop on real-time and fault tolerant systems held in Warwick in 1988. His approach to the subject is presented in [MSB91]. A closely related study of timed and hybrid systems is presented in [NSY91].

An important motivation for developing this theory came from applications to robotics and to process control and embedded systems.

2 Timed Systems

To deal with reactive systems whose behavior may depend on timing considerations, we present the discrete framework of *timed behaviors*. The notions of *timed trace* and *timed* *transition system* are taken with some small changes from [HMP91], while the logic MTL (which extends temporal logic by adding time-bounded temporal operators) is taken from [AH90] and is based on [KVdR83].

To model metric time, we assume a totally ordered time domain $\mathbf{T}$ which contains a zero element $0 \in \mathbf{T}$, and a commutative, associative operation $+$, for which 0 is a unique identity element, and such that for every $t_1, t_2 \in \mathbf{T}$,

> $t_1 < t_2$ iff there exists a unique $t \neq 0$ (denoted by $t_2 - t_1$), such that $t_1 + t = t_2$.

We refer to the elements of $\mathbf{T}$ as *time elements* or sometimes as *moments*. In most cases we will take $\mathbf{T}$ to be either the natural numbers N, or the nonnegative real numbers $\mathsf{R}_{\geq 0}$. We extend the domain $\mathbf{T}$ to $\mathbf{T}^\infty = \mathbf{T} \cup \{\infty\}$, where it is assumed that $\infty \geq t$ for all $t \in \mathbf{T}^\infty$.

With a system to be analyzed we associate

- V : A finite set of *state variables*.
- Σ : A set of *states*. Each state $s \in \Sigma$ is an interpretation of V; that is, it assigns to every variable $x \in V$ a value $s[x]$ in its domain.
- $V_T = V \cup \{T\}$: A finite set of *situation variables*. They are obtained by augmenting V, the set of state variables, with a variable T representing the current time in each situation.
- Σ_T : A set of situations. Every situation $s \in \Sigma_T$ is an interpretation of V_T. In particular, $s[T] \in \mathbf{T}$ is the value of the real-time clock at situation s. We denote by $s[V]$ the *state* corresponding to the situation s. It is obtained by restricting the interpretation s to the state variables V.

Timed Traces

A *progressive time sequence* is an infinite sequence of time elements

$$\theta : \; t_0, t_1, \ldots,$$

where $t_i \in \mathbf{T}$, for each $i = 0, 1, \ldots$, satisfying

- $t_0 = 0$.
- Time does not decrease. That is, for every $i \geq 0$, $t_i \leq t_{i+1}$.
- Time eventually increases beyond any bound. That is, for every time element $t \in \mathbf{T}$, $t_i > t$ for some $i \geq 0$. This is called the *Non-Zeno* requirement in [AL91].

A *timed trace* describing a potential behavior of a timed reactive system is an infinite sequence of situations

$$\sigma : \; s_0, s_1, \ldots,$$

where $s_i \in \Sigma_T$, for each $i = 0, 1, \ldots$. We denote by $t_i = s_i[T]$ the moment at which situation s_i was observed (sampled).

It is required that

- The sequence $t_0, t_1, \ldots$ is a progressive time sequence.
- For every $i \geq 0$, state and time do not change at the same time, i.e., either $s_i[V] = s_{i+1}[V]$ or $t_i = t_{i+1}$. This requirement ensures that each state change is precisely timed.

To illustrate the motivation for the last requirement, assume for a moment that we allow state and time to change in a single step. Then, the following would be an admissible trace

$$\langle x : 0\,,\, T : 0\rangle\ ,\ \langle x : 1\,,\, T : 1\rangle \ldots$$

This trace tells us that x was observed to equal 0 at time 0 while it was observed to equal 1 at time 1. It does not provide answers to the questions of when precisely did x change from 0 to 1 and how long it had been 0 before changing. The second requirement above forces us to choose between several possibilities, such as

$$\langle x : 0\,,\, T : 0\rangle\ ,\ \langle x : 1\,,\, T : 0\rangle\ ,\ \langle x : 1\,,\, T : 1\rangle\ ,\ \ldots,\ \text{or}$$
$$\langle x : 0\,,\, T : 0\rangle\ ,\ \langle x : 0\,,\, T : 1\rangle\ ,\ \langle x : 1\,,\, T : 1\rangle\ ,\ \ldots.$$

Or perhaps even

$$\langle x : 0\,,\, T : 0\rangle\ ,\ \langle x : 0\,,\, T : 0.5\rangle\ ,\ \langle x : 1\,,\, T : 0.5\rangle\ ,\ \langle x : 1\,,\, T : 1\rangle\ ,\ \ldots.$$

Timed Transition Systems

A *timed transition system* $S = \langle V, \Theta, \mathcal{T}, l, u\rangle$ consists of the following components:

- State variables V. We denote by Σ the set of all interpretations of V.
- The *initial condition* Θ. A satisfiable assertion that characterizes the states that can appear as initial states in a computation.
- A finite set $\mathcal{T}$ of *transitions*. Every 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-successors $\tau(s) \subseteq \Sigma$.

  A transition τ is *enabled* on s iff $\tau(s) \neq \phi$. Otherwise τ is *disabled* on s.
- A *minimal delay* $l_\tau \in \mathbf{T}$ for every transition $\tau \in \mathcal{T}$.
- A *maximal delay* $u_\tau \in \mathbf{T}^\infty$ for every transition $\tau \in \mathcal{T}$. We require that $u_\tau \geq l_\tau$ for all $\tau \in \mathcal{T}$.

Given the state variables V, we can obtain the corresponding set of situation variables $V_T = V \cup \{T\}$, and the set of situations Σ_T interpreting V_T.

With each transition $\tau \in \mathcal{T}$ we associate an assertion $\rho_\tau(V, V')$, called the *transition relation*, which refers to both an unprimed and a primed version of the state variables. The purpose of the transition relation ρ_τ is to express the relation holding between a state s and its τ-successor $s' \in \tau(s)$. We use the unprimed version of variables to refer to values in s, and the primed version to refer to values 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.

A *computation* of a timed transition system $S = \langle V, \Theta, \mathcal{T}, l, u\rangle$ is a timed trace

$$\sigma : \; s_0, s_1, \ldots,$$

where $s_i \in \Sigma_T$ for each $i = 0, 1, \ldots,$ which satisfies the following requirements:

- [*Initiality*] $s_0 \models \Theta$.
- [*Consecution*] For all $i \geq 0$,
  - Either $t_i = t_{i+1}$ and there is a transition $\tau \in \mathcal{T}$ such that $s_{i+1}[V] \in \tau(s_i[V])$, or
  - $s_i[V] = s_{i+1}[V]$ and $t_i < t_{i+1}$.

  In the first case, we say that τ is *taken* at position i. In the second case, we say that the clock has progressed at position i. Sometimes we refer to the second case by saying that a "tick" step has been taken at position i.
- [*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 $s_i[V], s_{i+1}[V], \ldots, s_j[V]$ and not taken at any of the positions $i, i+1, \ldots, j-1$. 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.

As shown in [HMP91], the model of timed transition systems is expressive enough to express most of the features specific to real-time programs such as delays, timeouts, preemption, interrupts and multi-programming scheduling.

Example

Consider the simple timed transitions system given by:

- State Variables $V : \{x, y\}$.
- Initial Condition: $\Theta : (x = 0) \wedge (y = 0)$.
- Transitions: $\mathcal{T} : \{\tau_1, \tau_2\}$ where

τ	ρ_τ	l_τ	u_τ
τ_1	$(y = 0) \wedge (x' = x + 1)$	1	2
τ_2	$(y = 0) \wedge (y' = 1)$	3	3

We present two computations of this timed transitions system. The first computation σ_1 attempts to let x reach its maximal possible value. Therefore, we try to activate τ_1 always at the first possible position and τ_2, which causes both transitions to become disabled, as late as possible.

$$\begin{array}{ll}\sigma_1: & \langle x:0,\, y:0,\, T:0\rangle \xrightarrow{tick} \langle x:0,\, y:0,\, T:1\rangle \xrightarrow{\tau_1} \langle x:1,\, y:0,\, T:1\rangle \xrightarrow{tick} \\ & \langle x:1,\, y:0,\, T:2\rangle \xrightarrow{\tau_1} \langle x:2,\, y:0,\, T:2\rangle \xrightarrow{tick} \langle x:2,\, y:0,\, T:3\rangle \xrightarrow{\tau_1} \\ & \langle x:3,\, y:0,\, T:3\rangle \xrightarrow{\tau_2} \langle x:3,\, y:1,\, T:3\rangle \xrightarrow{tick} \cdots\end{array}$$

The second computation σ_2 attempts to keep the value of x as low as possible. Consequently, it delays the activation of τ_1 to the latest possible position and tries to activate τ_1 at the earliest possible position.

$$\begin{array}{ll}\sigma_2: & \langle x:0,\, y:0,\, T:0\rangle \xrightarrow{tick} \langle x:0,\, y:0,\, T:2\rangle \xrightarrow{\tau_1} \langle x:1,\, y:0,\, T:2\rangle \xrightarrow{tick} \\ & \langle x:1,\, y:0,\, T:3\rangle \xrightarrow{\tau_2} \langle x:1,\, y:1,\, T:3\rangle \xrightarrow{tick} \cdots\end{array}$$ ▟

There are several observations that can be made concerning the computational model of timed transitions systems.

- Computations alternate between *tick* steps that advance the clock and sequences of state-changing transitions that take zero time.
- Transitions *wait* together but *execute* separately in an interleaving manner.

Define $wait(\tau, j)$ to be the largest t such that for some $i \le j$

- $t = t_j - t_i$,
- τ is enabled on all of $s_i[V], \ldots, s_j[V]$, and
- τ is not taken at any of $i, \ldots, j-1$.

The function $wait(\tau, j)$ measures the length of time that τ has been continuously enabled but not taken up to position j (assuming it is enabled at j).

We say that

τ is *ready* at position j if $wait(\tau, j) \ge l_\tau$, and

τ is *ripe* at position j if $wait(\tau, j) = u_\tau$.

We observe that, in a computation of a timed transition system,

- Time can progress only after all ripe transitions are taken or become disabled.
- When time progresses, it can jump forward only by an amount on which all the enabled transitions agree. That is, it must be such that it will not cause any enabled transition to become "over-ripe."

Not every timed transition system is guaranteed to have computations that satisfy all the requirements given above. For example, a TTS with a single transition τ whose transition relation is $\rho_\tau : (x' = x + 1)$ with lower and upper bounds given by $l_\tau = u_\tau = 0$ does not have a computation. This is because τ is always ripe and does not ever allow time to progress.

Let $\mathcal{T}_0$ be the subset of transitions whose upper bound is 0. A TTS is called *progressive* if there does not exists an infinite sequence of states

$$s_0, s_1, \ldots,$$

such that for every $i = 0, 1, \ldots,$ there exists a $\tau \in \mathcal{T}_0$, such that $s_{i+1} \in \tau(s_i)$. It is not difficult to see that every progressive TTS has at least one computation.

From now on, we restrict our attention to progressive transition systems.

Transitions that belong to $\mathcal{T}_0$ are called *immediate*. Transitions that have a positive upper bound are called *nonimmediate*. The set of all nonimmediate transitions is denoted by $\mathcal{T}_>$.

Specification 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. 1 a timed specification of two manufacturing machines which communicate by a conveyer that holds only one item at a time. The conveyer is a mechanical device that travels back and forth between the two machines.

