Models for Reactivity [*,†]

Zohar Manna [‡]   Amir Pnueli [§]

October 17, 1994

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

For each of the three levels, we present a computational model, a requirement specification language based on extensions of temporal logic, system description languages based on Statecharts and a textual programming language, proof rules for proving validity of properties, and examples of such proofs.

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

Contents

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

[†]A preliminary version of Section 4 appeared in the proceedings of the Workshop on Hybrid Systems, Lyngby, Oct., 92.

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

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

1 Introduction

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

As is the case with the application of mathematics to other scientific and engineering disciplines, no single model can fully capture the physical phenomenon under study. Instead, we construct a hierarchy of models, each refining (but not necessarily invalidating) its predecessor by the inclusion of additional details.

A good example of the efficient utilization of a hierarchy of models can be found in hardware design, where the orderly development of a large circuit may proceed through several stages, starting at a functional system specification and proceeding through register transfer description, gate level description, device level representation, layout design, and so on. Each of those descriptions adds more details to its predecessor but is consistent with it, and in many cases is even derived from the previous stage. Some interesting approaches even propose multi-level simulation and analysis in which different parts of the same system are represented at different levels of detail.

Following this approach, this paper presents a hierarchy of three models for the specification and verification of reactive systems:

- A *reactive systems* model that captures the *qualitative* (non-quantitative) temporal precedence aspect of time. This model can only identify that one event precedes another but not by how much.

- A *real-time systems* model that captures the *metric* aspect of time in a reactive system. This model can measure the time elapsing between two events.

- A *hybrid systems* model that allows the inclusion of *continuous* components in a reactive real-time system. Such continuous components may cause continuous change in the values of some state variables according to some physical or control law.

For each of these levels of description, the paper provides:

- A *computational model* defining the set of behaviors (computations) that are to be associated with systems in the considered model.

- A *requirement specification* language for specifying properties of systems within the model. The languages we will consider are all variants of temporal logic extended to deal with the new aspects included in the model.

- A *system description* language for describing systems within the model. We will use both a textual programming language and appropriate extensions of the graphical language of statecharts [Har87] to present systems.

- A set of *proof rules* by which valid properties of systems can be verified, showing that the systems satisfy their specifications.

- Examples illustrating the use of the presented proof rules for the verification of properties.

While the qualitative model is well established and has been in use for several years now (e.g., [MP91b]), the real-time model presented here represents work in progress, and research on the hybrid model has just started.

Excerpts of this paper have been presented in a preliminary form in [MP92a] and [MP92b].

2 Reactive Systems

The qualitative model of reactive systems uses an abstract notion of time, based on the ordering of events during an observed computation. This is the main model used, for example, in [MP91b].

2.1 Computational Model: Fair Transition System

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

- $V = \{u_1, ..., u_n\}$: A finite set of *state variables*. Some of these variables represent *data* variables, which are explicitly manipulated by the program text. Other variables are *control* variables, which represent, for example, the location of control in each of the processes in a concurrent program. We assume each variable to be associated with a domain over which it ranges.

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

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

- $\mathcal{T}$: A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is a function

$$\tau : \Sigma \mapsto 2^{\Sigma},$$

mapping each state $s \in \Sigma$ into a (possibly empty) set of τ-successor states $\tau(s) \subseteq \Sigma$.

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

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

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

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

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

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

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

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

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

which is true in s iff s has some τ-successor.

We require that every state $s \in \Sigma$ has at least one transition enabled on it. This is often ensured by including in $\mathcal{T}$ the *idling* transition τ_I (also called the *stuttering* transition), whose transition relation is $\rho_I : (V = V')$. Thus, s' is a τ_I-successor of s iff $s' = s$.

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

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

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

2.2 A Simple Programming Language: Syntax

To present programs, we introduce a simple concurrent programming language (SPL) in which processes communicate by shared variables. The following is a list of some of the statements with an explanation of their intended meanings. We present only the statements that are used in this paper. The reader is referred to [MP91b] for a description of the full language.

- *Assignment*: For a variable y and an expression e of appropriate type,

  $y \ := \ e$

is an *assignment* statement.

- *Await*: For a boolean expression c,

  await c

is an *await* statement. We refer to condition c as the *guard* of the statement.

Execution of **await** c changes no variables. Its sole purpose is to wait until c becomes true, at which point it terminates.

- *Concatenation*: For statements $S_1, \ldots, S_k$,

  $S_1; \cdots ; S_k$

is a *concatenation* statement. Its intended meaning is sequential composition. The first step in an execution of $S_1; \cdots ; S_k$ is the first step in an execution of S_1. Subsequent steps continue to execute the rest of S_1, and when S_1 terminates, proceed to execute $S_2, S_3, \ldots, S_k$.

In a program presented as a multi-line text, we often omit the separator ';' at the end of a line.

- *While*: For a boolean expression c and a statement S,

  while c **do** S

is a *while* statement. Its execution begins by evaluating c. If c evaluates to F, execution of the statement terminates. Otherwise, subsequent steps proceed to execute S. When S terminates, c is tested again.

Programs

A program P has the form

$$P :: \ \Big[\text{declaration}; \ [P_1 :: \ [\ell_1\colon \ S_1; \ \ \widehat{\ell}_1 :] \ \| \ \cdots \ \| \ P_m :: \ [\ell_m\colon \ S_m; \ \ \widehat{\ell}_m :]]\Big],$$

where $P_1 :: [\ell_1\colon \ S_1; \ \widehat{\ell}_1 :], \ldots, P_m :: [\ell_m\colon \ S_m; \ \widehat{\ell}_m :]$ are *named processes*. The names of the program and of the processes are optional, and may be omitted. The *body* $[\ell_i\colon \ S_i; \ \ \widehat{\ell}_i :]$ of process P_i consists of a statement S_i and an *exit label* $\widehat{\ell}_i$, which is where control resides after execution of S_i terminates. Label $\widehat{\ell}_i$ can be viewed as labeling an empty statement following S_i.

A declaration consists of a sequence of *declaration statements* of the form

variable, ..., variable: type **where** φ.

Each declaration statement lists several variables that share a common type and identifies their type, i.e., the domain over which the variables range. We use *basic types* such as **integer**, **character**, etc., as well as *structured types*, such as **array**, **list**, and **set**.

The optional assertion φ imposes constraints on the initial values of the variables declared in this statement.

Let $\varphi_1, \ldots, \varphi_n$ be the assertions appearing in the declaration statements of a program. We refer to the conjunction $\varphi : \varphi_1 \wedge \cdots \wedge \varphi_n$ as the *data-precondition* of the program.

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

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

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

Figure 1: Program ANY-Y: A simple concurrent program.

2.3 Semantics of the Programming Language

The semantics of the simple programming language is obtained by showing how each program can be viewed as a fair transition system. This is done by identifying each of the components of a fair transition system for a given program.

Consider a program P given by

$$\Big[\text{declaration; } [P_1 :: \ [\ell_1\text{: } S_1; \ \ \widehat{\ell}_1 :] \ \| \ \cdots \ \| \ P_m :: \ [\ell_m\text{: } S_m; \ \ \widehat{\ell}_m :]]\Big].$$

Let L_P denote the set of *locations* of program P. We refer the reader to [MP91b] where a location is defined as an equivalence class of labels. For our simpler treatment here, it suffices to consider locations as identical to labels. We also assume that we know how to compute, for each statement S of a given program, its *post-location*, which is the location reached after the termination of S.

In program ANY-Y, for example, the post-location of statement m_0 is m_1, while the post-location of statement ℓ_1 is ℓ_0.

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

State Variables and States

The *state variables* V for system $\mathcal{S}_P$ consist of the *data variables* $Y = \{y_1, \ldots, y_n\}$ that are declared at the head of the program, and a single *control variable* π. The data variables Y range over their respectively declared data domains. The control variable π ranges over subsets of L_P, i.e., sets of locations. The value of π in a state denotes all the locations of the program in which control currently resides.

For example, the state variables for program ANY-Y are $V : \{\pi, x, y\}$, where x and y range over the integers while π ranges over subsets of $\{\ell_0, \ell_1, \ell_2, m_0, m_1\}$.

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

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

Transitions

To ensure that every state has some transition enabled on it, we uniformly include the idling transition τ_I in the transition system corresponding to each program. The transition relation for τ_I is

$$\rho_I\colon\ V' = V.$$

We proceed to define the transition relations for the transitions associated with each of the previously introduced statements.

- *Assignment*: Consider the statement $[\ell : y := e\,;\ \widehat{\ell} : \ldots]$, where $\widehat{\ell}$ is the post-location of ℓ.

With this statement we associate a transition τ_ℓ, with the transition relation

$$\rho_\ell\colon\quad \ell \in \pi\ \wedge\ \pi' = \pi - \{\ell\} \cup \{\widehat{\ell}\}\ \wedge\ y' = e\ \wedge \bigwedge_{u \in Y - \{y\}} (u' = u)$$

The last conjunct claims that all data variables, excluding y, retain their values over the transition τ_ℓ.

- *Await*: With the statement $[\ell\colon\ \textbf{await}\ c\,;\ \widehat{\ell} : \ldots]$, we associate a transition τ_ℓ, with the transition relation

$$\rho_\ell\colon\quad \ell \in \pi\ \wedge\ \pi' = \pi - \{\ell\} \cup \{\widehat{\ell}\}\ \wedge\ c\ \wedge\ Y' = Y$$

The transition τ_ℓ is enabled only when control is at ℓ and the condition c holds. When taken, it moves from ℓ to location $\widehat{\ell}$. The conjunct $Y' = Y$ stands for the conjunction $\bigwedge_{u \in Y} (u' = u)$.

- *While*: With the statement $[\ell\colon\ \big[\textbf{while}\ c\ \textbf{do}\ [\widetilde{\ell}\colon \widetilde{S}\,]\big];\ \widehat{\ell} : \ldots]$, we associate a transition τ_ℓ with the transition relation

$$\rho_\ell\ :\ \ell \in \pi\ \wedge \left(\begin{array}{c} c\ \wedge\ \pi' = \pi - \{\ell\} \cup \{\widetilde{\ell}\} \\ \vee \\ \neg c\ \wedge\ \pi' = \pi - \{\ell\} \cup \{\widehat{\ell}\} \end{array}\right) \wedge\ Y' = Y.$$

According to ρ_ℓ, when c evaluates to T control moves from ℓ to $\widetilde{\ell}$, and when c evaluates to F control moves from ℓ to $\widehat{\ell}$. Note that, in this context, the post-location of $\widetilde{S}$ is ℓ. Note also that the enabling transition of τ_ℓ is $\ell \in \pi$ which does not depend on the value of c.

Thus, the fair transition system corresponding to program ANY-Y has the transitions

$$\tau_I,\ \tau_{\ell_0},\ \tau_{\ell_1},\ \text{and}\ \tau_{m_0}.$$

Transition τ_{ℓ_1}, for example, has the transition relation

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

The Initial Condition

Let φ denote the *data precondition* of program P. We define the *initial condition* Θ for $\mathcal{S}_P$ as

$$\Theta: \quad \pi = \{\ell_1, \ldots, \ell_m\} \ \wedge \ \varphi.$$

This implies that the first state in an execution of the program begins with the control variable pointing to the initial locations of the processes, and the data variables satisfying the data precondition.

For example, the initial condition for program ANY-Y is given by

$$\Theta: \quad \pi = \{\ell_0, m_0\} \ \wedge \ x = 0 \ \wedge \ y = 0.$$

Justice and Compassion

For the simplistic programming language we have presented, the justice and compassion requirements are straightforward.

- *Justice*: As the *justice* set, we take $\mathcal{T} - \{\tau_I\}$, the set of all transitions, excluding the idling transition τ_I.
- *Compassion*: As the *compassion* set, we take the empty set. This will suffice for the examples presented in this paper. The programs presented in [MP91b] use additional statements such as semaphore and communication statements and these give rise to nonempty compassion sets.