The specification consists of three automata: M_1, M_2, and *Conveyer*, which operate concurrently. These components may represent a first machine that does the initial processing of a part, a second machine which applies more advanced processing to the part, and the conveyer device which moves the part from machine M_1 to machine M_2.

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

$$name : e/g!,$$

where *name* is an optional name of the transition (with no semantic meaning), e is a *triggering event* which causes the transition to become enabled, and $g!$ is an optional action which generates the event g when the transition is taken. When the transition has no triggering event, such as transition *good-part* in the diagram, the transition is enabled whenever the state from which it departs (state *Busy* in the diagram) is active.

In addition, each transition is optionally labeled by a pair of real numbers $[l, u]$, which specify the minimal and maximal delays of the transition. Transitions which are not explicitly labeled are considered to be immediate, i.e., to have the time bounds $[0, 0]$. We require that all transitions which have a triggering event be immediate. A transition is called *relevant* if the state from which it departs is currently active. Non-immediate transitions (which have no triggering event) are enabled whenever they are relevant. A transition with a triggering event e is enabled if it is relevant and the event e has just occurred, meaning that time has not progressed since the event was generated. More elaborate trigger conditions such as $put \wedge \neg first$ are allowed. Such a condition is true if

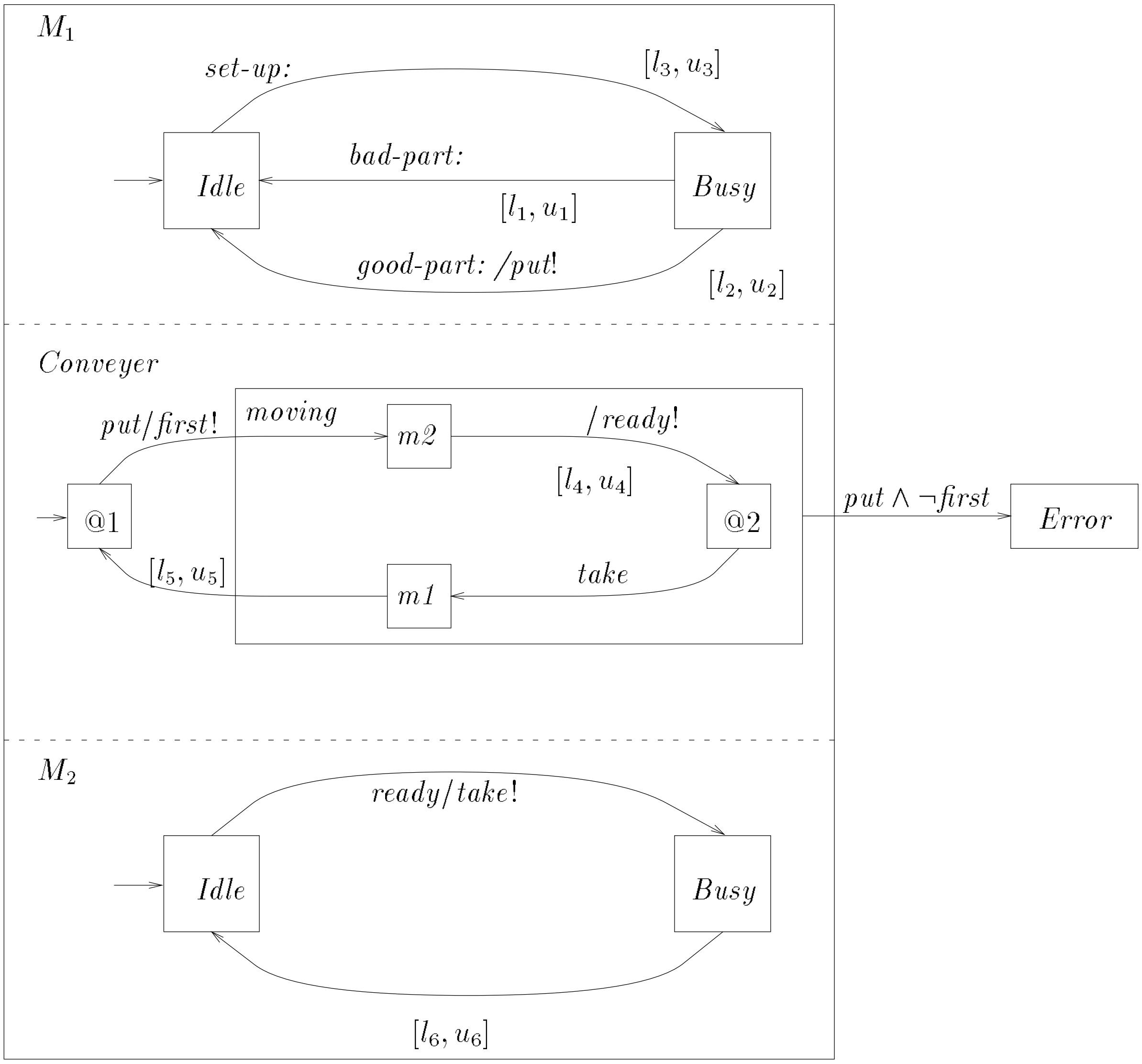


Figure 1: Specification of Two Machines with a Conveyer.

the event *put* has been generated since time has last progressed but the event *first* has not been generated since then.

Some states are *compound*. For example, the state encompassing basic states *m2*, @2, and *m1* in the automaton *Conveyer* is compound. It is considered active whenever one of the basic states it contains is active. A transition departing from a compound state (such as the transition leading into *Error*) is relevant whenever the compound state is active and, when taken, it makes the compound state and all of its substates inactive and activates the state to which the transition leads (*Error* in the example above).

When a transition that generates an event e is taken, one or more immediate transitions that have e as a triggering event and are currently relevant can be taken. This is the mechanism by which the concurrent automata synchronize. For example, if M_1 takes transition *good-part* while *Conveyer* is in state @1, then the transition labeled by *put* can be taken next. Note that, since this transition is immediate and ready (therefore also ripe), it must be taken before time can progress.

Consider for example the transition connecting state *m2* to state @2. Assume that it is taken at position j while, at the same time, machine M_2 is in state *Idle*. On being taken, this transition generates event *ready*. Machine M_2 responds to this generation by taking the transition leading into state *Busy*, generating the event *take*. Since at this point *Conveyer* is at state @2, it can respond immediately by moving to state *m1*. These three transitions must be taken before the clock advances.

Another important element of the behavior of Statecharts is that events that are generated at a certain step persist until time progresses. This allows more than one transition triggered by an event e to respond to e before time progresses.

The given specification describes the following possible scenarios. Both machines start at an idle state. After some time ranging between l_3 and u_3, machine M_1 concludes its set-up procedure and moves to the busy state. While being busy, M_1 may either take time between l_1 and u_1 to produce a bad part, or take time between l_2 and u_2 to produce a good part. In both cases it moves to state *Idle* where it completes another set-up procedure. If a good part is produced, the event *put* is generated, which triggers the transition departing from state @1 in *Conveyer* if it is relevant. This transition represents the initiation of movement of the conveyer between M_1 and M_2. The movement itself may take time between l_4 and u_4 to reach M_2. Reaching M_2 is represented by the transition connecting to state @2, which also generates the event *ready*. This event is sensed by M_2, which removes the part from the conveyer and starts processing it in its *Busy* state.

If all timings are right, M_1 should never issue the *put* signal when the conveyer is not at state @1. The diagram represents this expectation by having a transition that moves to state *Error* in all other cases.

An interesting analysis question is what the relation between the various time constants should be to ensure that this never happens. Various verification tools, based on algorithms similar to the one proposed in [ACD90], can answer such questions algorithmically.

It is important to note that the semantics of Statecharts presented here is not the standard semantics considered, for example, in [PS91]. The Statecharts presented here are *timed statecharts*, and their behavior is somewhat different than that given by the standard semantics. One of the main differences is that the notions of macro- and micro-steps no longer play such an important role in the semantics. Instead, there is a sequence of steps that can be taken before time progresses. By definition, any signal that is generated at a given time persists as long as time does not progress.

This explains the need for the signal *first*, which is emitted on entry to state *moving*, and whose negation labels the exit to *Error*. The system may enter state *moving* only when it senses the signal *put*. After the transition entering state *moving* is taken, signal *put* is still available. If the transition exiting to *Error* were labeled only by *put*, it would have always been taken following the entry to *moving*. Signal *first*, which is generated by the transition entering *moving*, prevents the error transition from being taken immediately. When time progresses, both *put* and *first* disappear, and the error transition will respond only to a new *put* signal generated at a later time.

For a more detailed description of timed statecharts, the reader is referred to [KP92].

3 Specification and Verification of Timed Transition Systems

To specify properties of timed systems, we use the language of temporal logic with some extensions.

Extensions to the Temporal Language

We assume familiarity with temporal logic and its operators as presented, for example, in [MP91]. The following extensions are introduced.

References to the Next Value

When evaluating a formula at position j of a computation σ, it is often necessary to evaluate terms that appear in atomic formulas. In the standard definition this evaluation is based on the value of the variables in the situation s_j. For a term r, we introduce the notation

$$\bigcirc r,$$

referring to *the next value* of r.

The precise definition uses the notation $val(\sigma, j, r)$ which defines the value of the term r at position j of computation σ.

- For a constant c, $val(\sigma, j, c) = c$, where we use the same notation for a value and its syntactic representation.
- For a variable $x \in V_t$, $val(\sigma, j, x) = s_j[x]$, i.e., the interpretation of x in the situation s_j.
- For a term r, $val(\sigma, j, \bigcirc r) = val(\sigma, j+1, r)$.
- For a function f and terms $r_1, \ldots, r_k$,
$$val(\sigma, j, f(r_1, \ldots, r_k)) \quad = \quad f(val(\sigma, j, r_1), \ldots, val(\sigma, j, r_k))$$
- For a predicate p and terms $r_1, \ldots, r_k$,
$$(\sigma, j) \models p(r_1, \ldots, r_k) \quad \text{iff} \quad p(val(\sigma, j, r_1), \ldots, val(\sigma, j, r_k)) = \mathrm{T}.$$

In [MP91], $\bigcirc r$ is denoted as r^+.

Age of a Formula

To refer to the passage of time, we introduce a temporal function that expresses the age of a formula.

For a formula φ, we introduce the term

$$\Gamma(\varphi),$$

called the *age* of φ. Its intended meaning is that the value of $\Gamma(\varphi)$ at a φ-position j records the time length of the largest interval ending at j in which φ holds continuously. If φ does not hold at j then $\Gamma(\varphi) = 0$ at j. Thus, we define

- $val(\sigma, j, \Gamma(\varphi))$ = The largest t such that, for some $i \leq j$, $t = t_j - t_i$ and φ holds at all positions $i, \ldots, j$. It is taken to be 0 if φ does not hold at j.

Age functions can be viewed as an alternative and generalization of the delay counters δ_τ introduced in [HMP91], which measure the length of time a transition τ has been continuously enabled and not taken.

A closely related concept is used in [SBM91] to allow assertional reasoning about real-time.

A notion similar to age functions is that of *duration*, proposed in [CHR92] to express properties of continuous systems. One difference between the two notions, is that durations are introduced in the context of interval temporal logic while we use point-based temporal logic. Another difference is that durations measure the *accumulated* time in which a formula was true within an interval, while the age of a formula measures the length of the largest time interval ending in the current position in which the formula held continuously.

Interval-Bounded Operators

Following [HMP91], we introduce for each temporal operator (excluding $\bigcirc$, $\circleddash$, and the weak previous operator $\widetilde{\circleddash}$) a bounded version of the operator obtained by subscripting the operator by an interval specification I. An interval specification may have one of the forms

$$[l, u] \quad [l, u) \quad (l, u] \quad (l, u).$$

In the first form, it is required that $l \leq u$, while in the others $l < u$. The semantic meaning of these *bounded operators* is straightforward. For example, $p\mathcal{U}_{(l,u]}q$ holds at position i of a timed trace $\sigma : (s_0, t_0), (s_1, t_1), \ldots$, iff there exists a j, $i \leq j$, such that $t_i + l < t_j \leq t_i + u$, q holds at j, and for all k, $i \leq k < j$, p holds at k.

We often use abbreviations such as $\square_{<u}$ and $\diamondsuit_{\leq u}$ to stand for $\square_{[0,u)}$ and $\diamondsuit_{[0,u]}$.

Two Styles of Specification and Verification

We refer to the logic obtained by the described extensions as MTL, standing for *metric temporal logic*. This logic can be used to state both timed and untimed properties of programs or even of detailed specifications. As an example, we will state three properties of the buffer system of Fig. 1. In the formulas describing these properties we use the name of a state as a proposition holding precisely when the state is active.

The first property simply states that the system will never reach state *Error*. This is stated by the untimed formula

$$\varphi_1 \quad : \quad \square \neg Error.$$

In order to verify this property, we may want to prove some lemmas that guarantee that state *Error* will never be reached. Obviously, state *Error* can be reached if signal *put* is issued while *Conveyer* is in state *moving*. The following liveness-like property states that *Conveyer* cannot stay in *moving* too long; namely, at most $u_4 + u_5$ time units. This requirement can be expressed by the formula

$$\varphi_2 \quad : \quad moving \Rightarrow \diamondsuit_{\leq u_4 + u_5} \neg moving.$$

Note that this formula is not always valid, since it depends on the fact that when *Conveyer* generates the signal *ready*, M_2 is in state *Idle* ready to respond to it.

As a last property, consider the formula

$$\varphi_3 \quad : \quad put \Rightarrow \Box_{(0,u_4+u_5]} \neg put.$$

This formula states that any two consecutive *put* signals are separated by a time distance greater than $u_4 + u_5$. This property, together with property φ_2, guarantees the validity of property φ_1.

The Age-based Approach

As explained in [HMP91] and previously in [PH88], there are two different approaches to the specification and verification of timed systems. The *bounded operators* approach expresses the dependency of properties on real time only through the bounded operators introduced above. The preceding formulas illustrate specifications based on this approach, and [HMP91] presents several useful proof rules for establishing properties expressed in this style.

An alternative approach can be described as the *explicit clock* approach. It does not use any bounded operators, but allows instead explicit references to the clock variable T. A possible methodology for explicit clock specification and verification is presented in [HLP90]. The formulas considered there allow arbitrary references to the clock variable T within terms. A brief description of a the explicit clock approach is also included in [HMP91], based on a transformation of a timed transition system into a fair transition system in which the clock variable T is made into a normal state variable, and the *tick* transition made a normal transition equal to the others.

This transformation introduces a special delay variable for each transition, which measures the time the transition has been continuously enabled and not taken. These delay variables are updated by all transitions that may modify the enabling condition or the waiting time of the transition with which they are associated. They are also updated by the *tick* transition.

The bounded operators approach is adequately covered in [HMP91]. In this paper we develop further the explicit clock approach. There are some differences between the presentation of this approach here and its presentation in [HMP91]. The main differences are that instead of introducing explicit delay variables, we work with the temporal function $\Gamma(\varphi)$, and all references to time must be expressed by this function.

We illustrate first the specification style appropriate to this approach. Reconsider the three properties that have been previously specified using the bounded operators approach. Property φ_1 does not use bounded operators, so no changes are necessary. To express property φ_2, stating that *Conveyer* cannot stay in the *moving* state too long, we may use the formula

$$\psi_2 \quad : \quad \Box(\Gamma(moving) \leq u_4 + u_5).$$

This formula states that proposition *moving* cannot hold continuously more than $u_4 + u_5$ at a stretch.

Property φ_3 can be stated without using bounded operators by the formula

$$\psi_3 \quad : \quad \neg put \wedge \bigcirc put \Rightarrow (\Gamma(\neg put) > u_4 + u_5).$$

This formula states that if signal *put* is about to be generated, which means that in the present situation it is still off but in the next situation it will be turned on, then *put* must have been continuously off for a period exceeding $u_4 + u_5$.

Axiomatization of Timed Transitions Systems

There are several ways to construct a proof system that will support proofs of properties of timed transitions systems under the explicit clock approach. Some are based on the translation (or a priori representation) of timed systems into conventional transition systems. Such a translation is described in [HMP91]. A full description of such an approach is presented in [AL91]. See also [SBM91] for a non-temporal proof approach which treats the clock as another state variable.

A certain overhead is associated with the consideration of the clock variable as a regular state variable. As a first step, several auxiliary variables are introduced which measure how long each transition has been continuously enabled. These are called delay variables in the translation of [HMP91]. An alternative but equivalent approach defines instead "deadline" variables which predict when each transition should be next activated. The auxiliary variable associated with a transition τ can be modified by any transition that may cause τ to change its enabling condition. Consequently, each transition relation must be augmented by a clause for each delay variable it may affect.

Another necessary element is the introduction of an explicit *tick* transition that causes time to progress. This transition must update all the delay variables for the transitions that are currently enabled. Furthermore, in order to compute the permissible time step that can be taken, the *tick* transition must make sure that advancing the clock will not cause any transition to be enabled longer than is allowed by its upper bound.

The approach presented here attempts to avoid the introduction of new auxiliary variables. Instead it uses the temporal age function Γ to express the same type of constraints.

We will present a set of axioms that are intended to characterize the set of admissible computations for a given timed transition system, and which will serve as the basis for proving its properties.

Axioms for the Progress of Time

The first set of axioms ensures that the sequence of values $t_0, t_1, \ldots$ forms a progressive time sequence.

$$\begin{array}{rcl} T_{init} & : & T = 0 \\ T_{\geq} & : & \Box(\bigcirc T \geq T) \\ T_{\neg Z} & : & \forall n \Diamond (T \geq n) \\ T_X & : & (x \neq \bigcirc x) \Rightarrow (T = \bigcirc T) \end{array}$$

Axiom T_{init} states that the initial value of T is 0. Axiom $T_{\geq}$ states that time never decreases from one situation to the next. Axiom $T_{\neg Z}$ expresses the *Non-Zeno* requirement by stating that for any natural number n (assuming that the natural numbers are embedded within the time domain $\mathbf{T}$), T eventually grows beyond n. Axiom T_X states that changes of state and changes of time are exclusive. Namely, if some state variable x changes in the current step then the value of T is preserved.

Axioms for Age

This set of axioms deals with the age function Γ. They are stated for an arbitrary formula φ.

$$\begin{aligned}
G_{init} &: \Gamma(\varphi) = 0 \\
G_F &: \bigcirc\neg\varphi \Rightarrow (\bigcirc\Gamma(\varphi) = 0) \\
G_T &: \bigcirc\varphi \Rightarrow (\bigcirc\Gamma(\varphi) = \Gamma(\varphi) + (\bigcirc T - T))
\end{aligned}$$

Axiom G_{init} states that the initial value of all ages is 0. Axioms G_F and G_T describe how the next value of $\Gamma(\varphi)$ is determined. For the case that φ is false in the next situation, G_F states that the next value of $\Gamma(\varphi)$ will be 0. For the case that φ is true in the next situation, G_T states that the next value of $\Gamma(\varphi)$ will be its current value plus the time increment $\bigcirc T - T$, i.e., the amount by which T will increase between the current and the next situation.

Transitions and their Activation

The next set of axioms deals with the effect of the transitions in $\mathcal{T}$ on the computation. We assume that each transition $\tau \in \mathcal{T}$ is associated with a transition relation $\rho_\tau(V, V')$ which expresses the relation between the values of the state variables in the present state (represented by V) and their values in the next state (represented by V').

We define

$$\begin{aligned}
\mathit{enabled}(\tau) &: \exists V'.\rho_\tau(V, V') \\
\mathit{taken}(\tau) &: \rho_\tau(V, \bigcirc V) \\
\mathit{last\text{-}taken}(\tau) &: \circleddash\, \mathit{taken}(\tau) \\
\mathit{waiting}(\tau) &: \Gamma(\mathit{enabled}(\tau) \wedge \neg \mathit{last\text{-}taken}(\tau))
\end{aligned}$$

The formula *enabled*(τ) expresses the fact that transition τ is enabled at the current situation. Formula *taken*(τ) is true at a position j if the next situation s_{j+1} can be obtained by applying τ to s_j. Note that *taken*(τ) may hold at a certain position for more than one transition. Formula *last-taken*(τ) holds at position j if $j > 0$ and τ can account for the passage between s_{j-1} to s_j. The value of function *waiting*(τ) at position j is equal to the length of time that τ has been continuously enabled and not taken up to and including position j.

In most of the timed transition systems we consider, all the transitions are *self-disabling*. This means that for every state s and transition τ, $\tau(\tau(s)) = \phi$, i.e., τ cannot be applied twice in succession to any state because it becomes disabled after the first application. For this prevalent situation, *enabled*(τ) entails $\neg$*last-taken*(τ), and we can therefore take *waiting*(τ) to be simply $\Gamma(\mathit{enabled}(\tau))$.

The axioms dealing with the requirements of proper initiation and consecution of computations are

$$
\begin{aligned}
C_{init} &: \Theta \\
C_{cons} &: \Box\Big((\bigcirc V = V \wedge \bigcirc T > T) \vee \bigvee_{\tau \in \mathcal{T}} (waiting(\tau) \geq l_\tau \wedge taken(\tau)) \wedge (\bigcirc T = T)\Big) \\
C_{upper} &: \Box(waiting(\tau) \leq u_\tau) \qquad \text{for every } \tau \in \mathcal{T}
\end{aligned}
$$

Axiom C_{init} states that the initial condition Θ holds at position 0 of the computation. Axiom C_{cons} describes what can happen on each step of the computation. One possibility is that all state variables remain the same, which is described by the clause $\bigcirc V = V$, and time increases. Alternately, some transition τ which has been waiting for at least l_τ is taken, while time remains the same. Axiom C_{upper} ensures that all upper bound requirements are respected by requiring that no transition τ ever waits more than u_τ.

Derived Proof Rules

The axioms presented above are adequate for proving properties of timed transition systems expressed by temporal logic formulas that may use the age function $\Gamma(\psi)$ for assertions ψ.

However, for concrete proofs, it is often useful to first derive some auxiliary proof rules that encapsulate common modes of reasoning. Rule INVT is such a rule and is useful for establishing the validity of formulas of the form $\Box q$ where q (and φ appearing in the rule) is an assertion, possibly containing terms of the form $\Gamma(p)$ for assertions p.

$$
\begin{array}{lll}
\text{INVT} & \text{I1.} & \varphi \to q \\
 & \text{I2.} & \Theta \to \varphi \\
 & \text{I3.} & (\rho_\tau \wedge \Gamma(enabled(\tau)) \geq l_\tau \wedge T' = T \wedge \varphi) \to \varphi', \quad \text{for every } \tau \in \mathcal{T} \\
 & \text{I4.} & \left(\begin{array}{c} V' = V \wedge T' > T \wedge \varphi \\ \wedge \\ \bigwedge_{\tau \in \mathcal{T}} \Big(enabled(\tau) \;\to\; (\Gamma(enabled(\tau)) + T' - T) \leq u_\tau \Big) \end{array} \right) \to \varphi' \\
\hline
 & & \Box q
\end{array}
$$

The rule uses an auxiliary assertion φ which is stronger than q (i.e., implies q) and is shown to be invariant. Premise I1 states that φ implies q. Therefore, if φ is invariant over every computation, so is q. Premise I2 requires that the initial condition Θ implies φ. This establishes that φ holds at position 0 of every computation.