This concludes the definition of the transition system $\mathcal{S}_P$.

Examples of Computations

Identification of the fair transition system $\mathcal{S}_P$ corresponding to a program P gives rise to a set of computations $\mathcal{C}omp(P)$ which can be viewed as the possible executions of P, i.e., $\mathcal{C}omp(P) = \mathcal{C}omp(\mathcal{S}_P)$.

Consider the following computation of (the transition system corresponding to) program ANY-Y:

$$\begin{array}{l} \langle \pi : \{\ell_0, m_0\},\ x : 0,\ y : 0 \rangle \ \xrightarrow{m_0}\ \langle \pi : \{\ell_0, m_1\},\ x : 1,\ y : 0 \rangle \ \xrightarrow{\ell_0} \\ \langle \pi : \{\ell_2, m_1\},\ x : 1,\ y : 0 \rangle \ \xrightarrow{\tau_I} \cdots \xrightarrow{\tau_I} \cdots \end{array}$$

The presentation of this computation contains arrows labeled by the transition that is taken at each step. This computation corresponds to the case that m_0 is the first transition taken. Taking this transition sets x to 1, following which process P_1 terminates in one step leading to the terminal state $\langle \pi : \{\ell_2, m_1\},\ x : 1,\ y : 0 \rangle$. The only transition enabled on this state is τ_I, which is repeated forever.

The following computation corresponds to the case that process P_1 executes statement ℓ_1 before m_0 is executed.

$$\begin{array}{l} \langle \pi : \{\ell_0, m_0\},\ x : 0,\ y : 0 \rangle \ \xrightarrow{\ell_0}\ \langle \pi : \{\ell_1, m_0\},\ x : 0,\ y : 0 \rangle \ \xrightarrow{\ell_1} \\ \langle \pi : \{\ell_0, m_0\},\ x : 0,\ y : 1 \rangle \ \xrightarrow{m_0}\ \langle \pi : \{\ell_0, m_1\},\ x : 1,\ y : 1 \rangle \ \xrightarrow{\ell_0} \\ \langle \pi : \{\ell_2, m_1\},\ x : 1,\ y : 1 \rangle \ \xrightarrow{\tau_I} \cdots \xrightarrow{\tau_I} \cdots \end{array}$$

In a similar way, we can construct for each $n \geq 0$ a computation that executes the body of statement ℓ_0 n times and then terminates in the final state $\langle \pi : \{\ell_2, m_1\},\ x : 1,\ y : n \rangle$.

However, the sequence

$$\begin{array}{llll}
\langle \pi : \{\ell_0, m_0\}\,,\; x:0\,,\; y:0\rangle & \xrightarrow{\ell_0} & \langle \pi : \{\ell_1, m_0\}\,,\; x:0\,,\; y:0\rangle & \xrightarrow{\ell_1} \\
\langle \pi : \{\ell_0, m_0\}\,,\; x:0\,,\; y:1\rangle & \xrightarrow{\ell_0} & \langle \pi : \{\ell_1, m_0\}\,,\; x:0\,,\; y:1\rangle & \xrightarrow{\ell_1} \\
\langle \pi : \{\ell_0, m_0\}\,,\; x:0\,,\; y:2\rangle & \xrightarrow{\ell_0} & \langle \pi : \{\ell_1, m_0\}\,,\; x:0\,,\; y:2\rangle & \xrightarrow{\ell_1} \\
\langle \pi : \{\ell_0, m_0\}\,,\; x:0\,,\; y:3\rangle & \xrightarrow{\ell_0} & \cdots &
\end{array}$$

in which transition m_0 is never taken is not an admissible computation. This is because it violates the justice requirement towards m_0, which is continually enabled but never taken.

This illustrates how the requirement of justice ensures that program ANY-Y always terminates.

2.4 Requirement Specification Language: Temporal Logic

As a requirement specification language for reactive systems (under the qualitative model) we take *temporal logic* [MP91b].

We assume an underlying assertion language $\mathcal{L}$ which contains the predicate calculus and interpreted symbols for expressing the standard operations and relations over some concrete domains. Easy reference to the location of control is provided by the predicate at_ℓ_i, which is an abbreviation for the formula $\ell_i \in \pi$, stating that control is currently at location ℓ_i. We also use the expression $at\_\ell_{i,j}$ as an abbreviation for the disjunction $at\_\ell_i \vee at\_\ell_j$.

We refer to a formula in the assertion language $\mathcal{L}$ as a *state formula*, or simply as an *assertion*.

A *temporal formula* is constructed out of state formulas to which we apply the boolean operators $\neg$ and $\vee$ (the other boolean operators can be defined from these), and the following basic temporal operators:

$\bigcirc$ – Next $\quad \circleddash$ – Previous
$\mathcal{U}$ – Until $\quad \mathcal{S}$ – Since

A *model* for a temporal formula p is an infinite sequence of states $\sigma : s_0, s_1, \ldots$, where each state s_j provides an interpretation for the variables mentioned in p.

Given a model σ, as above, we present an inductive definition for the notion of a temporal formula p holding at a position $j \geq 0$ in σ, denoted by $(\sigma, j) \models p$.

- For a state formula p,
  $(\sigma, j) \models p \quad \Longleftrightarrow \quad s_j \models p$
  That is, we evaluate p locally, using the interpretation given by s_j.
- $(\sigma, j) \models \neg p \quad \Longleftrightarrow \quad (\sigma, j) \not\models p$
- $(\sigma, j) \models p \vee q \quad \Longleftrightarrow \quad (\sigma, j) \models p$ or $(\sigma, j) \models q$
- $(\sigma, j) \models \bigcirc p \quad \Longleftrightarrow \quad (\sigma, j+1) \models p$
- $(\sigma, j) \models p \mathcal{U} q \quad \Longleftrightarrow \quad$ for some $k \geq j, (\sigma, k) \models q$,
  and for every i such that $j \leq i < k, (\sigma, i) \models p$
- $(\sigma, j) \models \circleddash p \quad \Longleftrightarrow \quad j > 0$ and $(\sigma, j-1) \models p$
- $(\sigma, j) \models p \mathcal{S} q \quad \Longleftrightarrow \quad$ for some $k \leq j, (\sigma, k) \models q$,
  and for every i such that $j \geq i > k, (\sigma, i) \models p$

Additional temporal operators can be defined as follows:

$\diamondsuit\, p = \mathrm{T}\,\mathcal{U}\, p$	– Eventually
$\square\, p = \neg \diamondsuit \neg p$	– Henceforth
$p\,\mathcal{W} q = \square\, p \vee (p\,\mathcal{U}\, q)$	– Waiting-for, Unless, Weak Until
$\diamondminus\, p = \mathrm{T}\,\mathcal{S}\, p$	– Sometimes in the past
$\boxminus\, p = \neg \diamondminus \neg p$	– Always in the past
$p\,\mathcal{B} q = \boxminus\, p \vee (p\,\mathcal{S}\, q)$	– Back-to, Weak Since

Another useful derived operator is the *entailment* operator, defined by:

$$p \Rightarrow q \quad \iff \quad \square(p \rightarrow q).$$

We refer to $\bigcirc$, $\mathcal{U}$, $\diamondsuit$, $\square$, and $\mathcal{W}$ as *future operators* and to $\ominus$, $\mathcal{S}$, $\diamondminus$, $\boxminus$, and $\mathcal{B}$ as *past operators*.

A formula that contains no future operators is called a *past* formula. A formula that contains no past operators is called a *future* formula. Note that a state formula is both a past and a future formula.

We refer to the set of variables that occur in a formula p as the *vocabulary of* p.

For a state formula p and a state s such that p holds on s, we say that s is a *p-state*. A state formula that holds on all states is called *assertionally valid*.

For a temporal formula p and a position $j \geq 0$ such that $(\sigma, j) \models p$, we say that j is a *p-position* (in σ). Note that the satisfaction of a past formula at position $j \geq 0$ depends only on the finite prefix $s_0, \ldots, s_j$.

If $(\sigma, 0) \models p$, we say that p *holds* on σ, and denote it by $\sigma \models p$. A formula p is called *satisfiable* if it holds on some model. A formula is called *temporally valid* if it holds on all models.

Two formulas p and q are defined to be *equivalent*, denoted $p \sim q$, if the formula $p \leftrightarrow q$ is valid, i.e., $\sigma \models p$ iff $\sigma \models q$, for all models σ.

In the sequel, we adopt the convention by which a formula p that is claimed to be valid is assertionally valid if p is an assertion, and is temporally valid if p contains at least one temporal operator.

The formulas p and q are defined to be *congruent*, denoted $p \approx q$, if the formula $\square(p \leftrightarrow q)$ is valid, i.e., $(\sigma, j) \models p$ iff $(\sigma, j) \models q$, for all models σ and all positions $j \geq 0$. If $p \approx q$ then p can be replaced by q in any context, i.e., $\varphi(p) \approx \varphi(q)$ for any formula $\varphi(p)$ containing occurrences of p.

The notion of temporal validity requires that the formula holds over *all* models. Given a program P, we can restrict our attention to the set of models which correspond to computations of P, i.e., $\mathcal{C}omp(P)$. This leads to the notion of P-validity, by which a temporal formula p is *P-valid* (valid over program P) if it holds over all the computations of P. Obviously, any formula that is temporally valid is also P-valid for any program P. In a similar way, we obtain the notions of P-satisfiability and P-equivalence.

A state s that appears in some computation of P is called a *P-accessible* state. A state formula is called *P-state valid* if it holds over all P-accessible states. Obviously, any state formula that is assertionally valid is also P-state valid for any program P.

Again, we adopt the convention by which we may refer to a P-state valid formula simply as P-valid.

2.5 Specification of Properties

A temporal formula φ that is valid over a program P specifies a property of P, i.e., states a condition that is satisfied by all computations of P. As is explained in [MP91b], the properties expressible by temporal logic can be arranged in a hierarchy that identifies different classes of properties according to the form of formulas expressing them.

Here we will consider only properties falling into the two most important classes: *safety* and *response*.

Safety Properties

Safety properties are those that can be expressed by a formula

$$\Box\, \psi,$$

for some past formula ψ. We refer to a formula of this form as a *canonical safety formula*.

In this paper, we will mainly consider safety properties expressible by the *invariance* formula $\Box\, \varphi$, where φ is a state formula, and the *waiting-for* formula $p \Rightarrow (\varphi\, \mathcal{W}\, q)$ for state formulas p, φ, and q.

The formula $p \Rightarrow (\varphi\, \mathcal{W}\, q)$ states that, following each p-position, there is a succession of φ-positions that either extends to infinity or is terminated by a q-position. This is a safety property since the waiting-for formula is equivalent to the canonical safety formula

$$\Box\Big(\neg\varphi \;\rightarrow\; (\neg p)\, \mathcal{B}\, q\Big)$$

The latter formula states that every $\neg\varphi$-position j satisfies $(\neg p)\, \mathcal{B}\, q$, meaning that p is false all the way back from j to an occurrence of q or to the beginning of the computation. This implies that whenever a $\neg\varphi$-position is preceded by a p-position, there exists a q-position separating the two (possibly coinciding with either).

The following is a list of several safety formulas that are valid over program ANY-Y and therefore specify properties of this program:

- $\Box(y \geq 0)$
  This formula claims that y is nonnegative in all states appearing in computations of ANY-Y.

- $\Box(at\_\ell_2 \rightarrow x = 1)$
  This formula, that can also be rewritten as $at\_\ell_2 \Rightarrow (x = 1)$, claims that $x = 1$ in every state appearing in computations of ANY-Y at which control is at ℓ_2.