Premises I3 and I4 show that φ is preserved over every possible step in the computation. Premise I3 deals with a step that is caused by taking transition τ. The antecedent of the implication lists the conditions that are necessary for the current and next situation to be τ-related. It uses a primed version of the variables to refer to their values in the next situation and an unprimed version to refer to their values in the current situation. The clause $T' = T$ is derived from axiom T_X, which states that if variables change then time remains constant. The right hand side of I3 contains φ', the primed version of φ. It is obtained by replacing all variables not appearing within a Γ context by their primed version, and replacing the primed version of a Γ expression by

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

Assuming no nested Γ expressions, p' is obtained by priming all variables appearing within p.

Premise I4 deals with a step caused by the progress of time. Its antecedent contains the clause $V' = V$, requiring that the state variables retain their values when time progresses. The clause $T' > T$ represents the requirement that time progresses by a positive amount.

In proving the premises' implications, we may use freely primed and unprimed instantiations of the axioms. For example, for a self-disbaling transition τ, we may use the instantiations of C_{upper}

$$\Gamma(enabled(\tau)) \le u_\tau \qquad \text{and} \qquad \Gamma'(enabled(\tau)) \le u_\tau.$$

4 A Verification Example

In this section we demonstrate the style of proofs in the explicit clock approach.

Consider the program presented in Fig. 2.

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

$$P_1 :: \left[\begin{array}{ll} \ell_0: & \textbf{noncritical} \\ \ell_1: & \textbf{await } x = 0 \\ \ell_2: & x := 1 \\ \ell_3: & \textbf{skip} \\ \ell_4: & \textbf{await } x = 1 \\ \ell_5: & \textbf{critical} \end{array}\right] \quad \Big\| \quad P_2 :: \left[\begin{array}{ll} m_0: & \textbf{noncritical} \\ m_1: & \textbf{await } x = 0 \\ m_2: & x := 2 \\ m_3: & \textbf{skip} \\ m_4: & \textbf{await } x = 2 \\ m_5: & \textbf{critical} \end{array}\right]$$

Figure 2: Coordination by Timing.

This program has been suggested by Fred Schneider as a minimal yardstick for assessing the feasibility of proposed proof systems for real time. He attributes it to M. Fischer.

The Associated TTS

As a first step, we should identify the timed transition system associated with this program. For full details of the representation of programs as timed transition systems we refer the reader to [HMP91]. Here we will only provide the details necessary for the treatment of this program.

As state variables we take

$$V \;:\; \{x, \pi\}.$$

These consist of the data variable x and the control variable π ranging over sets of locations. A value $\pi = \{\ell_i, m_j\}$ implies that control of P_1 is currently at location ℓ_i and control of P_2 is currently at m_j.

The initial condition is given by the assertion

$$\Theta \quad : \quad (x = 0) \wedge (\pi = \{\ell_0, m_0\}),$$

requiring that the initial value of x is 0 and control starts at locations ℓ_0 and m_0.

There is a transition associated with each statement of the program. Let

$$\cdots \lambda : S; \;\; \mu : \cdots$$

represent any of the statements, where λ and μ stand for the labels appearing before and after the statement. For the case of the **critical** statements which appear last in the program, μ is taken to be empty. With each such statement we associate a transition τ_S, whose transition relation ρ_S is defined according to the type of the statement.

- For S being **noncritical**, **skip**, or **critical**,

  $$\rho_S \quad : \quad (\lambda \in \pi) \wedge (\pi' = \pi - \{\lambda\} \cup \{\mu\}).$$

  Thus, the enabling condition for these statements is that control is in front of the statement. When taken, control moves to the location following the statement. In this and other transition relations, we follow the convention that variables (such as x) whose primed versions do not appear in the formula are preserved by the transition. That is, for each such variable x the clause $x' = x$ is assumed.

- For S of the form $x := v$,

  $$\rho_S \quad : \quad (\lambda \in \pi) \wedge (\pi' = \pi - \{\lambda\} \cup \{\mu\}) \wedge (x' = v).$$

- For S of the form **await** $x = v$,

  $$\rho_S \quad : \quad (\lambda \in \pi) \wedge (x = v) \wedge (\pi' = \pi - \{\lambda\} \cup \{\mu\}).$$

  Thus, for these statements the enabling condition also includes the requirement $x = v$.

The lower and upper bounds associated with the transitions are as follows:

- For the transitions associated with statements **noncritical** and **critical** the bounds are $[0, \infty]$. This means that they may be taken immediately or at any time.

- For all other transitions we assume uniform bounds $[L, U]$, with $0 < L \leq U$.

The Specification and its Proof

Assuming that the time bounds satisfy $2 \cdot L > U$, we are asked to prove mutual exclusion, which can be stated by

$$\psi_0 \;:\; \Box\neg(at\_\ell_5 \wedge at\_m_5).$$

This formula states that there will never be a state in which P_1 is executing at ℓ_5 while P_2 is executing at m_5. The formula uses the control predicates at_ℓ_i and at_m_j which are abbreviations for $\ell_i \in \pi$ and $m_j \in \pi$, respectively.

To apply rule INVT, we identify q as $\neg(at\_\ell_5 \wedge at\_m_5)$. The rule calls for a construction of assertion φ which is stronger than q and is *inductive*, i.e., satisfies premises I2–I4. Usually, q as given is not inductive. Rather than present a complete inductive assertion, we prefer to share with the readers the process by which such an assertion is constructed.

The main heuristic is strengthening a given assertion by adding pre-conditions that must hold if the assertion is in fact invariant. For example, if q is not inductive, there must exist some transition τ such that q is not preserved under τ. Form the assertion

$$p_1 \;:\; \forall V'(\rho_\tau \rightarrow q').$$

Assertion p_1 characterizes precisely the requirement on situation s so that every τ-successor of s will satisfy q. It is also clear that p_1 is not implied by q, so $q \wedge p_1$ is stronger than q. This is because the validity of $q \rightarrow p_1$ is equivalent to the validity of $(\rho_\tau \wedge q) \rightarrow q'$, stating that q is preserved over τ.

Thus, our next candidate for an inductive assertion is $q_1 : q \wedge p_1$. We proceed to check whether q_1 is inductive, and if we find another transition that does not preserve q_1, this gives rise to an even stronger assertion, and so on. Hopefully this process will converge to identify an inductive assertion that implies q.

This incremental strengthening of q is often coupled with additional heuristics that attempt to simplify and generalize the precondition p_1.

In checking for the inductiveness of an assertion, we do not have to check all transitions in great detail. There are certainly transitions that can be discarded after a cursory syntactical examination. These are, for example, all the transitions that do not modify any variable on which the assertion depends. In general, we only should investigate transitions that look as though they may change the value of the assertion from false to true. We refer to such transitions as potentially falsifying or, sometimes, as potentially hazardous.

For assertions that have the form of an implication $p \rightarrow r$, it sufficient to consider transitions that may change p from false to true and those that may change r from true to false.

The two transitions which are potentially hazardous to the validity of q, which can also be written as the implication $at\_\ell_5 \rightarrow \neg at\_m_5$, are m_4 while P_1 is at ℓ_5, and ℓ_4 while P_2 is at m_5. We begin our analysis of the first case. The second case can be handled symmetrically.

The precondition that excludes taking m_4 while P_2 is at ℓ_5 is

$$at\_\ell_5 \wedge at\_m_4 \;\Rightarrow\; x \neq 2.$$

Indeed, if we believe q to be invariant, we must also believe (ignoring timing considerations for the moment) that the above formula is invariant. Otherwise, m_4 could be taken and lead to a violation of q.

With some generalization of this formula, we add to our assertion the requirement

$$\varphi_1 \quad : \quad at\_\ell_5 \;\Rightarrow\; x = 1$$

Checking the transitions that may endanger the validity of φ_1, we find m_2 which, when executed, will set x to 2. We therefore add the following requirement that guarantees that m_2 cannot be taken while P_1 is at ℓ_5.

$$\varphi_2 \quad : \quad at\_\ell_5 \;\Rightarrow\; \neg at\_m_2$$

To verify the validity of φ_2, we check all the transitions that may potentially threaten it.

- m_1 while at_ℓ_5.
  To prevent this from occurring, it is sufficient to show that

  $$at\_\ell_5 \wedge at\_m_1 \;\Rightarrow\; x \neq 0.$$

  We generalize this to the requirement

  $$\varphi_3 \quad : \quad at\_\ell_{3..5} \wedge at\_m_{0..2} \;\Rightarrow\; x = 1.$$

  By checking all the transitions, we find out that φ_3 is indeed inductive.

- ℓ_4 while at_m_2.
  An intuitive argument showing that this cannot happen is that by the time ℓ_4 is possible (ready), $x = 1$ must have held continuously for at least $2 \cdot L$ time units. However, P_2 cannot wait at m_2 that long without moving on (since $U < 2 \cdot L$).

Formalization of the Time-Dependent Reasoning

Up to this point, the part of the rule we have used and even the recommended heuristic are not influenced by timing considerations. It is only the formalization of the intuitive argument presented above that requires the stronger proof system that takes timing into account.

We start by proving several lemmas.

$$\psi_1 \quad : \quad \Box(\Gamma(at\_m_2) \leq U)$$

This invariant is an instance of axiom C_{upper} and the fact that m_2 is self-disabling.

The next lemma claims

$$\psi_2 \quad : \quad at\_m_2 \;\Rightarrow\; \Gamma(at\_m_2) \geq \Gamma(x = 1).$$

The potentially endangering transitions are

- m_1: Possible only if $x = 0$, which implies $\Gamma'(x = 1) = 0$.

- m_2: Making $at\_m'_2 = \text{F}$.

- *tick* (passage of time): Clearly if at_m_2 and $\Gamma(at\_m_2) \geq \Gamma(x=1)$ hold before the tick, then

$$\Gamma'(at\_m_2) \;=\; \Gamma(at\_m_2) + T' - T \;\geq\; \Gamma(x=1) + T' - T \;\geq\; \Gamma'(x=1).$$

A final lemma is

$$\psi_3 \;:\; at\_\ell_5 \wedge at\_m_{0..2} \Rightarrow \Gamma(x=1) \geq 2 \cdot L.$$

This is proven by proving separately

$$\begin{array}{l} at\_\ell_3 \wedge at\_m_{0..2} \Rightarrow \Gamma(x=1) \geq \Gamma(at\_\ell_3) \\ at\_\ell_4 \wedge at\_m_{0..2} \Rightarrow \Gamma(x=1) \geq L + \Gamma(at\_\ell_4) \end{array}$$

All these invariants can be proven directly by rule INVT taking φ to be q.

We may assemble now the three lemmas to obtain

$$at\_\ell_5 \wedge at\_m_2 \Rightarrow U \;\overset{\psi_1}{\geq}\; \Gamma(at\_m_2) \;\overset{\psi_2}{\geq}\; \Gamma(x=1) \;\overset{\psi_3}{\geq}\; 2 \cdot L$$

Since $2 \cdot L > U$, this shows that $at\_\ell_5 \wedge at\_m_2$ is impossible, leading to the validity of φ_2.