- $at\_\ell_0 \quad \Rightarrow \quad at\_\ell_{0,1}\, \mathcal{W}\, (x \neq 0)$
  This waiting-for formula claims that, starting at any state in which control is at ℓ_0, control within process P_1 is continuously at ℓ_0 or ℓ_1 either forever or until x differs from 0.

Response Properties

Response properties are those that can be expressed by a formula

$$p \;\Rightarrow\; \Diamond\, q,$$

for past formulas p and q. In this paper, we will mainly consider the case that p and q are state formulas.

For example, the response formula $\Theta \Rightarrow \Diamond(at\_\ell_2 \wedge at\_m_1)$ is valid over program ANY-Y. It claims that every state satisfying the initial condition $\Theta : at\_\ell_0 \wedge at\_m_0 \wedge x = 0 \wedge y = 0$ is followed by a terminal state characterized by $at\_\ell_2 \wedge at\_m_1$. This implies that all computations of ANY-Y eventually terminate.

2.6 Verifying Safety Properties

We present several proof rules for establishing the P-validity of a safety formula. From now on, we fix our attention on a particular program P, specified by the components $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C}\rangle$.

Verification Conditions

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

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

where ρ_τ is the transition relation corresponding to τ, and q', the *primed version* of the assertion q, is obtained from q by replacing each variable occurring in q by its primed version. Since ρ_τ holds for two states s and s' iff s' is a τ-successor of s, and q' states that q holds on s', it is not difficult to see that

> if the verification condition $\{p\}\tau\{q\}$ is valid, then every τ-successor of a p-state is a q-state.

For a set of transitions $T \subseteq \mathcal{T}$, we denote by $\{p\}T\{q\}$ the conjunction of verification conditions, containing the conjunct $\{p\}\tau\{q\}$ for each $\tau \in T$.

In the context of program ANY-Y, consider for example the verification condition of ℓ_1 (i.e., transition τ_{ℓ_1}), with respect to assertions $y \geq 0$ and $y > 0$:

$$\{y \geq 0\}\ \ell_1\ \{y > 0\}.$$

Expanding the definition of the verification condition, this yields

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

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

The Initiality Rule

Obviously, the initial condition Θ holds at the first position of every computation of P. Consequently, Θ is a P-valid formula. It is useful to cast this fact in the form of a rule, to which we refer as the *initiality rule* INIT.

$$\text{INIT} \quad \frac{\Theta \Rightarrow \psi}{\psi}$$

The rule states that if Θ entails ψ, i.e., implies ψ at any position, then ψ is P-valid. In this and subsequent rules we employ the convention that a line containing a temporal formula states its P-validity.

It is possible to have a version of the INIT rule in which the premise is the implication $\Theta \rightarrow \psi$, rather than the entailment $\Theta \Rightarrow \psi$. For our use here, the entailment is more convenient.

Rule INIT enables us to infer for program ANY-Y the property

$$\diamond(at\_\ell_2 \wedge at\_m_1)$$

from the entailment

$$\underbrace{\pi = \{\ell_0, m_0\} \wedge x = 0 \wedge y = 0}_{\Theta} \quad \Rightarrow \quad \diamond(at\_\ell_2 \wedge at\_m_1)$$

As shown by this example, which deals with a response property, rule INIT is not restricted to proofs of safety properties. We have presented it here since it is one of the most basic rules and should be considered first.

A Waiting-For Rule

The following rule can be used to establish the P-validity of the waiting-for formula $p \Rightarrow \varphi\,\mathcal{W}\,q$ for assertions p, φ, and q, over a given program P.

$$\begin{array}{lll} \text{WAIT} & \text{W1.} & p \rightarrow (q \vee \varphi) \\ & \text{W2.} & \{\varphi\}\mathcal{T}\{q \vee \varphi\} \\ \hline & & p \quad \Rightarrow \quad \varphi\,\mathcal{W}\,q \end{array}$$

This rule contains two premises, which are state formulas, and a temporal conclusion. By our common convention, a line (premise or conclusion) containing a state formula r claims that r is P-state valid while, as previously explained, a line containing a temporal formula ψ claims that ψ is P-valid.

Premise W1 of the rule claims that any p-state satisfies q or φ. Premise W2 claims that any successor of a φ-state satisfies q or φ. Together, they imply that any p-position in a computation of P initializes a sequence of φ-positions which either extends to infinity or is terminated by a q position. This shows that $p \Rightarrow \varphi\,\mathcal{W}\,q$ holds over all computations of P.

Example 1 Let us apply rule WAIT to establish the property

$$x = 0 \quad \Rightarrow \quad (x = 0)\,\mathcal{W}\,(x = 1)$$

for program ANY-Y.

Clearly, we apply rule WAIT with $p = \varphi : \ x = 0$ and $q : \ x = 1$.

Premise W1 assumes the form

$$x = 0 \quad \rightarrow \quad x = 1 \vee x = 0$$

which is obviously valid.

Premise W2 represents a set of verification conditions

$$\rho_\tau \wedge x = 0 \quad \rightarrow \quad x' = 1 \vee x' = 0,$$

for τ ranging over τ_I, τ_{ℓ_0}, τ_{ℓ_1}, and τ_{m_0}. In principle, we should check each of these four conditions separately. However, as is often the case, many of these transitions can be seen to trivially satisfy the verification condition since they do not modify x. Formally, the transition relation for these transitions contains the conjunct $x' = x$, leading to the obvious validity

$$\underbrace{\cdots \wedge x' = x}_{\rho_\tau} \wedge x = 0 \quad \rightarrow \quad \cdots \vee x' = 0,$$

Thus, we only need to consider transitions that modify x. This leaves transition m_0, whose verification condition can be written as

$$\underbrace{\cdots \wedge x' = 1}_{\rho_{m_0}} \wedge x = 0 \quad \rightarrow \quad x' = 1 \vee \cdots,$$

which is obviously valid. This establishes the property $x = 0 \Rightarrow (x = 0) \, \mathcal{W} \, (x = 1)$ as valid over program ANY-Y.

Monotonicity of Waiting-For Formulas

All the temporal operators are monotonic with respect to implication of state formulas. This means that if $p \rightarrow q$ and $\otimes p$ are both valid then so is $\otimes q$, where $\otimes$ stands for any of the unary temporal operators and p, q are state formulas. Similar monotonicity properties hold for each argument of the binary temporal operators.

This enables inference of a new waiting-for formula from a previously established formula by appropriate weakening and strengthening of the assertions appearing in the formula. The precise premises are listed in rule W-MON.

$$\boxed{\text{W-MON} \qquad \begin{array}{c} p \quad \Rightarrow \quad \varphi \, \mathcal{W} \, q \\ p' \rightarrow p, \quad \varphi \rightarrow \varphi', \quad q \rightarrow q' \\ \hline p' \quad \Rightarrow \quad \varphi' \, \mathcal{W} \, q' \end{array}}$$

Using this rule, we can infer the property $at\_m_0 \wedge x = 0 \Rightarrow (x < 1) \, \mathcal{W} \, (x = 1)$ for program ANY-Y from the previously established $x = 0 \Rightarrow (x = 0) \, \mathcal{W} \, (x = 1)$, using the state validities

$$\begin{array}{lcl} at\_m_0 \wedge x = 0 & \rightarrow & x = 0 \\ \quad x = 0 & \rightarrow & x < 1 \end{array}$$

The combination of rules WAIT and W-MON is complete for proving the P-validity of any waiting-for formula $p \Rightarrow \varphi \, \mathcal{W} \, q$ for assertions p, φ, and q [MP83].

Case Splitting

It often happens that the assertion φ appearing in rule WAIT naturally splits into a disjunction:

$$\varphi \; = \; \bigvee_{i \in M} \varphi_i$$

where M is some finite index range, e.g., $M = \{1, \ldots, m\}$. In this case, it may be easier to prove premises W1 and W2 of the rule in the form:

$$\begin{array}{lllll} \text{W1.} & p & \rightarrow & q \vee \varphi_i & \text{for some } i \in M \\ \text{W2.} & \{\varphi_i\} \; \mathcal{T} \; \{q \vee \varphi\} & & & \text{for every } i \in M \end{array}$$

Consider a proof of the property

$$at\_\ell_0 \wedge x = 0 \quad \Rightarrow \quad (at\_\ell_{0,1} \wedge x = 0) \, \mathcal{W} \, (x = 1)$$

for program ANY-Y. In this case $\varphi : \; at\_\ell_{0,1} \wedge x = 0$ naturally splits into the disjunction $\varphi_0 \vee \varphi_1$, where

$$\begin{array}{lll} \varphi_0 & : & at\_\ell_0 \wedge x = 0 \\ \varphi_1 & : & at\_\ell_1 \wedge x = 0. \end{array}$$

In proving premise W1, it is simpler to prove $at\_\ell_0 \wedge x = 0 \rightarrow \varphi_0$.

In proving W2, it is easier to consider separately the cases of φ_0 and φ_1. This is because we may summarize the effects of the various transitions on a φ_0-state by the following table:

Transition	Successor State
τ_I	satisfies φ_0
ℓ_0	satisfies φ_1
ℓ_1	no successor
m_0	satisfies $q : \; x = 1$

Note that transition ℓ_1 is disabled on φ_0 and therefore the verification condition $\{\varphi_0\}\ell_1\{\psi\}$ holds trivially for an arbitrary ψ.

The effects of transitions on a φ_1-state are summarized in the following table:

Transition	Successor State
τ_I	satisfies φ_1
ℓ_0	no successor
ℓ_1	satisfies φ_0
m_0	satisfies $q : \; x = 1$

Together, these two tables (with the associated formal proofs) establish premise W2.

Proof Diagrams for Waiting-For Formulas

As we become more experienced in conducting proofs according to rule WAIT and similar rules, proofs need not be presented with full formal detail. However, identification of the structure of the verification conditions and how transitions may lead from a state satisfying some φ_i into a state satisfying some φ_j is helpful in increasing confidence in the correctness of the proof. In the preceding discussion we illustrated how this information can be represented by tables. Here we will introduce another representation of this information, provided by proof diagrams.

A *proof diagram* is a directed graph consisting of a finite set of nodes N and a set of directed edges E connecting the nodes. Each node n_i is labeled by an assertion φ_i, and each edge is labeled with the name of a transition.

Two subsets of nodes are identified: $I \subseteq N$, the set of *initial nodes*, and $F \subseteq N$, the set of *terminal nodes*. We denote by $M = N - F$ the set of nonterminal nodes. An example of a proof diagram is presented in Fig. 2. This diagram consists of three nodes

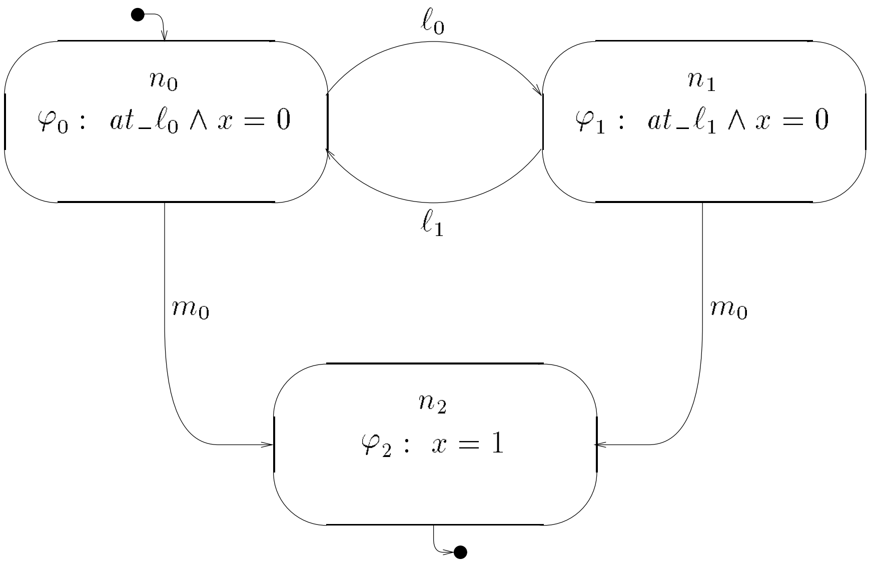


Figure 2: A proof diagram.

n_0, n_1, and n_2. There is one initial node $I = \{n_0\}$ and one terminal node $F = \{n_2\}$. Initial nodes are graphically identified by the annotation while terminal nodes are identified by .

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

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

- If $n_1, \ldots n_k$ are all the τ-successors of n for some $k > 0$, then the verification condition

$$\{\varphi\}\ \tau\ \{\bigvee_{i=1}^{k} \varphi_k\}$$

is assertionally valid.

- If n has no τ-successor then the verification condition

$$\{\varphi\}\ \tau\ \{\varphi\}$$

is assertionally valid.

According to this definition, a node n that has no τ-successor is the same as a node n having a self-connecting edge labeled by τ.

It is obvious that a sound proof diagram identifies a set of verification conditions that can serve as the premises to rule WAIT. Indeed, we have the following claim.

Claim 1 *A sound proof diagram establishes the P-validity of the formula* [1]

$$\bigvee_{i\in I} \varphi_i \quad \Rightarrow \quad (\bigvee_{i\in M} \varphi_i)\,\mathcal{W}\,(\bigvee_{i\in F} \varphi_i)$$

It is not difficult to show that the premises of rule WAIT for the choice

$$p:\ \bigvee_{i\in I} \varphi_i, \qquad \varphi:\ \bigvee_{i\in M} \varphi_i, \qquad q:\ \bigvee_{i\in F} \varphi_i$$

follow from the soundness of the proof diagram.

Premise W1 requires showing

$$\bigvee_{i\in I} \varphi_i \quad \rightarrow \quad (\bigvee_{i\in F} \varphi_i) \vee (\bigvee_{i\in M} \varphi_i).$$

This follows from the fact that $I \subseteq (F \cup M) = N$.

Premise W2 requires showing, for each node $n_j \in M$ labeled by φ_j and each transition $\tau \in \mathcal{T}$, the validity of the verification condition

$$\{\varphi_j\}\ \tau\ \{\bigvee_{i\in F\cup M} \varphi_i\}.$$

However, this follows immediately from the soundness of the proof diagram. ∎

To simplify the notation, we will often write φ_K as an abbreviation for the disjunction

$$\bigvee_{i\in K} \varphi_i,$$

for any $K \subseteq N$. With this notation, the conclusion of Claim 1 can be rewritten as

$$\varphi_I \quad \Rightarrow \quad \varphi_M\,\mathcal{W}\,\varphi_F.$$

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

To increase our proving power, we often combine proof diagrams with monotonicity arguments.

It is not difficult to see that the diagram presented in Fig. 2 is sound for program ANY-Y. This establishes the validity of the formula

$$at\_\ell_0 \land x = 0 \quad \Rightarrow \quad \big((at\_\ell_0 \land x = 0) \lor (at\_\ell_1 \land x = 0)\big)\, \mathcal{W}\, (x = 1)$$

over program ANY-Y. Note that this formula is equivalent to

$$at\_\ell_0 \land x = 0 \quad \Rightarrow \quad (at\_\ell_{0,1} \land x = 0)\, \mathcal{W}\, (x = 1).$$

We define a proof diagram D to be *valid* with respect to assertions p, φ, and q if D is sound and the following implications are assertionally valid:

$$p \;\rightarrow\; \bigvee_{i \in I} \varphi_i, \qquad (\bigvee_{i \in M} \varphi_i) \;\rightarrow\; \varphi, \qquad (\bigvee_{i \in F} \varphi_i) \;\rightarrow\; q.$$

Combining claim 1 with monotonicity, we obtain the following corollary:

Corollary 1 *If diagram D is valid with respect to assertions p, φ, and q, then the formula*

$$p \quad \Rightarrow \quad \varphi\, \mathcal{W}\, q$$

is P-valid.

Consider the sound diagram of Fig. 2 and the assertions

$$p:\; \Theta, \qquad \varphi:\; at\_\ell_{0,1}, \qquad q:\; x \neq 0.$$

The required monotonicity conditions for these three assertions are

$$\begin{array}{rcl} \Theta & \rightarrow & at\_\ell_0 \land x = 0 \\ (at\_\ell_0 \land x = 0) \lor (at\_\ell_1 \land x = 0) & \rightarrow & at\_\ell_{0,1} \\ x = 1 & \rightarrow & x \neq 0 \end{array}$$

All three are valid. It follows that the diagram of Fig. 2 is valid with respect to this choice of p, φ, and q, and therefore that the formula

$$\Theta \quad \Rightarrow \quad (at\_\ell_{0,1})\, \mathcal{W}\, (x \neq 0)$$

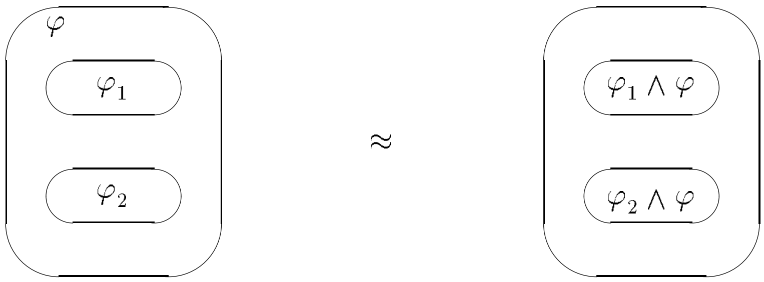
is valid over program ANY-Y.

Statechart Conventions

There are several conventions inspired by the visual language of statecharts [Har87] that improve the presentation and readability of proof diagrams. We extend the notion of a directed graph into a structured directed graph by allowing *compound nodes* that may encapsulate other nodes, and edges that may depart or arrive at compound nodes. A node that does not encapsulate other nodes is called a *basic node*. The role of compound nodes in a structured proof diagram is to provide a more succinct representation of the assertions labeling the basic nodes and the edges (labeled by transitions) that connect them.

We use the following conventions:

- Labels of compound nodes: a diagram containing a compound node n, labeled by an assertion φ and encapsulating nodes $n_1, \ldots, n_k$ with assertions $\varphi_1, \ldots, \varphi_k$, is equivalent to a diagram in which n is unlabeled and nodes $n_1, \ldots, n_k$ are labeled by $\varphi_1 \wedge \varphi, \ldots, \varphi_k \wedge \varphi$. This convention allows us to factor out a conjunct common to the encapsulated nodes and place it as a label of the compound node.



- Edges entering and exiting compound nodes: a diagram containing an edge e connecting node A to a compound node n encapsulating nodes $n_1, \ldots, n_k$ is equivalent to a diagram in which there is an edge connecting A to each n_i, $i = 1, \ldots, k$, with the same label as e. Similarly, an edge e connecting the compound node n to node B is the same as having a separate edge connecting each n_i, $i = 1, \ldots, k$, to B with the same label as e. These equivalences are illustrated in Fig. 3.

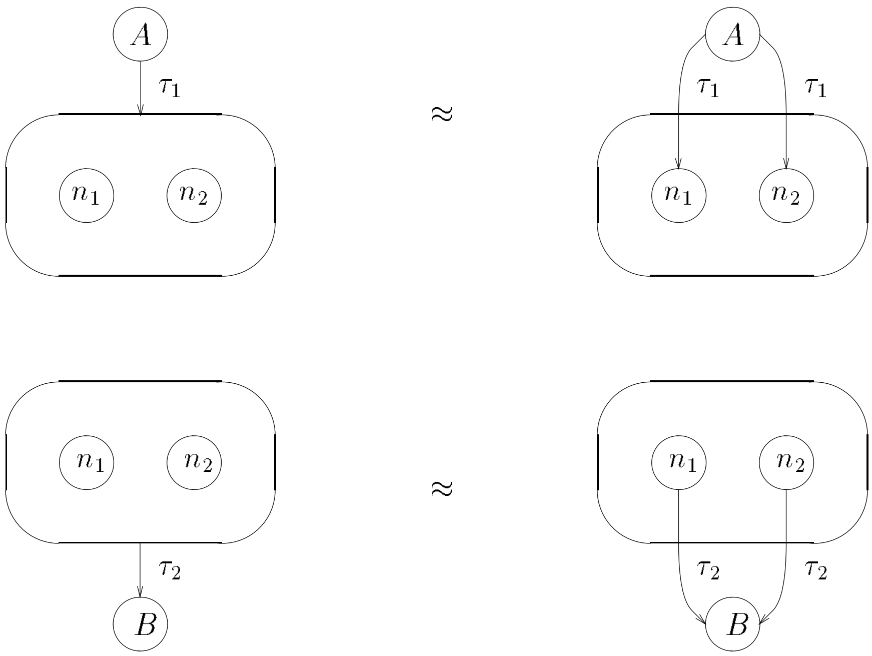


Figure 3: Edges entering and exiting compound nodes.

- Compound nodes designated as initial and terminal nodes: a diagram in which a compound node n is designated as an *initial* (respectively, *terminal*) node is the same as having all the nodes encapsulated by n designated as initial (respectively, terminal). These equivalences are illustrated in Fig. 4.

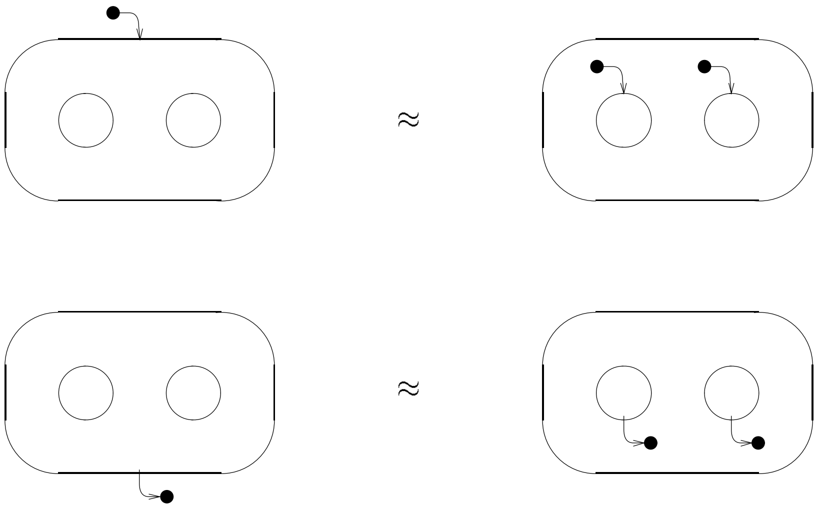


Figure 4: Initial and terminal compound nodes.

With these conventions we can redraw the proof diagram of Fig. 2 as shown in Fig. 5. Note that the common conjunct $x = 0$ has been factored out of nodes n_0 and n_1 and now

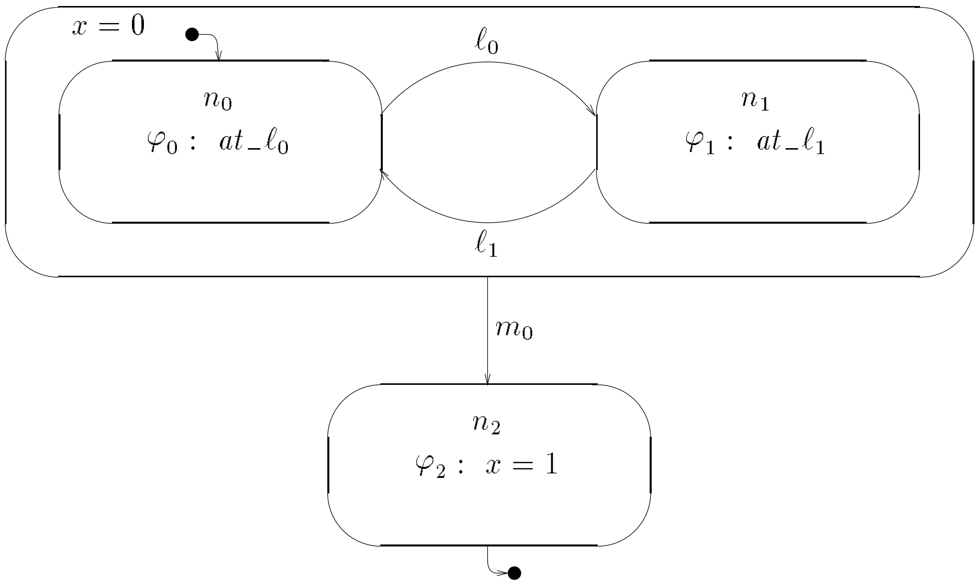


Figure 5: A structured proof diagram.

appears as the label of the compound node encapsulating them.