5 Hybrid Systems

As a first step in the development of semantics for hybrid systems we discuss the underlying *time model*. From now on, we assume that the time domain $\mathbf{T}$ is $\mathsf{R}_{\geq 0}$, the nonnegative real numbers.

Let $\theta : t_0, t_1, \ldots$ be a progressive time sequence. The *time structure* induced by θ is defined to be the set of pairs

$$\mathsf{T}_\theta \;:\; \{\langle i,t\rangle \mid \quad i = 0,1,\ldots, \quad t = t_i \;\vee\; t_i < t < t_{i+1}\}$$

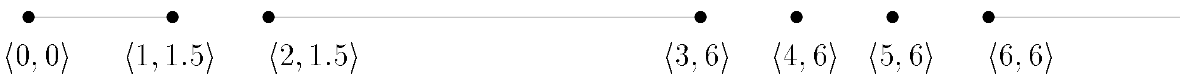
Thus, T_θ consists of all the pairs $\langle i, t_i\rangle$, $i = 0,1,\ldots$, corresponsing to the elements of θ, as well as the pairs $\langle i,t\rangle$ corresponding to intermediate points $t_i < t < t_{i+1}$.

We refer to the elements of T_θ as *θ-moments*, or simply as *moments* when θ is understood. For each moment $m = \langle i,t\rangle$ we write $time(m)$ for the value t, the time stamp of m. We refer to moments of the form $\langle i, t_i\rangle$ as the *discrete moments* of T_θ, and to moments of the form $\langle i,t\rangle$, where $t_i < t < t_{i+1}$ as the *continuous moments* of T_θ. The set T_θ is ordered by the lexicographic ordering

$$\langle i,t\rangle \prec \langle i',t'\rangle \quad \textit{iff} \quad i < i' \text{ or } (i = i' \text{ and } t < t').$$

We write $m_1 \preceq m_2$ for the case that either $m_1 \prec m_2$ or $m_1 = m_2$. Note that if $\langle i,t\rangle \preceq \langle i',t'\rangle$ then $i \leq i'$ and $t \leq t'$.

The following diagram represents a prefix of a time structure induced by the time sequence θ : 0, 1.5, 1.5, 6, 6, 6, 6, 7, 8, ...



In this diagram, we have only marked the discrete moments, but any two discrete moments with different time stamps are seperated by uncountably many continuous moments. For example, $\langle 2, 1.5\rangle$ and $\langle 3, 6\rangle$ are separated by all (continuous) moments of the form $\langle 2, t\rangle$ for $1.5 < t < 6$.

A time structure can be viewed as consisting of alternations between discrete and continuous phases. A *discrete phase* is a maximal subsequence $\langle i, t_i\rangle, \langle i+1, t_{i+1}\rangle, \ldots, \langle j, t_j\rangle$, for $i \leq j$, where $t_i = t_{i+1} = \cdots = t_j$. A *continuous phase* consists of a nonempty open interval of the form $O_i : \{\langle i, t\rangle \mid t_i < t < t_{i+1}\}$, for $t_i < t_{i+1}$. Sometimes we refer to the closed interval $C_i : \{\langle i, t_i\rangle\} \cup O_i \cup \{\langle i+1, t_{i+1}\rangle\}$ or to the half-open interval $H_i : \{\langle i, t\rangle \mid t_i \leq t < t_{i+1}\}$.

For any closed interval C_i, we denote by $K_i = [t_i, t_{i+1}]$ the set of time values associated with the moments of C_i. A closed interval C_i such that $t_i < t_{i+1}$ is called a *nontrivial* interval.

Hybrid Traces

The state variables of a hybrid system are partitioned into $V = V_c \cup V_d$, where

- V_c is the set of *continuous variables*. These variables are modified by continuous activities in the behavior of a hybrid system.
- V_d is the set of *discrete variables*. These variables are changed by discrete steps.

As before, we define Σ the set of states to consist of all interpretations of V, and Σ_T the set of situations to consist of all interpretations of $V_T = V \cup \{T\}$.

A *hybrid trace* is a pair (θ, σ), where

- θ is a progressive time sequence, and
- $\sigma : \mathsf{T}_\theta \mapsto \Sigma$ is a function assigning a state $\sigma(m) \in \Sigma$ to each θ-*moment* $m \in \mathsf{T}_\theta$. We extend σ to map moments into situations by taking $\sigma(\langle i, t\rangle)[T] = t$.

For ease of notation, we will write $\sigma(i, t)$ for $\sigma(\langle i, t\rangle)$.

Consider a nontrivial closed interval C_i. For each state variable $y \in V$, σ induces a function y_σ from $K_i = [t_i, t_{i+1}]$ to the domain of y, which is defined by

$$y_\sigma(t) = \begin{cases} \sigma(i, t)[y] & \text{for } t_i \leq t < t_{i+1} \\ \sigma(i+1, t_{i+1})[y] & \text{for } t = t_{i+1} \end{cases}$$

Thus, the value of y_σ at the left and right boundaries of K_i are taken to be the values of y at the delimiting discrete moments $\langle i, t_i\rangle$ and $\langle i+1, t_{i+1}\rangle$, respectively.

For each variable $y \in V$ and each nontrivial closed interval C_i, it is required that

- If y is a discrete variable then y_σ is a constant function over K_i. This means that discrete variables do not change over continuous phases.

- If y is a continuous variable, then y_σ is a continuous function over K_i. For the end points t_i and t_{i+1} it is only required that y_σ be continuous from the right and from the left, respectively.

Thus, if we consider the time domain depicted above, y_σ should have a right limit at 1.5 which equals the value of y at the state corresponding to the moment $\langle 2, 1.5 \rangle$.

A hybrid trace can be described as a continuous activity interspersed with countably many bursts of discrete activity which take zero time.

Phase Transition Systems

The generalization of a timed transition system to the hybrid domain is called a *phase transition system*. Phase transition systems allow an effective description of systems that can generate hybrid traces as previously described.

Before presenting the formal definition, we make the observation that changes in a phase transition system are governed by the dual constructs of *transitions* and *activities*. The table below compares some of the features of these two constructs.

	Transitions	*Activities*
Govern	Discrete Change	Continuous Change
Take	No Time	Positive Time
Execute	By Interleaving	In parallel
Defined by	Transition Relations	Differential Equations

Transitions and activities interact. Transitions start and stop activities and modify the parameters on which the behavior of activities depends. Activities may generate events and conditions that enable or trigger transitions. A typical scenario is that a transition is triggered by the event $becomes(x \geq 0)$, which occurs precisely at the moment in which x switches from a negative value to a non-negative one. An immediate transition that depends on this event for its activation will interrupt the continuous change and execute at this precise time point.

A phase transition system Φ consists of $\langle V, \Theta, \mathcal{T}, \mathcal{A}, l, u \rangle$, where

- $V = V_c \cup V_d$ is the set of state variables, partitioned into the continuous variables V_c and the discrete variables V_d.

- Θ is an assertion, characterizing the admissible *initial states*.

- $\mathcal{T}$ is a finite set of *transitions*, each transition $\tau \in \mathcal{T}$, mapping each state $s \in \Sigma$ into a set of successors $\tau(s) \subseteq \Sigma$. Each transition τ is associated with a transition relation ρ_τ which characterizes the relation between states and their τ-successors. Transitions are allowed to change the values of continuous variables.

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

$$a_\alpha \;\rightarrow\; \mathcal{E}_\alpha,$$

where a_α is a boolean expression over the discrete variables, called the *activation condition*, and $\mathcal{E}_\alpha$ is a differential equation of the form $\dot{y} = r$, where y is a continuous variable and r is a term over V. We say that the activity *constrains the variable* y. It is required that the activation conditions of different activities that constrain the same variable be exclusive.

An activity α is called *operational* in a state if a_α holds there.

- l is a *lower bound* assigning to each transition τ a minimal delay $l_\tau \in \mathsf{R}_{\geq 0}$.
- u is an *upper bound* assigning to each transition τ a maximal delay $u_\tau \in \mathsf{R}^{\infty}_{\geq 0}$. We require that $u_\tau \geq l_\tau$ for all $\tau \in \mathcal{T}$.

Hybrid Computations

A hybrid trace (θ, σ) is a computation of a phase transition system Φ if it satisfies the following requirements.

- [*Initiality*] $\sigma(0,0) \models \Theta$.
- [*Discrete Consecution*] For each $i \geq 0$ such that $t_i = t_{i+1}$, there exists a $\tau \in \mathcal{T}$, such that
$$\sigma(i+1, t_{i+1}) \in \tau(\sigma(i, t_i)).$$
We say that τ *is taken* at $\langle i, t_i \rangle$.
- [*Lower bound*] If τ is taken at $\langle j, t' \rangle$, there exists a moment $\langle i, t \rangle \preceq \langle j, t' \rangle$ such that $t + l_\tau \leq t'$ and τ is enabled on $\sigma(m)$ for all m, $\langle i, t \rangle \preceq m \preceq \langle j, t' \rangle$ and not taken at any m, $\langle i, t \rangle \preceq m \prec \langle j, t' \rangle$.
- [*Upper bound*] If τ is enabled at $\langle i, t \rangle$, there exists a moment $\langle j, t' \rangle \succeq \langle i, t \rangle$ such that $t + u_\tau \geq t'$ and either τ is not enabled at $\langle j, t' \rangle$, or τ is taken at $\langle j, t' \rangle$.
- [*Continuous change*] For every nontrivial closed interval C_i, the functions $y_\sigma(t)$ for each $y \in V_c$ are continuous functions that satisfy *all* the activities $\alpha \in \mathcal{A}$. Note that if a_α is false at a state then activity α is satisfied by any functions. For each discrete variable $y \in V_d$, the function $y_\sigma(t)$ is constant in the interval, implying that discrete variables retain their value throughout the interval. Note that, since activation conditions only depend on discrete variables, an activity is operational at one point in the interval iff it is operational at all points of the interval.

  By default, the function $y_\sigma(t)$, for a continuous variable $y \in V_c$ which is not constrained by any activity that is operational in the interval, is also constant.

Let $\mathcal{T}_0$ denote the set of all *immediate* transitions, i.e., transitions whose upper bound is 0, and $\mathcal{T}_>$ denote the set of transitions whose upper bound is positive. To simplify matters we require that all transitions whose enabling condition depends on continuous variables be immediate. Let $\tau \in \mathcal{T}_0$ be an immediate transition. A careful examination of the *upper bound* requirement shows that τ cannot be enabled at a moment that belongs to a half-open continuous interval, i.e., at a moment $\langle i, t \rangle$, such that $t_i \leq t < t_j$. This is because the only moment $\langle j, t' \rangle \succeq \langle i, t \rangle$ such that $t + u_\tau = t'$ is $\langle j, t' \rangle = \langle i, t \rangle$ ($u_\tau = 0$) and τ is neither disabled nor taken there.

Example

Consider a phase transition system Φ_1 defined as follows:

The state variables consist of a continuous variable x ranging over the reals, and a discrete variable y ranging over the naturals.