A Rule for Invariance Formulas

Our approach to proving invariance formulas of the form $\Box\varphi$, for a state formula φ, is based on the congruence

$$\Box\varphi \quad \approx \quad \varphi\,\mathcal{W}\,\mathrm{F}.$$

Consider rule WAIT for the special case that p is taken to be Θ while q is taken to be F. The conclusion for this case is $\Theta \Rightarrow \varphi\,\mathcal{W}\,\mathrm{F}$, which is congruent to $\Theta \Rightarrow \Box\varphi$. Invoking rule INIT, we may infer $\Box\varphi$ as a P-valid conclusion. Consequently, simplifying the premises, we obtain the following rule for proving invariance formulas.

$$\boxed{\begin{array}{lll} \text{INV} & \text{I1.} & \Theta \to \varphi \\ & \text{I2.} & \{\varphi\}\mathcal{T}\{\varphi\} \\ \hline & & \Box\varphi \end{array}}$$

Rule INV states the obvious fact that if assertion φ is implied by the initial condition and preserved by any transition of the program, then it is an invariant of the program, i.e., holds on all P-accessible states.

Let us illustrate the application of rule INV for proving the property $\Box(y \geq 0)$ for program ANY-Y. Clearly, φ is taken to be $y \geq 0$. Premise I1 for this case is

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

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

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

which is also valid. All other transitions preserve the value of y and therefore trivially preserve $\varphi: \; y \geq 0$.

Monotonicity of Invariance Formulas

Invariance formulas are also monotonic with respect to the assertion claimed to be invariant. This is expressed by the following rule I-MON.

$$\boxed{\begin{array}{ll} \text{I-MON} & \Box\varphi \\ & \varphi \to \varphi' \\ \hline & \Box\varphi' \end{array}}$$

Using this rule, we can infer the property $\Box(x \geq 0)$ from a previous proof of the invariance $\Box(x = 0 \vee x = 1)$ (valid for program ANY-Y) and the state validity

$$(x = 0 \vee x = 1) \quad \to \quad x \geq 0.$$

The combination of rules INV and I-MON is complete for proving the P-validity of any invariance formula $\Box\,p$ for assertion p [MP91a].

Proof Diagrams for Invariance

Since, as previously shown, invariance formulas are a special case of waiting-for formulas, it is not surprising that their proof can also be conveniently represented by proof diagrams.

A proof diagram D is defined to be *invariance-sound* if

- D is sound (in the sense defined for waiting-for diagrams).
- There are no terminal nodes, i.e., $F = \phi$ and therefore $M = N$. This reflects the fact that $q = \mathrm{F}$.
- $\Theta \quad \rightarrow \quad \varphi_I$.

Incorporating monotonicity, diagram D is said to be *invariance-valid* with respect to φ if it is invariance-sound (satisfies the three requirements listed above) and, in addition, satisfies

- $\varphi_M \quad \rightarrow \quad \varphi$

The following claim summarizes the use of invariance-valid diagrams.

Claim 2 *If diagram D is invariance-valid with respect to assertion φ, then the formula*

$$\Box\varphi$$

is P-valid.

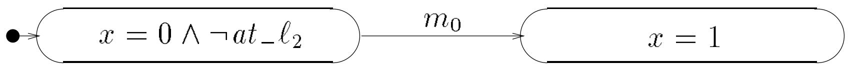
As an example, we present in Fig. 6 a proof diagram that is invariance-valid over program ANY-Y with respect to $\varphi: \; at\_\ell_2 \rightarrow x = 1$. It is not difficult to verify that the diagram of Fig. 6 is sound. One of the important steps in this verification observes that transition ℓ_0 is disabled on node n_1, while transition ℓ_1 is disabled on n_0. Therefore, ℓ_0 trivially preserves $\varphi_1 : at\_\ell_1 \wedge x = 0 \wedge \neg at\_\ell_2$, while ℓ_1 preserves $\varphi_0 : at\_\ell_0 \wedge x = 0 \wedge \neg at\_\ell_2$. It is also evident that this diagram has no final nodes. Invariance-soundness is completed by checking the obviously valid implication

$$\underbrace{\pi = \{\ell_0, m_0\} \wedge x = 0 \wedge \cdots}_{\Theta} \quad \rightarrow \quad \underbrace{at\_\ell_0 \wedge x = 0 \wedge \neg at\_\ell_2}_{\varphi_I : \varphi_0}.$$

Validity with respect to $at\_\ell_2 \rightarrow x = 1$, which can also be written as $\neg at\_\ell_2 \vee x = 1$, follows from the implication

$$\underbrace{(at\_\ell_0 \wedge x = 0 \wedge \neg at\_\ell_2) \;\vee\; (at\_\ell_1 \wedge x = 0 \wedge \neg at\_\ell_2) \;\vee\; (x = 1)}_{\varphi_M : \, \varphi_0 \vee \varphi_1 \vee \varphi_2} \quad \rightarrow \quad \neg at\_\ell_2 \vee x = 1$$

The same property can also be established by the simpler diagram



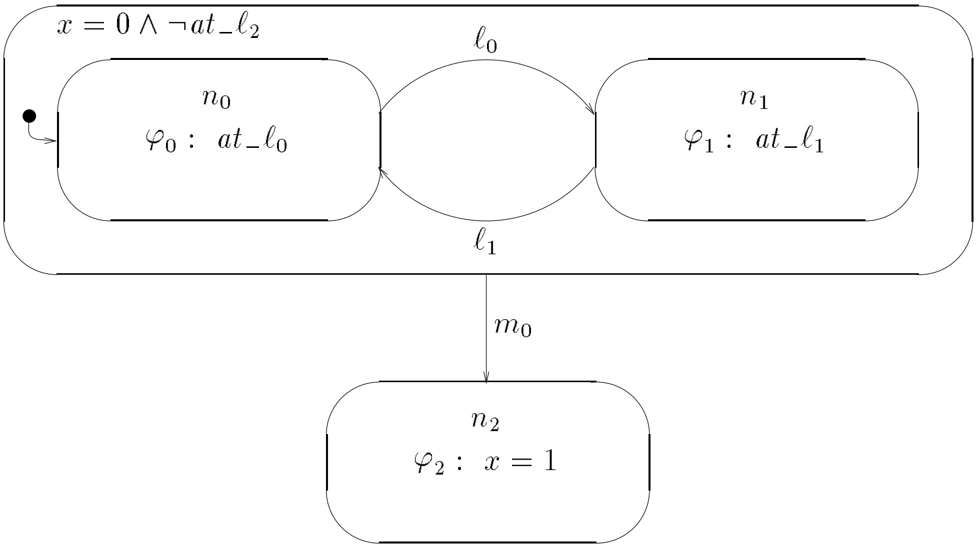


Figure 6: An invariance proof diagram.

2.7 Verifying Response Properties

Here we consider rules (and diagrams) for verifying the validity of a response formula $p \Rightarrow \Diamond\, q$ for the case that p and q are assertions.

The basic response rule RESP relies on a *helpful transition* τ_h whose activation accomplishes the goal q in one helpful step. It also uses an auxiliary assertion φ, characterizing the situation between the occurrence of p and the occurrence of q.

$$
\begin{array}{lll}
\text{RESP} & \text{R1.} & p \Rightarrow (q \lor \varphi) \\
 & \text{R2.} & \{\varphi\}\; \mathcal{T} - \{\tau_h\}\; \{q \lor \varphi\} \\
 & \text{R3.} & \{\varphi\}\; \tau_h\; \{q\} \\
 & \text{R4.} & \varphi \Rightarrow (q \lor En(\tau_h)) \\
\hline
 & & p \Rightarrow \Diamond\, q
\end{array}
$$

Premise R1 ensures that p entails q or φ. Premise R2 states that any transition of the program, excluding τ_h, either leads from φ to q, or preserves φ. Premise R3 states that the helpful transition τ_h leads from φ to q. Premise R4 ensures that τ_h is enabled on any φ-state that does not satisfy q. It is not difficult to see that if p happens, but is not followed by a q, then φ must hold continuously beyond this point, and τ_h is not taken. However, due to R4, this means that τ_h is continuously enabled but never taken, which violates the requirement of justice with respect to τ_h. Consequently, any occurrence of p must be followed by an occurrence of q.

We illustrate the use of rule RESP for proving the response property

$$at\_m_0 \quad \Rightarrow \quad \Diamond(x = 1)$$

for program ANY-Y.

As the helpful transition τ_h we take m_0. As the intermediate assertion φ we take $p: at\_m_0$. Premise R1 assumes the form

$$\underbrace{at\_m_0}_{p} \quad \rightarrow \quad \cdots \vee \underbrace{at\_m_0}_{\varphi},$$

which is obviously valid. Premise R2 requires showing that all transitions, excluding m_0, preserve $\varphi: at\_m_0$ which is clearly the case.

Premise R3 requires showing that m_0 leads from any φ-state to a q-state, expressed by

$$\underbrace{\cdots \wedge x' = 1}_{\rho_{m_0}} \wedge \cdots \quad \rightarrow \quad \underbrace{x' = 1}_{q'},$$

which is obviously valid. Finally, R4 requires

$$\underbrace{at\_m_0}_{\varphi} \quad \rightarrow \quad \cdots \vee \underbrace{at\_m_0}_{En(m_0)},$$

which is also valid. This establishes that the response property $at\_m_0 \Rightarrow \Diamond(x = 1)$ is valid over program ANY-Y.

Combining Response Properties

Not all response properties are achieved by a single activation of a helpful transition. In general, several helpful steps are necessary. In this subsection, we present several rules that may be used to combine single-step response properties into more complex response properties.