The initial condition is

$$\Theta \ : \ (x = 0) \wedge (y = 0)$$

There is a single transition τ, with transition relation

$$\rho \ : \ (x = 1) \wedge (y = 0) \wedge (y' = 1)$$

The time bounds for τ are $[0, 0]$.
There is a single activity α given by

$$\alpha \ : \ y = 0 \ \rightarrow \ \dot{x} = 1.$$

The computations of this phase transition system are all of the form

$$\langle x:0,\, y:0,\, i:0,\, T:0\rangle,\ \ \{\langle x:t,\, y:0,\, i:0,\, T:t\rangle \mid 0 < t < 1\},\ \ \langle x:1,\, y:0,\, i:1,\, T:1\rangle,$$
$$\langle x:1,\, y:1,\, i:2,\, T:1\rangle,\ \ \{\langle x:1,\, y:1,\, i:2,\, T:t\rangle \mid 1 < t < t_3\},\ \ \langle x:1,\, y:1,\, i:3,\, T:t_3\rangle,$$
$$\{\langle x:1,\, y:1,\, i:3,\, T:t\rangle \mid t_3 < t < t_4\}, \cdots$$

for a progressive time sequence $0 < 1 \leq 1 < t_3 < t_4 < \cdots$. The presentation of these computations lists, for each moment $\langle i, t \rangle$, a tuple consisting of $\sigma(i,t)[x]$, $\sigma(i,t)[y]$, i, and t.

All of these computations have a continuous phase in the time interval $[0, 1]$ in which activity α is operational and causes x to rise from 0 to 1 continuously. The phase stops at $t = 1$ because the immediate transition τ becomes enabled. This transition is taken at moment $\langle 1, 1 \rangle$, leading to the state $\sigma(2, 1)$ in which $y = 1$. Beyond this moment neither α nor τ can be active, and the only thing that happens is that time progresses. The progress of time is described by an alternation of discrete states and continuous intervals in which all state variables remain constant and the only changing parameter is time itself. ▪

Not every phase transition system has computations. For example, if we replace the transition relation in the preceding example by the relation

$$\rho \ : \ (x > 1) \wedge (y = 0) \wedge (y' = 1),$$

then the resulting phase transition system has no computations. The reason is that while x is increasing uniformly, there is no definite value of T in which the predicate $x > 1$ precisely becomes true. Thus, we cannot let the continuous phase extend beyond 1. On the other hand, if we stop it at 1 then $x = 1$, and τ is not enabled yet.

Consequently, consider an assertion $\varphi(x)$ which depends on a continuous variable x. We say that φ is *sharp* in x if for every t_1 and t_2 and every function $f(t)$ continuous for $t \in [t_1, t_2]$ such that $\varphi(f(t_1))$ is false and $\varphi(f(t_2))$ is true, there exists a t, $t_1 < t \leq t_2$ such that $\varphi(f(t))$ is true and for every t', $t_1 \leq t' < t$, $\varphi(f(t'))$ is false. This guarantees a definite point at which φ changes from false to true. If all the enabling conditions of every transition are sharp then the problem encountered above is not possible. For the case that φ depends on two or more continuous variables, e.g., on $x_1, x_2, \ldots$, we require that a similar condition holds for every list of continuous functions $f_1(t), f_2(t), \ldots$.

This is not the only obstacle to having a computation. There are other cases when we can obtain a pair (θ, σ) which satisfy all the requirements of a computation except that the elements of θ are bounded by some integer N. This violates the requirement of *Non-Zeno*. Additional conditions may be required to avoid this situation.

An Example of a Hybrid Specification

To give a better picture of how phase transition systems operate, we provide an example of a specification of a hybrid system. The first presentation of the specification uses a Statechart augmented with a notation that annotates some basic states by a set of differential equations. The implied meaning is that, whenever the state is active, the associated differential equations are operational. Thus, activities are associated with annotated *states* of the Statechart while transitions are associated with the *arrows* connecting the states.

The example can be described as follows: 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. 3 describes the possible scenarios.

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.

A behavior of the system starts by setting the distance variables x_m and x_c to their initial value X_0. Then each of the players begins its local behavior. The mouse proceeds immediately to the state of running, in which his variable x_m changes continuously according to the equation $\dot{x}_m = -V_m$. The cat waits for a delay of Δ before entering its running state. Then there are several possible scenarios. If the event $x_m = 0$ happens first, the mouse reaches sanctuary and moves to state *safe*, where it waits for the cat to reach the wall. As soon as this happens, detectable by the condition $x_c = 0$ becoming true, the system moves to state *Mouse-Wins*. The other possibility is that the event $x_c = x_m > 0$ occurs first, which means that the cat overtook the mouse before the mouse reached sanctuary. In this case they both move to state *Cat-Wins*.

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

Note that the condition $x_c = x_m > 0$ is not sharp according to our definition. For the current system it is obvious (under the assumption that $\Delta > 0$) that if the condition changes from false to true it always happens at a definite moment. Consequently, in the presented specification, we should not be concerned with the fact that this condition is not sharp. However, an alternative sharp condition that can replace $x_c = x_m > 0$ is

$$(x_c = x_m) \wedge Mouse.running.$$

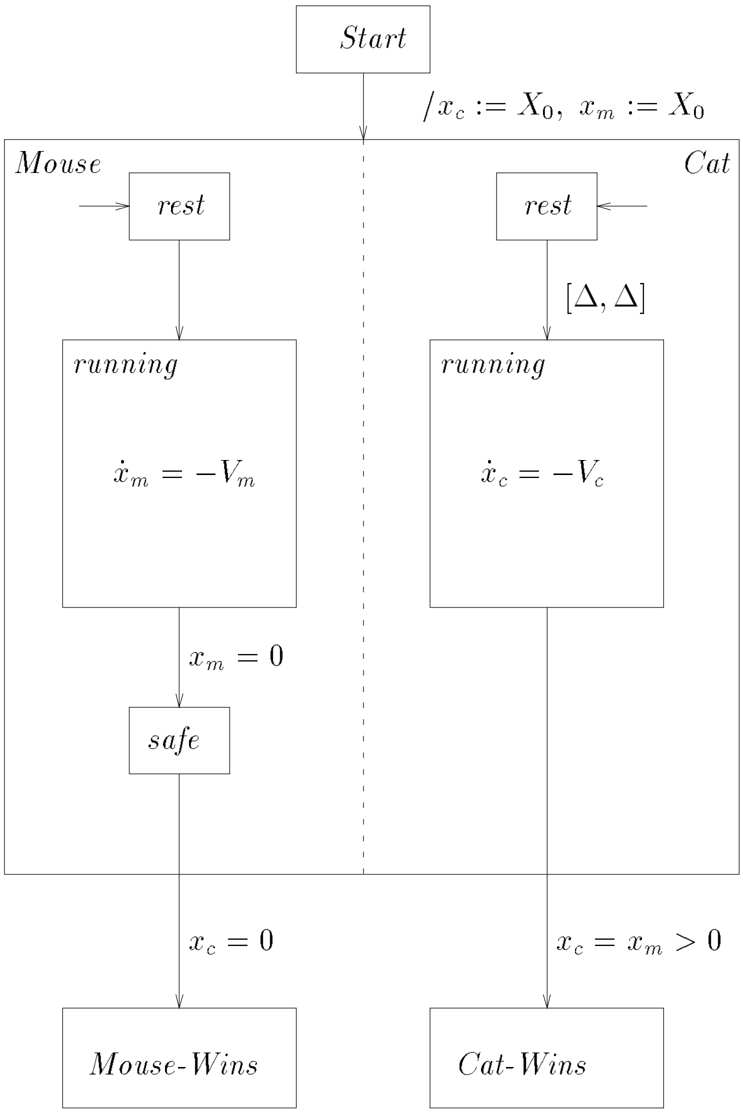


Figure 3: Specification of Cat and Mouse.

The idea of using Statecharts with continuous activities associated with certain states (usually basic ones) was already suggested in [Har84]. According to this suggestion, these states are associated with activities that represent physical (and therefore possibly continuous) operations and interactions with the environment.

The Underlying Phase Transition System

Following the graphical representation, we will now identify the phase transition system underlying the picture of Fig. 3.

As state variables we take $V_c = \{x_c, x_m\}$ and $V_d = \{\pi\}$. The variable π is a control variable whose value is a set of basic states of the statechart.

The initial condition is given by

$$\Theta \quad : \quad \pi = \{\mathit{Start}\}$$

There are several transitions. Following is a list of transitions and the transition relations associated with them. We name transitions according to the states from which they depart.

$$\begin{array}{rcl}
\mathit{Start} & : & (\mathit{Start} \in \pi) \wedge (\pi' = \{\mathit{Mouse.rest},\ \mathit{Cat.rest}\}) \wedge (x'_c = x'_m = X_0) \\
\mathit{Mouse.rest} & : & (\mathit{Mouse.rest} \in \pi) \wedge (\pi' = \pi - \{\mathit{Mouse.rest}\} \cup \{\mathit{Mouse.running}\}) \\
\mathit{Cat.rest} & : & (\mathit{Cat.rest} \in \pi) \wedge (\pi' = \pi - \{\mathit{Cat.rest}\} \cup \{\mathit{Cat.running}\}) \\
\mathit{Mouse.running} & : & (\mathit{Mouse.running} \in \pi) \wedge (x_m = 0) \wedge \\
 & & \qquad (\pi' = \pi - \{\mathit{Mouse.running}\} \cup \{\mathit{Mouse.safe}\}) \\
\mathit{Mouse.safe} & : & (\mathit{Mouse.safe} \in \pi) \wedge (x_c = 0) \wedge (\pi' = \mathit{Mouse\text{-}Wins}) \\
\mathit{Cat.running} & : & (\mathit{Cat.running} \in \pi) \wedge (x_c = x_m > 0) \wedge (\pi' = \mathit{Cat\text{-}Wins})
\end{array}$$

There are two activities α_m and α_c representing the running activities of the two participants. Their equations sets are given by

$$\begin{array}{lcrcl}
\alpha_m & : & (\mathit{Mouse.running} \in \pi) & \rightarrow & \dot{x}_m = -V_m \\
\alpha_c & : & (\mathit{Cat.running} \in \pi) & \rightarrow & \dot{x}_c = -V_c
\end{array}$$

The time bounds for transition *Cat.rest* are $[\Delta, \Delta]$. All other transitions are immediate.

Temporal Specification and Verification

At this preliminary stage of research on hybrid systems, we consider as specification language a minimally extended version of temporal logic. The two main extensions are:

- The underlying time domain are time structures of the form T_θ. Since the basic temporal operators $\mathcal{U}$ and $\mathcal{S}$ are defined on arbitrary totally ordered domains, there is no problem in interpreting them over the hybrid time structures.

  More care is needed to deal with the operators $\bigcirc$ and $\odot$. Some (but not all) moments in T_θ have successors or predecessors. For any pair of moments $\langle i, t_i\rangle$ and $\langle i+1, t_{i+1}\rangle$ such that $t_i = t_{i+1}$, $\langle i+1, t_{i+1}\rangle$ is the successor of $t_i = t_{i+1}$, while $t_i = t_{i+1}$ is the predecessor of $\langle i+1, t_{i+1}\rangle$. We define the formula $\bigcirc p$ (respectively, $\odot p$) to hold at moment $m \in \mathsf{T}_\theta$ if m has a successor (respectively, predecessor) m' and p holds at m'.

- The other extension is allowing references to *age formulas* of the form $\Gamma(p)$. We restrict these reference to formulas p that only refer to discrete variables.

A Rule for Invariance

We propose rule INVH for proving invariance properties of hybrid systems.

Premise I4 of the rule claims that if there is a continuous phase starting at some moment m and leading to another moment m', then it preserves φ. The premise refers to variables at m in their unprimed version and to variables at m' in their primed version.

The clause $T = t_1 < T' = t_2$ names the time at m as t_1 and the time at m' as t_2. It also states that time has increased between m and m'. The clause φ states that φ holds at m.

INVH

I1. $\varphi \rightarrow q$

I2. $\Theta \rightarrow \varphi$

I3. $(\rho_\tau \wedge \Gamma(enabled(\tau)) \geq l_\tau \wedge T' = T \wedge \varphi) \rightarrow \varphi'$, for every $\tau \in \mathcal{T}$

I4.
$$\left(\begin{array}{c}(T = t_1 < T' = t_2) \wedge \varphi \wedge \bigwedge_{\alpha \in J} a_\alpha \wedge \bigwedge_{\alpha \in \mathcal{A}-J} \neg a_\alpha \\ \wedge \\ (V_J = \widehat{V}_J(t_1)) \wedge (V'_J = \widehat{V}_J(t_2)) \wedge (V_{\overline{J}} = V'_{\overline{J}}) \\ \wedge \\ (\forall t \in [t_1, t_2])\widehat{\mathcal{E}}_J(t) \wedge (\forall t \in [t_1, t_2))\Big[\bigwedge_{\tau \in \mathcal{T}_0} \neg \widehat{enabled}(\tau)\Big] \\ \wedge \\ \bigwedge_{\tau \in \mathcal{T}_>} \Big(enabled(\tau) \rightarrow (\Gamma(enabled(\tau)) + t_2 - t_1) \leq u_\tau\Big)\end{array}\right) \rightarrow \varphi' \quad \text{for every } J \subseteq \mathcal{A}$$

$\Box q$

Let $J \subseteq \mathcal{A}$ be a subset of activities. The premise considers the case that the set of activities that are operational in the considered phase is J. The conjunction

$$\bigwedge_{\alpha \in J} a_\alpha \wedge \bigwedge_{\alpha \in \mathcal{A}-J} \neg a_\alpha$$

states that the activation conditions that hold at m correspond precisely to the activities that are in J.

Let $\mathcal{E}_J$ be the set of all equations that appear in $\mathcal{E}_\alpha$ for $\alpha \in J$. We denote by V_J the set of all continuous variables that appear on the left hand side of equations in $\mathcal{E}_J$, and by $V_{\overline{J}}$ the set of all other variables, which includes all discrete variables and some continuous variables that do not appear on the left hand side of equations in $\mathcal{E}_J$.

Clearly, if J is the set of equations that are active in a continuous phase, then the only state variables that can change their values are the variables in V_J, while the variables in $V_{\overline{J}}$ retain their old values. Assume that for each variable $x \in V_J$ there exists a (differentiable) function $\widehat{x}(t)$ defined for $t \in [t_1, t_2]$. The conjunction

$$(V_J = \widehat{V}_J(t_1)) \wedge (V'_J = \widehat{V}_J(t_2)) \wedge (V_{\overline{J}} = V'_{\overline{J}})$$

is an abbreviation for

$$\bigwedge_{x \in V_J} (x = \widehat{x}(t_1)) \wedge \bigwedge_{x \in V_J} (x' = \widehat{x}(t_2)) \wedge \bigwedge_{x \in V_{\overline{J}}} (x' = x).$$

These clauses state that, for each variable $x \in V_J$, its value at m is equal to the value of the function $\widehat{x}$ at t_1, and the value of x at m' equals the value of $\widehat{x}$ at t_2. Thus, x and $\widehat{x}$ agree on the endpoints of the phase. In addition, the third clause states that variables not in V_J retain their values from m to m'.

The clause $(\forall t \in [t_1, t_2])\widehat{\mathcal{E}}_J(t)$ states that the set of differential equations $\mathcal{E}_J$ holds for $t \in [t_1, t_2]$. This set is obtained by replacing each $x \in V_J$ by the function $\widehat{x}(t)$.

The clause

$$(\forall t \in [t_1, t_2))\Big[\bigwedge_{\tau \in \mathcal{T}_0} \neg \widehat{enabled}(\tau)\Big]$$

states that no immediate transition is enabled at any intermediate moment m'', $m \preceq m'' \prec m'$. The assertion $\widehat{enabled}(\tau)$ is obtained from $enabled(\tau)$ by replacing each variable $x \in V_J$ by $\widehat{x}(t)$.

The clause

$$\bigwedge_{\tau \in \mathcal{T}_>} \Big(enabled(\tau) \ \rightarrow \ (\Gamma(enabled(\tau)) + t_2 - t_1) \leq u_\tau\Big)$$

states that the time step taken in this phase, i.e., $t_2 - t_1$ is such that no enabled transition becomes overripe during the phase.

The primed assertion φ' is obtained by priming all variables in V_J and replacing each occurrence of $\Gamma(p)$ by

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

Example of Verification

Consider the phase transition system Φ_1 presented in the preceding section. We wish to verify that it satisfies the invariant

$$\Box(x \leq 1).$$

We use rule INVH with $q : x \leq 1$. In this case we can take $\varphi = q$, which trivially satisfies premise I1. Premise I2 is similarly obvious since initially $x = 0 \leq 1$. Premise I3 is also immediate since the (only) transition is enabled only when $x = 1$, and it preserves the value of x. It therefore remains to verify premise I4. There are two candidates for the set of operational activities J: ϕ and $\{\alpha\}$. The case $J = \phi$ is trivial since in this case $V_J = \phi$ and therefore $\varphi' = \varphi$.

Let us examine the case $J = \{\alpha\}$. In this case, $V_J = \{x\}$, $V_{\overline{J}} = \phi$, and $\mathcal{E}_J$ contains only $\dot{x} = 1$. Writing I4 with these concrete values we obtain the implication

$$\left(\begin{array}{c} (T = t_1 < T' = t_2) \ \wedge \ (x \leq 1) \ \wedge \ (y = 0) \\ \wedge \\ (x = \widehat{x}(t_1)) \ \wedge \ (x' = \widehat{x}(t_2)) \ \wedge \ (y = y') \\ \wedge \\ (\forall t \in [t_1, t_2])\Big[\dot{\widehat{x}}(t) = 1\Big] \ \wedge \ (\forall t \in [t_1, t_2))\Big[(\widehat{x}(t) \neq 1) \vee (y \neq 0)\Big] \end{array}\right) \rightarrow (x' \leq 1)$$

Since $x' = \widehat{x}(t_2)$, we have to show that $\widehat{x}(t_2) \leq 1$. From $x \leq 1$, $x = \widehat{x}(t_1)$, $y = 0$, and the fact that for all $t, t_1 \leq t < t_2$, either $\widehat{x}(t) \neq 1$ or $y \neq 0$, we infer that $\widehat{x}(t_1) < 1$ and that for all $t \in [t_1, t_2)$, $\widehat{x}(t) \neq 1$. Since $\widehat{x}(t)$ is a continuous function for $t \in [t_1, t_2]$, $\widehat{x}(t_2)$ cannot be greater than 1 because, by the intermediate value theorem, there would be a $t \in (t_1, t_2)$ such that $\widehat{x}(t) = 1$, contradicting one of the previous conditions.

Thus $\widehat{x}(t_2)$ and, therefore, x' must be lesser than or equal to 1.

An Invariant for the Cat and Mouse Specification

A more interesting invariant concerns the Cat and Mouse example. Here we may want to determine conditions under which the cat will never catch the mouse. A simple calculation leads to the requirement

$$\frac{X_0}{V_m} < \Delta + \frac{X_0}{V_c} \tag{1}$$

The condition states that the time it takes the mouse to reach the wall is smaller than the time it takes the cat to reach the wall. Since both run at constant speed, this guarantees that they will not meet, except at the wall, the mouse arriving there first.

Assume that condition (1) is given and that $\Delta > 0$). We then would like to establish the invariant

$$Cat.running \wedge (x_c = x_m) \Rightarrow x_m = 0. \tag{2}$$

To prove this invariant we use an auxiliary assertion φ given by a conjunction of the following implications

$$\begin{array}{rcll}
Mouse.rest & \rightarrow & x_m = X_0 & \\
Mouse.running & \rightarrow & x_m = X_0 - V_m \cdot \Gamma(Mouse.running) \geq 0 & (3) \\
Mouse.safe & \rightarrow & x_m = 0 & (4) \\
Cat.rest & \rightarrow & x_c = X_0 & \\
Cat.running & \rightarrow & x_c = X_0 - V_c \cdot \Gamma(Cat.running) \geq 0 & (5) \\
Cat.rest & \rightarrow & Mouse.rest \vee Mouse.running \vee Mouse.safe & \\
Cat.running & \rightarrow & Mouse.running \vee Mouse.safe & (6) \\
Mouse.rest \wedge Cat.rest & \rightarrow & \Gamma(Mouse.rest) = \Gamma(Cat.rest) = 0 & \\
Mouse.running \wedge Cat.rest & \rightarrow & \Gamma(Mouse.running) = \Gamma(Cat.rest) \leq \Delta & \\
Mouse.running \wedge Cat.running & \rightarrow & \Gamma(Mouse.running) = \Gamma(Cat.running) + \Delta & (7)
\end{array}$$

Assuming that φ has been shown to satisfy premises I2–I4, we will show that it implies $q : (Cat.running \wedge (x_c = x_m)) \rightarrow x_m = 0$. By implication (6), when the cat is running the mouse is either in state *running* or in state *safe*. If it is in state *safe* then, by implication (4), $x_m = 0$. If it is in state *running*, then the assumption $x_c = x_m$ with implications (3), (5), and (7), leads to $V_c \cdot \Gamma(Cat.running) = V_m \cdot (\Gamma(Cat.running) + \Delta)$. From this, we can conclude $V_c > V_m$ and

$$\Gamma(Cat.running) = \frac{V_m \cdot \Delta}{V_c - V_m}.$$

On the other hand, from implication (5) we obtain $X_0 - V_c \cdot \Gamma(Cat.running) \geq 0$ which can be written as

$$\Gamma(Cat.running) \leq \frac{X_0}{V_c}.$$

Comparing the equality and inequality involving $\Gamma(Cat.running)$, we obtain

$$\frac{V_m \cdot \Delta}{V_c - V_m} \leq \frac{X_0}{V_c},$$

which can be rewritten as

$$\frac{X_0}{V_m} \geq \Delta + \frac{X_0}{V_c},$$

contradicting condition (1).

This shows that, under condition (1), if the cat is in state *running* and $x_c = x_m$, then the mouse can only be in state *safe* with $x_m = 0$, implying assertion q.

A more careful analysis shows that the weaker requirement

$$\frac{X_0}{V_m} \leq \Delta + \frac{X_0}{V_c} \tag{8}$$

is already a sufficient condition for the cat not catching up with the mouse except possibly at the wall. A proof of this fact follows similar lines to the proof presented above.

6 Sampling Computations of Hybrid Systems

The notions of hybrid trace and hybrid computation presented in the preceding section are based on extending the discrete sequence structure of timed computations into the dense time structure T_θ. An alternative approach that we will now consider takes a less radical step and bases the description of hybrid behavior on *sequences* of situations, very similar to timed traces.

The Continuous Step

Assume a given phase transition system Φ with state variables $V = V_c \cup V_d$. Consider two situations $s, s' \in \Sigma_T$. We characterize the relation holding between s and s' if they are possible left and right endpoints of a continuous phase in the behavior of Φ. The characterization uses the notation of premise I4 of rule INVH. This notation writes

J	for	a set of activities
$\mathcal{E}_J$	for	the set of differential equations appearing in $\mathcal{E}_\alpha$ for some $\alpha \in J$
V_J	for	the set of (continuous) variables appearing on the left hand side of some equation in $\mathcal{E}_J$
$V_{\overline{J}}$	for	$V - V_J$
$\widehat{x}(t)$	for	a function representing the value of $x \in V_J$ over the continuous phase
$\widehat{V}_J$	for	the set of functions $\widehat{x}(t)$, $x \in J$.