First, we list two basic rules, which express the monotonicity and transitivity of response properties.

$$\text{R-MON} \quad \frac{p \Rightarrow \Diamond q \qquad p' \rightarrow p\ ,\ \ q \rightarrow q'}{p' \Rightarrow \Diamond q'} \qquad\qquad \text{R-TRANS} \quad \frac{p \Rightarrow \Diamond q \qquad q \Rightarrow \Diamond r}{p \Rightarrow \Diamond r}$$

The last rule for response is R-CASE, which allows proofs by case analysis.

$$\text{R-CASE} \quad \frac{p \Rightarrow \Diamond q \qquad r \Rightarrow \Diamond q}{(p \vee r) \Rightarrow \Diamond q}$$

We will illustrate the use of these rules by proving termination of program ANY-Y, expressible by

$$\Diamond(at\_\ell_2 \wedge at\_m_1).$$

The proof consists of the following steps:

1. $at\_\ell_0 \wedge at\_m_0 \wedge x = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_{0,1} \wedge at\_m_1 \wedge x = 1)$
   by rule RESP, taking $\tau_h: m_0$ and $\varphi: at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0$
2. $at\_\ell_0 \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$
   by rule RESP, taking $\tau_h: \ell_0$ and $\varphi: at\_\ell_0 \wedge at\_m_1 \wedge x = 1$
3. $at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond(at\_\ell_0 \wedge at\_m_1 \wedge x = 1)$
   by rule RESP, taking $\tau_h: \ell_1$ and $\varphi: at\_\ell_1 \wedge at\_m_1 \wedge x = 1$

4. $at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$
by rule R-TRANS, applied to 3 and 2

5. $(at\_\ell_0 \wedge at\_m_1 \wedge x = 1) \vee (at\_\ell_1 \wedge at\_m_1 \wedge x = 1) \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$
by rule R-CASE, applied to 2 and 4

6. $at\_\ell_{0,1} \wedge at\_m_1 \wedge x = 1 \quad \rightarrow \quad (at\_\ell_0 \wedge at\_m_1 \wedge x = 1) \vee (at\_\ell_1 \wedge at\_m_1 \wedge x = 1)$
an assertional validity

7. $at\_\ell_{0,1} \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$
by rule R-MON, using 5 and 6

8. $at\_\ell_0 \wedge at\_m_0 \wedge x = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$
by rule R-TRANS, applied to 1 and 7

9. $\Theta \quad \rightarrow \quad at\_\ell_0 \wedge at\_m_0 \wedge x = 0$
an assertional validity

10. $\Theta \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$
by rule R-MON, applied to 8 and 9

11. $\Diamond(at\_\ell_2 \wedge at\_m_1)$
by rule INIT, applied to 10

The combination of rules RESP, R-MON, R-TRANS, and R-CASE with well-founded induction is complete for proving the P-validity of any response formula $p \Rightarrow \Diamond\, q$ for assertions p and q [MP91a].

Proof Diagrams for Response

As seen in the previous proof, it is often the case that a proof of a response property involves several applications of rule RESP. Such a proof can be conveniently represented in the form of a *response proof diagram*.

A response diagram is a proof diagram in which some of the edges are drawn in a special font. In this paper, we draw them as dotted lines. We refer to such edges as the *helpful edges*. They correspond to the helpful transitions.

A response diagram D is defined to be *response-sound* if it satisfies

- D is sound, i.e., it satisfies all the verification conditions.
- Every non-terminal node $n_i \in M$ has a helpful edge departing from it.
- The graph of D is acyclic, i.e., D contains no cycle.
- If the helpful edge departing from $n_i \in M$ is labeled by transition τ, then

  $$\varphi_i \quad \rightarrow \quad En(\tau)$$

  is valid. This condition corresponds to premise R4 of rule RESP.

A response diagram is *response-valid* with respect to assertions p and q if it is response-sound and the following two implications are assertionally valid.

$$p \quad \rightarrow \quad \varphi_I$$
$$\varphi_F \quad \rightarrow \quad q$$

The following claim summarizes the use of response-valid diagrams.

Claim 3 *If diagram D is response-valid with respect to assertions p and q, then the formula*

$$p \Rightarrow \Diamond q$$

is P-valid.

In Fig. 7 we present a response diagram that is response-valid over program ANY-Y with respect to $p : \Theta$ and $q : at\_\ell_2 \wedge at\_m_1$. It is not difficult to check that this diagram

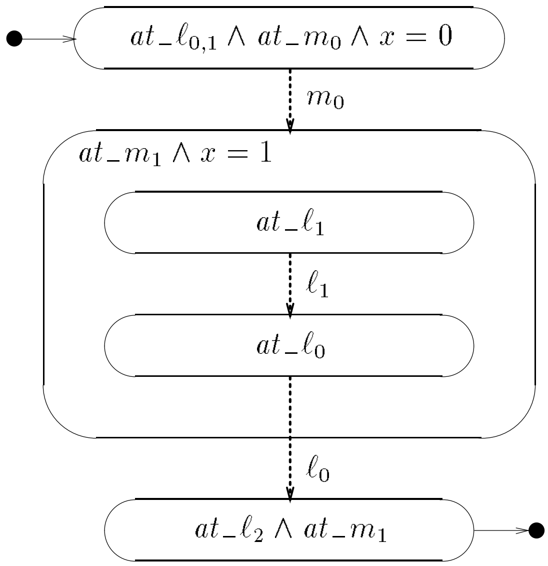


Figure 7: A response proof diagram.

is sound, i.e., satisfies all the verification conditions. It is obviously acyclic, and every M-node (i.e., every node except the final one), has a helpful edge departing from it.

It is also straightforward to check that m_0 is enabled on any state satisfying the assertion $at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0$, ℓ_1 is enabled on any state satisfying at_ℓ_1, and ℓ_0 is enabled on any state satisfying $at\_\ell_0 \wedge at\_m_1 \wedge x = 1$.

Finally, to check validity with respect to Θ and $at\_\ell_2 \wedge at\_m_1$, we observe the state validity

$$\underbrace{at\_\ell_0 \wedge at\_m_0 \wedge x = 0 \wedge y = 0}_{\Theta} \quad \rightarrow \quad at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0$$

and the fact that φ_F is $at\_\ell_2 \wedge at\_m_1$.

We may conclude from the diagram that the response property

$$\Theta \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$$

is valid for program ANY-Y. By rule INIT, we may conclude that ANY-Y always terminates.

It is interesting to note that each helpful edge corresponds to a single application of rule RESP in the previously presented deductive proof of the same property, i.e., steps 1, 2, and 3 in the proof.

3 Real-Time Systems

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

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

3.1 Computational Model: Timed Transition System

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

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

- $V = \{u_1, ..., u_n\}$: A finite set of *state variables.* A state is any type consistent interpretation of V. The set of all states is denoted by Σ.
- Θ : The *initial condition.* A satisfiable assertion characterizing the initial states.
- $\mathcal{T}$: A finite set of *transitions.* Each transition $\tau \in \mathcal{T}$ is a function

  $$\tau : \Sigma \mapsto 2^\Sigma,$$

  defined by a transition relation $\rho_\tau(V, V')$.
- A *minimal delay* $l_\tau \in \mathsf{R}^+$ (also called *lower bound*) for every transition $\tau \in \mathcal{T}$.
- A *maximal delay* $u_\tau \in \mathsf{R}^\infty$ (also called *upper bound*) for every transition $\tau \in \mathcal{T}$. It is required that $u_\tau \geq l_\tau$ for all $\tau \in \mathcal{T}$.

Note that, in going from a fair transition system to a timed transition system, we eliminate the fairness related components of justice and compassion and replace them by the specification of lower and upper bounds.

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

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

Computations

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

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

satisfying:

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

  This implies that τ must be continuously enabled for at least l_τ time units before it can be taken.
- *Upper bound:* For every transition $\tau \in \mathcal{T}$ and position $i \geq 0$, if τ is enabled at position i, there exists a position j, $i \leq j$, such that $t_i + u_\tau \geq t_j$ and τ is disabled on s_j.

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

Unlike the untimed case, it is not necessary to require that every state has at least one transition enabled on it. This is because, even if all transitions are disabled, we can always take tick steps which ensures that all computations are infinite. Consequently, we no longer need the idling transition and its removal causes no harm.

The upper bound requirement claims an equivalence between the formal condition that τ is disabled on s_j, for some $j \geq i$, $t_i + u_\tau \geq t_j$, and the intended requirement that τ cannot be continuously enabled for more than u_τ time units without being taken. This equivalence holds only due to the assumption that transitions are self disabling. Without this assumption, we would have to require that there exists some $j \geq i$, $t_i + u_\tau \geq t_j$, such that either τ is disabled on s_j or τ is taken at position $j - 1$. The simplification resulting from the self-disabling assumption becomes significant when we express the formal condition as a formula.

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

Example

Consider the simple timed transition system given by:

- State Variables $V : \{x, y\}$.
- Initial Condition: $\Theta : (x = 0) \wedge (y = 0)$.
- Transitions: $\mathcal{T} : \{\tau_0, \tau_1, \tau_2\}$ where

τ	ρ_τ	l_τ	u_τ
τ_0	$(y = 0) \wedge even(x) \wedge (x' = x + 1)$	1	2
τ_1	$(y = 0) \wedge odd(x) \wedge (x' = x + 1)$	1	2
τ_2	$(y = 0) \wedge (y' = 1)$	3	3

The predicates $even(x)$ and $odd(x)$ test whether the value of x is even or odd, respectively.

We present two computations of this timed transition system. The first computation σ_1 attempts to let x reach its maximal possible value. Therefore, we always try to activate τ_0 and τ_1 at the first possible position and τ_2, which causes all three transitions to become disabled, as late as possible.

$$\begin{array}{llll} \sigma_1 : & \langle x:0\,,\ y:0\,,\ T:0\rangle \xrightarrow{tick} & \langle x:0\,,\ y:0\,,\ T:1\rangle \xrightarrow{\tau_0} & \langle x:1\,,\ y:0\,,\ T:1\rangle \xrightarrow{tick} \\ & \langle x:1\,,\ y:0\,,\ T:2\rangle \xrightarrow{\tau_1} & \langle x:2\,,\ y:0\,,\ T:2\rangle \xrightarrow{tick} & \langle x:2\,,\ y:0\,,\ T:3\rangle \xrightarrow{\tau_0} \\ & \langle x:3\,,\ y:0\,,\ T:3\rangle \xrightarrow{\tau_2} & \langle x:3\,,\ y:1\,,\ T:3\rangle \xrightarrow{tick} & \cdots \end{array}$$

Note that transition τ_0 cannot be taken before $T \geq 1$ and, after it is taken, we must wait one additional time unit before being able to take τ_1. Transition τ_2 must be taken before time progresses beyond 3 in order to respect its upper bound.

The second computation σ_2 attempts to keep the value of x as low as possible. Consequently, it delays the activation of τ_0 to the latest possible position and tries to activate τ_2 at the earliest possible position.

$$\begin{array}{llll} \sigma_2 : & \langle x:0\,,\ y:0\,,\ T:0\rangle \xrightarrow{tick} & \langle x:0\,,\ y:0\,,\ T:2\rangle \xrightarrow{\tau_0} & \langle x:1\,,\ y:0\,,\ T:2\rangle \xrightarrow{tick} \\ & \langle x:1\,,\ y:0\,,\ T:3\rangle \xrightarrow{\tau_2} & \langle x:1\,,\ y:1\,,\ T:3\rangle \xrightarrow{tick} & \cdots \end{array}$$ ▟

We say that a transition τ is *ripe* at position j if it has been continuously enabled for u_τ time units.

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

- Computations alternate between *tick* steps that advance the clock by a positive amount and (possibly empty) sequences of state-changing transitions that take zero time.
- Transitions *mature* together but *execute* separately in an interleaving manner.
- 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."

The requirement of time divergence excludes *Zeno computations* in which there are infinitely many state-changes within a finite time interval [AL92]. Unfortunately, not every timed transition system is guaranteed to have computations that satisfy all the requirements given above.

Consider, for example, a TTS with a state variable x, initial condition $x = 1$ and two transitions τ_1 and τ_2 whose transition relations and time bounds are given by

τ	ρ_τ	l_τ	u_τ
τ_1	$x > 0 \ \wedge \ x' = -x$	0	0
τ_2	$x < 0 \ \wedge \ x' = -x$	0	0

This TTS does not have a computation. This is because one of τ_1 or τ_2 is always enabled (and ripe) and does not allow time to progress.

A transition whose maximal delay is 0 is called an *immediate transition*. Let $\mathcal{T}_0$ denote the set of all immediate transitions. A *Zeno sequence* is 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 $s_{i+1} \in \tau(s_i)$. The existence of such a sequence may cause the requirement of time divergence to be violated, since time cannot progress until all enabled immediate transitions are taken and, if there are infinitely many of them, time will never progress.

A TTS is called *progressive* if it cannot generate a Zeno sequence. Progressive systems cannot have an infinite chain of immediate transitions and are, therefore, guaranteed to have at least one computation.

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

3.2 System Description by Timed Statecharts

A very convenient specification of timed systems can be obtained by extending the visual notation of statecharts [Har87] by annotating each transition with a pair of numbers $[l, u]$, denoting the lower and upper time bounds of that transition. As an example, we present in Fig. 8 a timed specification of a producer-consumer system.

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

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

$$name : c/\,assignment,$$

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

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

In the description of Fig. 8, states *produce* and *consume* are identified as taking time. This is seen by the fact that the transitions departing from these states have time bounds.

Initially $x = b = y = 0$.

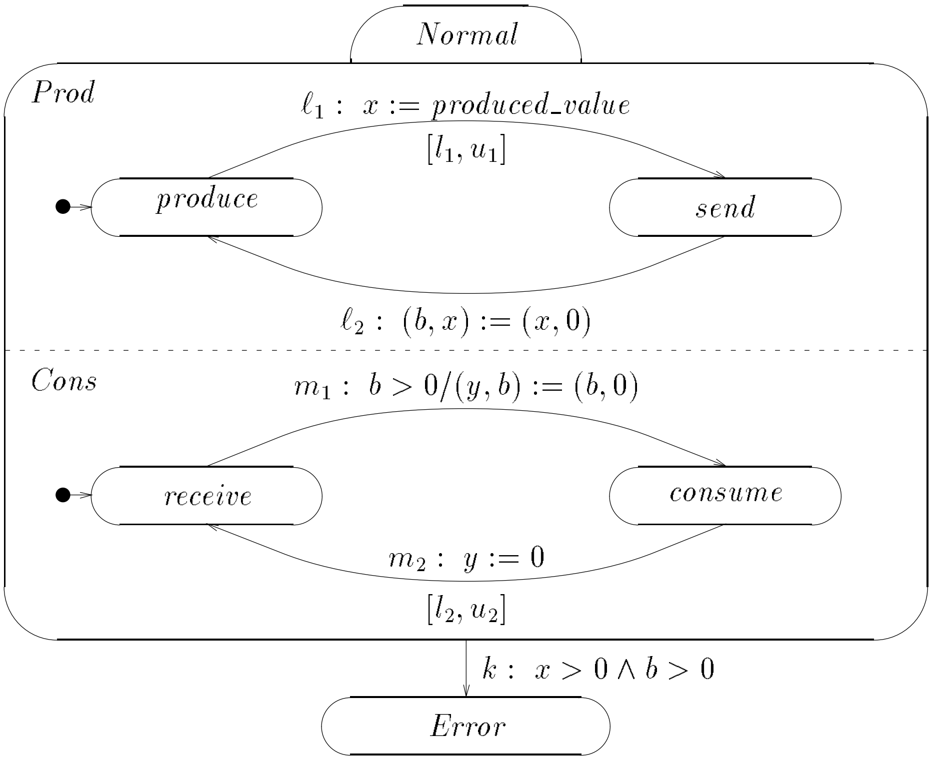


Figure 8: PROD-CONS: A producer consumer system.

On the other hand, states *send* and *receive* are described as immediate, and can be exited as soon as the transitions departing from them are enabled.

In the example presented here, the two concurrent processes communicate by the shared variable b.

The diagram contains a transition k leading from the compound state *Normal* to state *Error*. This transition identifies an error state occurring when the producer is at state *send*, ready to send its next produced value (and hence $x > 0$), while the buffer is still occupied $b > 0$. Obviously, if the producer were to proceed, the value currently stored in b would be lost.

An interesting analysis question one would like to address in this situation is the conditions under which state *Error* is guaranteed to be unreachable. A simple calculation implies that

$$l_1 > u_2$$

is a sufficient condition for the unreachability of *Error*. This is because two consecutive executions of transition ℓ_2 which assigns a positive value to b, must be separated by at least l_1 time units. Assuming that the first assignment caused process *Cons* to move from state *receive* to state *consume*, it will return to state *receive* within at most $u_2 < l_1$ time units. Thus, when b is assigned a new value by ℓ_2, *Cons* is already waiting at state *receive* with $y = 0$.

The associated formal question is how can this fact be proven formally. In the sequel we discuss an approach to the verification of such statements.

Timed Statecharts as a TTS

While we refer the reader to [KP92b] for a full definition of the semantics of timed statecharts, we show here how statechart PROD-CONS of Fig. 8 can be viewed as a timed transition system.

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

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

- *State Variables*: As state variables we take the control variable π and the integer data variables x, b, and y. Variable π ranges over subsets of the basic states *produce*, *send*, *receive*, *consume*, and *Error*.

- *Initial Condition*: given by

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

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

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

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

Timed Extension of the Textual Language

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

On the lowest level, very few extensions are necessary. At the minimum, it is only necessary to assign time bounds to the transitions associated with statements of the program. For example, we can assign uniform time bounds $l_\tau = L$ and $u_\tau = U$ to every transition. As mentioned earlier, the set of transitions associated with a real-time program no longer includes the idling transition τ_I.

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

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

However, we may become more ambitious and attempt to describe within SPL a system such as the producer-consumer system presented in Fig. 8. To do so, we have to extend SPL by additional statements. We refer the reader again to [KP92b] where such extensions are discussed.

To distinguish between the interpretation of a program P as a fair transition system and its interpretation as a timed transition system (when provided time bounds for its transitions), we denote the latter as P_T. For example, the property of termination within $3 \cdot U$ time units is valid for ANY-Y_T but not for program ANY-Y. This property is actually meaningless for ANY-Y, whose computations as a fair transition system do not contain any timing information.

3.3 Requirement Specification Languages

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

To inspect the utility of these languages, we will demonstrate their ability to specify two important timed properties:

- *Bounded response*: Every p should be followed by an occurrence of a q, not later than d time units.
- *Minimal separation*: No q can occur earlier than d time units after an occurrence of p.

Metric Temporal Logic (MTL)

One approach to the specification of timing properties presented in [HMP91] introduces a bounded version of each temporal operator (excluding $\bigcirc$ and $\ominus$) 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 computation $\sigma :\ (s_0, t_0), (s_1, t_1), \ldots$ 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 $\Box_{<u}$ and $\Diamond_{\leq u}$ to stand for $\Box_{[0,u)}$ and $\Diamond_{[0,u]}$.

This approach to the specification of timing properties has been advocated in [KVdR83], [KdR85], and [Koy90], although an early proposal in [BH81] can be viewed as a precursor to this specification style.

Metric temporal logic can easily specify the properties of bounded response and minimal separation.

- Bounded response: $p \Rightarrow \Diamond_{\leq d}\ q$.
- Minimal separation: $p \Rightarrow \Box_{<d}\ \neg q$.

Temporal Logic with Age

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

- the largest t such that, for some $i \leq j$, $t = t_j - t_i$ and φ holds at all positions $i, \ldots, j$, or
- 0 if φ does not hold at position j.

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

In one style of specification, we can specify the two yardstick properties using only references to T but not to Γ.

- Bounded response: $p \ \wedge\ T = t_0 \quad \Rightarrow \quad \Diamond(q\ \wedge\ T \leq t_0 + d)$.
- Minimal separation: $p \ \wedge\ T = t_0 \quad \Rightarrow \quad \Box(T < t_0 + d\ \rightarrow\ \neg q)$.

Another style of specification uses the age function but does not refer directly to the clock T.

- Bounded response: $\Box\Big[\Gamma\Big((\neg q)\,\mathcal{S}\,(p \wedge \neg q)\Big)\ \leq\ d\Big]$.
- Minimal separation: $q \quad \Rightarrow \quad \Big(\boxminus(\neg p)\ \vee\ \Gamma(\neg p) \geq d\Big)$.

The formula for bounded response uses the subformula $(\neg q)\,\mathcal{S}\,(p \wedge \neg q)$, which characterizes points of *ungratified request*, i.e., a position j that is preceded by an occurrence of p but no matching q appears since then till j. The full formula states that periods of ungratified request cannot extend more than d.

The formula for minimal separation claims that an occurrence of q must be preceded by a p-free period that extends either to the beginning of the computation, or for at least d time units.

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

3.4 Verification of MTL Formulas

There are several proof rules that have been proposed for proving properties specified by MTL formulas. We refer the reader to [HMP91] and [Hen91] for a deductive system for such proofs. Here we will illustrate only a set of rules which is adequate for proving bounded response properties.

There is a strong resemblance between the rules for bounded response and the rules for response, presented in subsection 2.7.

The basic response rule RESP relies on a *helpful transition* τ_h whose activation accomplishes the goal q in one helpful step. Let u_h denote the maximal delay associated with τ_h.

B-RESP		
	B1.	$p \Rightarrow (q \vee \varphi)$
	B2.	$\{\varphi\}\ \mathcal{T} - \{\tau_h\}\ \{q \vee \varphi\}$
	B3.	$\{\varphi\}\ \tau_h\ \{q\}$
	B4.	$\varphi \Rightarrow (q \vee En(\tau_h))$
		$p \Rightarrow \Diamond_{\leq u_h} q$

The premises of rule B-RESP are identical to those of rule RESP, but the conclusion states not only that every p is followed by a q but that q must occur within u_h time units. This is because a p not followed by a q initiates a period in which τ_h is continuously enabled, and such a period cannot extend for more than u_h.

Consider program ANY-Y and assume that all transitions, excluding τ_I, are assigned the minimal delay $l : 1$ and the maximal delay $u : 5$. We refer to the resulting timed program and its associated TTS as ANY-Y_T. Rule B-RESP can be used to prove the bounded response property

$$at\_m_0 \quad \Rightarrow \quad \underset{\leq 5}{\Diamond}(x = 1)$$

In fact, no new proof is needed since we have established all the premises for this case in subsection 2.7 while proving the untimed version of this property $at\_m_0 \Rightarrow \Diamond(x = 1)$.

Combining Bounded Response Properties

As in the untimed case, rule B-RESP is useful only for bounded response properties that are achieved by a single activation of a helpful transition. We present here several rules that may be used to combine single-step bounded response properties into more complex bounded response properties.

The following two rules express the monotonicity and transitivity of bounded response properties.

BR-MON	
	$p \Rightarrow \Diamond_{\leq u} q$
	$p' \rightarrow p\ ,\quad q \rightarrow q'$
	$p' \Rightarrow \Diamond_{\leq u} q'$

BR-TRANS	
	$p \Rightarrow \Diamond_{\leq u_1} q$
	$q \Rightarrow \Diamond_{\leq u_2} r$
	$p \Rightarrow \Diamond_{\leq u_1 + u_2} r$

Note that in combining bounded response within u_1 followed by a bounded response within u_2, the resulting response has the upper bound $u_1 + u_2$.

The last rule for bounded response is BR-CASE, which allows proofs by case analysis

BR-CASE	$$\frac{\begin{array}{c} p \Rightarrow \Diamond_{\leq u_1} q \\ r \Rightarrow \Diamond_{\leq u_2} q \end{array}}{(p \vee r) \Rightarrow \Diamond_{\leq max(u_1,u_2)} q}$$

Note that if p ensures a response within u_1 and r ensures a response within u_2, then the best upper bound we can expect from a $(p \vee r)$-state is the maximum of u_1 and u_2.

We will illustrate the use of these rules by proving termination of program ANY-Y$_T$ within 15 time units, expressible by

$$\underset{\leq 15}{\Diamond}(at\_\ell_2 \wedge at\_m_1).$$

The proof follows steps identical to those taken in the proof of untimed termination of program ANY-Y, presented in subsection 2.7.