The formula ρ_{cont} expresses the relation holding between the situations at the two endpoints of a continuous phase. It is defined by

$$\rho_{cont}: \quad \exists J, \widehat{V}_J, t_1, t_2 \left(\begin{array}{c} (T = t_1 < T' = t_2) \ \wedge \ \varphi \ \wedge \ \bigwedge\limits_{\alpha \in J} a_\alpha \ \wedge \ \bigwedge\limits_{\alpha \in \mathcal{A} - J} \neg a_\alpha \\ \wedge \\ (V_J = \widehat{V}_J(t_1)) \ \wedge \ (V_J' = \widehat{V}_J(t_2)) \ \wedge \ (V_{\overline{J}} = V_{\overline{J}}') \\ \wedge \\ (\forall t \in [t_1, t_2]) \widehat{\mathcal{E}}_J(t) \ \wedge \ (\forall t \in [t_1, t_2)) \Big[\bigwedge\limits_{\tau \in \mathcal{T}_0} \neg \widehat{enabled}(\tau) \Big] \end{array} \right)$$

As usual, unprimed situation variables refer to their values in s while primed variables refer to their values in s'. For detailed explanation of the various clauses appearing in ρ_{cont}, we refer the readers to the explanations following rule INVH. Note that we have omitted the clause

$$\bigwedge_{\tau \in \mathcal{T}_>} \Big(enabled(\tau) \ \rightarrow \ (\Gamma(enabled(\tau)) + t_2 - t_1) \leq u_\tau \Big)$$

from the definition of ρ_{cont}. This is because it will be implied by other requirements.

Sampling Computations

In close analogy with the notion of timed computations, we define a *sampling computation* of a phase transition system Φ to be an infinite sequence of situations

$$\sigma : \ s_0, s_1, \ldots,$$

where $s_i \in \Sigma_T$ for each $i = 0, 1, \ldots$, and the time stamps, $s_0[T], s_1[T], \ldots$ form a progressive time sequence, which satisfies the following requirements:

- [*Initiality*] $s_0 \models \Theta$.
- [*Consecution*] For all $i \geq 0$,
  - Either $t_i = t_{i+1}$ and there is a transition $\tau \in \mathcal{T}$ such that $s_{i+1}[V] \in \tau(s_i[V])$, or
  - s_i and s_{i+1} jointly satisfy ρ_{cont}.

  In the first case, we say that τ is *taken* at position i. The second case is described as a *continuous step* being taken at position i.
- [*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 $s_i[V], s_{i+1}[V], \ldots, s_j[V]$ and not taken at any of the positions $i, i+1, \ldots, j-1$. 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.

A temporal axiomatization of sampling computations can be easily obtained by taking all the axioms for timed systems presented in Section 3, except for C_{cons}, which has to be modified. Writing the continuous step relation as $\rho_{cont}(V_T, V_T')$, the axiom for hybrid consecution replacing C_{cons} is given by

$$H_{cons} \ : \ \Box\Big[\rho_{cont}(V_T, \bigcirc V_T) \ \vee \ \bigvee_{\tau \in \mathcal{T}} (waiting(\tau) \geq l_\tau \wedge taken(\tau))\Big].$$

This axiom states that, at any position, either a continuous step or a transition is taken.

How Faithful is the Sampling Model?

The main difference between continuous computations based on dense time structures, as presented in the previous section, and the more conservative notion of sampling computations is in the amount of formally visible details given about the change of continuous variables over continuous phases.

Consider two moments $m_i : \langle i, t_i \rangle$ and $m_{i+1} : \langle i+1, t_{i+1} \rangle$ in a computation of a hybrid system, such that $t_i < t_{i+1}$. Obviously there is a continuous phase delineated by m_i and m_{i+1} in which some continuous variables, say $x \in V_c$, change continuously. Continuous computations represent the history of change of x in this phase by recognizing a continuum of intermediate moments $m, m_i \prec m \prec m_{i+1}$, and specifying a state $\sigma(m)$ for each of them. A sampling computation, on the other hand, officially recognizes only the endpoints m_i and m_{i+1} and hides the history of continuous change between them inside the definition of ρ_{cont}.

What are the implications of this difference? One aspect to be considered is the degree of correspondence with our intuition. Clearly, both approaches admit that there is continuous change occurring in the continuous phase. Why not represent it explicitly?

There is, however, another point which we would like to address and clarify. Assume that for a continuous variable x we claim an invariant such as $\Box(x \neq 1)$, and that a given computation satisfies this requirement. In the continuous computation case, this implies that x differs from 1 at any time point *including* the continuous phases. On the other hand, if a sampling computation $\sigma : s_0, s_1, \ldots$ is claimed to satisfy $\Box(x \neq 1)$, we only know that x differs from 1 at the discrete sampling points but can infer nothing about its value within the continuous phases.

Example

Consider a phase transition system Φ_2 defined as follows:

$V = V_c = \{x\}$
$\Theta : (x = 2)$
Transitions: a single transition τ with $\rho_\tau : (x' = 2)$ and time bounds

$$[l, u] = [1, 1]$$

Activities: a single activity α with conditional equation $\dot{x} = -2$.

This system has the sampling computation

$$\begin{aligned} \sigma :\ & \langle x:2\,,\, T:0 \rangle \xrightarrow{cont} \langle x:0\,,\, T:1 \rangle \xrightarrow{\tau} \langle x:2\,,\, T:1 \rangle \xrightarrow{cont} \\ & \langle x:0\,,\, T:2 \rangle \xrightarrow{\tau} \langle x:2\,,\, T:2 \rangle \xrightarrow{cont} \cdots \end{aligned}$$

In each continuous phase x drops continuously from 2 to 0, and therefore, at time points $T = 0.5, 1.5, 2.5, \ldots$ its value is 1. On the other hand, the formula $\Box(x \neq 1)$ obviously holds over σ. ▪

However, this problem arises only when we consider formulas that hold over *individual* computations. If we consider specifications not at the level of individual computations but at the system level, and restrict our attention to safety properties, this apparent discrepancy disappears. Namely, if $\square(x \neq 1)$ is claimed to be valid over *all* sampled computations then it is guaranteed that x differs from 1 at all time points, including those falling within continuous phases.

This is because, in taking a continuous step, we do not necessarily have to take the maximal time step possible. The definition allows us to stop at any earlier time point. Thus, while Φ_2 has σ as one of its computations, it also has the computation

$$\begin{array}{llllll}\sigma': & \langle x:2\,,\,T:0.0\rangle \xrightarrow{cont} & \langle x:1\,,\,T:0.5\rangle \xrightarrow{cont} & \langle x:0\,,\,T:1.0\rangle \xrightarrow{\tau} & \langle x:2\,,\,T:1.0\rangle \xrightarrow{cont} \\ & \langle x:1\,,\,T:1.5\rangle \xrightarrow{cont} & \langle x:0\,,\,T:2.0\rangle \xrightarrow{\tau} & \langle x:2\,,\,T:2.0\rangle \xrightarrow{cont} & \cdots\end{array}$$

which does not satisfy $\square(x \neq 1)$.

It follows that, at the system level, $\square(x \neq 1)$ is not a valid specification for Φ_2. This observation suggests that the sampling model provides faithful representation of all invariance properties of the form $\square p$, where p is a state formula, that are valid for the hybrid system.

On the other hand, this is not true if we consider other properties. Consider the specification

$$\diamond\,(x = 1)$$

which states that x eventually equals 1. This formula is valid over all continuous computations of Φ_2 and therefore should be considered a valid property of this system. On the other hand, while it is satisfied by σ', it is not satisfied by σ. This shows that non-invariance properties are not fully captured by the sampling model.

The sampling model has been studied in [MSB91] and recommended by Lamport as a simple way to represent hybrid systems.

References

[ACD90] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking for real-time systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

[AH89] R. Alur and T.A. Henzinger. A really temporal logic. In *Proc. 30th IEEE Symp. Found. of Comp. Sci.*, pages 164–169, 1989.

[AH90] R. Alur and T.A. Henzinger. Real-time logics: Complexity and expressiveness. In *Proc. 5th IEEE Symp. Logic in Comp. Sci.*, 1990.

[AL91] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In *Real-Time: Theory in Practice.* Lec. Notes in Comp. Sci., Springer-Verlag, 1991. This volume.

[CHR92] Z. Chaochen, C.A.R Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1992.

[EC82] E.A. Emerson and E.M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comp. Prog.*, 2:241–266, 1982.

[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.

[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.

[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.

[HRR91] K.M. Hansen, A.P. Ravn, and H. Rischel. Specifying and verifying requirements of real-time systems. *Proc. ACM SIGSOFT'91 Conf. on Software for Critical Systems*, 15(5):44–54, 1991.

[KKZ87] R. Koymans, R. Kuiper, and E. Zijlstra. Specifying message passing and real-time systems with real-time temporal logic. In *Esprit 87 Results and Achievements*. North-Holland, 1987.

[KP92] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Lec. Notes in Comp. Sci., Springer-Verlag, 1992. Proceedings of a Symposium, Nijmegen.

[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.

[Lam83] L. Lamport. What good is temporal logic. In R.E.A. Mason, editor, *Proc. IFIP 9th World Congress*, pages 657–668. North-Holland, 1983.

[MP81] Z. Manna and A. Pnueli. Verification of concurrent programs: The temporal framework. In R.S. Boyer and J.S. Moore, editors, *The Correctness Problem in Computer Science*, pages 215–273. Academic Press, London, 1981.

[MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 201–284. Lec. Notes in Comp. Sci. 354, Springer-Verlag, 1989.

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

[MSB91] K. Marzullo, F.B. Schneider, and N. Budhiraja. Derivation of sequential, real-time, process-control programs. Technical report, Cornell University, 1991. To appear in: *Foundations of Real-Time Computing: Formal Specifications and Methods.*

[NRSV90] X. Nicollin, J.-L. Richier, J. Sifakis, and J. Voiron. ATP: an algebra for timed processes. In *Proc. IFIP Working Conference on Formal Description of Programming Concepts, Tiberias, Israel*. North-Holland, 1990.

[NSY91] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In *Real-Time: Theory in Practice*. Lec. Notes in Comp. Sci., Springer-Verlag, 1991. This volume.

[Ost89] J.S. Ostroff. *Temporal Logic for Real-Time Systems.* Advanced Software Development Series. Research Studies Press (John Wiley & Sons), Taunton, England, 1989.

[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*, pages 84–98. Lec. Notes in Comp. Sci. 331, Springer-Verlag, 1988.

[Pnu86] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Current Trends in Concurrency*, pages 510–584. Lec. Notes in Comp. Sci. 224, Springer-Verlag, 1986.

[PS91] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, pages 244–264. Lec. Notes in Comp. Sci. 526, Springer-Verlag, 1991.

[RR87] G.M. Reed and V.W. Roscoe. Metric spaces as models for real-time concurrency. In *Mathematical Foundations of Programming*, pages 331–343. Lec. Notes in Comp. Sci. 298, Springer-Verlag, 1987.

[San89] E. Sandewall. Combining logic and differential equations for describing real-world systems. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning*, pages 412–420. Morgan Kaufmann, 1989.

[SBM91] F. B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In *Real-Time: Theory in Practice*. Lec. Notes in Comp. Sci., Springer-Verlag, 1991. This volume.