1. $at\_\ell_0 \wedge at\_m_0 \wedge x = 0 \quad \Rightarrow \quad \Diamond_{\leq 5}(at\_\ell_{0,1} \wedge at\_m_1 \wedge x = 1)$
   by rule B-RESP, taking $\tau_h : m_0$ and $\varphi : at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0$
2. $at\_\ell_0 \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond_{\leq 5}(at\_\ell_2 \wedge at\_m_1)$
   by rule B-RESP, taking $\tau_h : \ell_0$ and $\varphi : at\_\ell_0 \wedge at\_m_1 \wedge x = 1$
3. $at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond_{\leq 5}(at\_\ell_0 \wedge at\_m_1 \wedge x = 1)$
   by rule B-RESP, taking $\tau_h : \ell_1$ and $\varphi : at\_\ell_1 \wedge at\_m_1 \wedge x = 1$
4. $at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond_{\leq 10}(at\_\ell_2 \wedge at\_m_1)$
   by rule BR-TRANS, applied to 3 and 2
5. $(at\_\ell_0 \wedge at\_m_1 \wedge x = 1) \vee (at\_\ell_1 \wedge at\_m_1 \wedge x = 1) \quad \Rightarrow \quad \Diamond_{\leq 10}(at\_\ell_2 \wedge at\_m_1)$
   by rule BR-CASE, applied to 2 and 4
6. $at\_\ell_{0,1} \wedge at\_m_1 \wedge x = 1 \quad \rightarrow \quad (at\_\ell_0 \wedge at\_m_1 \wedge x = 1) \vee (at\_\ell_1 \wedge at\_m_1 \wedge x = 1)$
   an assertional validity
7. $at\_\ell_{0,1} \wedge at\_m_1 \wedge x = 1 \quad \Rightarrow \quad \Diamond_{\leq 10}(at\_\ell_2 \wedge at\_m_1)$
   by rule BR-MON, using 5 and 6
8. $at\_\ell_0 \wedge at\_m_0 \wedge x = 0 \quad \Rightarrow \quad \Diamond_{\leq 15}(at\_\ell_2 \wedge at\_m_1)$
   by rule BR-TRANS, applied to 1 and 7
9. $\Theta \quad \rightarrow \quad at\_\ell_0 \wedge at\_m_0 \wedge x = 0$
   an assertional validity
10. $\Theta \quad \Rightarrow \quad \Diamond_{\leq 15}(at\_\ell_2 \wedge at\_m_1)$
    by rule BR-MON, applied to 8 and 9
11. $\Diamond_{\leq 15}(at\_\ell_2 \wedge at\_m_1)$
    by rule INIT, applied to 10

Proof Diagrams for Bounded Response

Since the premises for the bounded response rules are very similar to those of the response rules, it is straightforward to represent proofs of bounded response properties by response proof diagrams. The only additional information included in bounded-response diagrams is that helpful edges are labeled by a number representing the maximal delay of the associated helpful transition.

The notions of a diagram being response-sound and response-valid with respect to assertions p and q are identical to those of the untimed case.

In Fig. 9 we present a bounded-response diagram that is response-valid over program ANY-Y_T with respect to $p : \Theta$ and $q : at\_\ell_2 \wedge at\_m_1$. This diagram is very similar to the response diagram in Fig. 7 except for the maximal delays annotating the edges.

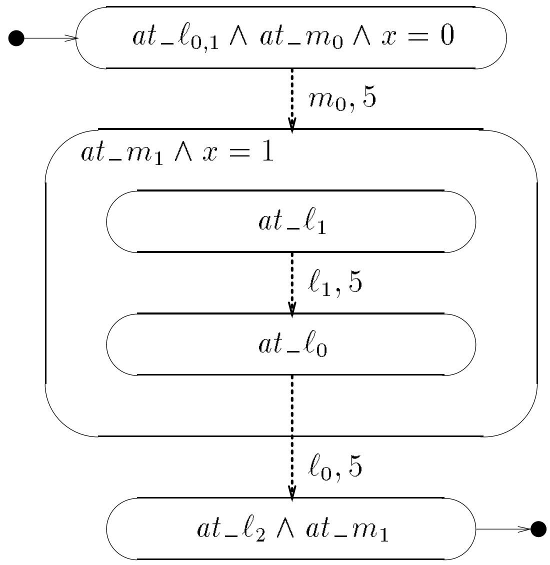


Figure 9: A bounded response proof diagram.

Consider a path connecting an initial node to a terminal node along helpful edges. The *weight* of the path is defined as the sum of upper bounds along the path. For example, the weight of the path traversing the four nodes in the diagram of Fig. 9 is 15. In comparison, the path that proceeds from the initial node directly to the node labeled by at_ℓ_0 has the weight 10.

We define the *weight of a diagram D* to be the maximal weight of a path connecting initial to terminal nodes in the diagram. Obviously, the weight of the diagram in Fig. 9 is 15. It is not difficult to see that the weight of the diagram is the longest delay possible between the occurrence of a state satisfying an initial assertion (an assertion labeling an initial node) and the occurrence of a state satisfying a terminal assertion. This is summarized in the following claim.

Claim 4 *If bounded-response diagram D with weight w is response-valid with respect to assertions p and q, then the formula*

$$p \Rightarrow \underset{\leq w}{\Diamond} q$$

is P_T-valid.

Since the proof diagram of Fig. 9 has been previously shown to be response-valid with respect to Θ and $at\_\ell_2 \wedge at\_m_1$ and its weight is 15, we conclude that the bounded response property

$$\Theta \;\Rightarrow\; \underset{\leq 15}{\Diamond}(at\_\ell_2 \wedge at\_m_1)$$

is valid for program ANY-Y_T. By rule INIT, we may conclude that ANY-Y_T always terminates within 15 time units.

3.5 Verification of Age Formulas

For verifying properties specified by TL_Γ, we develop an extended version of rule WAIT. This will enable us to prove properties expressed by formulas of the form

$$p \;\Rightarrow\; \varphi\,\mathcal{W}\,q$$

for the more general case that p, φ, and q are timed assertions, i.e., state formulas that may contain references to the clock variable T and occurrences of age expressions $\Gamma(\psi)$, where ψ is a state formula.

Before presenting the rule itself, we will present some axioms governing the behavior of age expressions and the timed version of the initiality rule.

Preliminary Axioms and the Initiality Rule

There are several axioms that govern the range of age expressions and are valid over all computations of a timed transition system P_T.

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

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

For timed transition systems, we use a stronger version of the initiality rule INIT. Define Θ_T to be

$$\Theta_T \;:\; \Theta \;\wedge\; T = 0.$$

Then the timed initiation rule T-INIT specifies that any formula entailed by Θ_T is P_T-valid.

$$\boxed{\text{T-INIT}\quad \frac{\Theta_T \;\Rightarrow\; \psi}{\psi}}$$

Verification Conditions

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

- The condition $\{p\}\, \tau\, \{q\}_{\mathcal{T}}$ is given by

  $$\rho^*_\tau \;\wedge\; p \quad\rightarrow\quad q'$$

  where ρ^*_τ stands for

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

  In addition to ρ_τ, ρ^*_τ also requires that time does not progress and that τ has been continuously enabled for at least l_τ and at most u_τ time units.

- The condition $\{p\}\, tick\, \{q\}$ is given by

  $$\rho_{tick} \;\wedge\; p \quad\rightarrow\quad q',$$

  where ρ_{tick} stands for

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

  The formula ρ_{tick} requires that the state variables do not change, time progresses by a positive amount, and no transition becomes over-ripe as a result of the progress of time.

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

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

For a set of transitions $S \subseteq \mathcal{T}$, we say that $\{p\}\, S\, \{q\}_{\mathcal{T}}$ is valid if $\{p\}\, \tau\, \{q\}_{\mathcal{T}}$ is valid for every $\tau \in S$.

As in the untimed case, the validity of $\{p\}\, \mathcal{T}\, \{q\}_{\mathcal{T}}$ and $\{p\}\, tick\, \{q\}$ implies:

if $\langle s, t\rangle$ and $\langle s', t'\rangle$ are two consecutive situations in a computation of P_T and $\langle s, t\rangle$ satisfies p, then $\langle s', t'\rangle$ satisfies q.

Let us check the verification condition

$$\{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\wedge\; T \le 5 + \Gamma(at\_\ell_1) \le 10\} \quad \ell_1$$
$$\{at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\wedge\; T \le 10 + \Gamma(at\_\ell_0) \le 15\}_{\mathcal{T}}$$

for program ANY-Y$_T$ with uniform time bounds $L = 1$ and $U = 5$. Expanding the definition of the condition, we get

$$\left(\begin{array}{c} \underbrace{\pi' = \pi - \{\ell_1\} \cup \{\ell_0\} \;\wedge\; x' = x \wedge \ldots T' = T \wedge \ldots \Gamma(\overbrace{at\_\ell_1}^{En(\ell_1)}) \le \overbrace{5}^{u_{\ell_1}}}_{\rho^*_{\ell_1}} \\ \wedge \\ \underbrace{\cdots \wedge at\_m_1 \;\wedge\; x = 1 \;\wedge\; T \le 5 + \Gamma(at\_\ell_1) \le 10}_{p} \end{array}\right) \quad \rightarrow$$

$$\underbrace{(at\_\ell_0)' \;\wedge\; (at\_m_1)' \;\wedge\; x' = 1 \;\wedge\; T' \le 10 + \Gamma'(at\_\ell_0) \le 15}_{q'}.$$

Clearly $\pi' = \pi - \{\ell_1\} \cup \{\ell_0\}$ implies $(at\_\ell_0)'$ and $(at\_m_1)' = at\_m_1 = \text{T}$. The conjunct $x = 1$ of p and the conjunct $x' = x$ of $\rho^*_{\ell_1}$ imply $x' = 1$. Since $T' = T$, we obtain from $T \leq 5 + \Gamma(at\_\ell_1)$ and $\Gamma(at\_\ell_1) \leq 5$ the inequality $T' \leq 10$ from which, by AGE-RANGE, follows $T' \leq 10 + \Gamma'(at\_\ell_0)$. By axiom UPPER-BOUND, $\Gamma'(at\_\ell_0) \leq 5$, leading to $T' \leq 10 + \Gamma'(at\_\ell_0) \leq 15$.

Next, let us check the verification condition

$$\{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\wedge\; T \leq 5 + \Gamma(at\_\ell_1) \leq 10\}\ tick$$
$$\{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\wedge\; T \leq 5 + \Gamma(at\_\ell_1) \leq 10\}.$$

Expanding its definition we get

$$\left(\begin{array}{c} \underbrace{\pi' = \pi \;\wedge\; x' = x \cdots \wedge T' > T \cdots \wedge \Gamma'(at\_\ell_1) \leq 5}_{\rho_{tick}} \\ \wedge \\ \underbrace{at\_\ell_1 at\_m_1 \wedge x = 1 \wedge T \leq 5 + \Gamma(at\_\ell_1) \leq 10}_{p} \end{array}\right) \rightarrow$$

$$\underbrace{(at\_\ell_1)' \wedge (at\_m_1)' \wedge x' = 1 \wedge T' \leq 5 + \Gamma'(at\_\ell_1) \leq 10}_{q'}$$

Since $\pi' = \pi$ and $at\_\ell_1 \wedge at\_m_1$ holds, so does $(at\_\ell_1)' \wedge (at\_m_1)'$. The conjuncts $x' = x$ and $x = 1$ imply $x' = 1$. Expanding $\Gamma'(at\_\ell_1)$ under $(at\_\ell_1)'$, we obtain $\Gamma'(at\_\ell_1) = \Gamma(at\_\ell_1) + T' - T$. Consider the inequality $T \leq 5 + \Gamma(at\_\ell_1)$. By adding $T' - T$ to both sides, we obtain

$$T' \;\leq\; 5 + \Gamma(at\_\ell_1) + T' - T \;=\; 5 + \Gamma'(at\_\ell_1).$$

Using the conjunct $\Gamma'(at\_\ell_1) \leq 5$ from ρ_{tick}, we conclude

$$T' \;\leq\; 5 + \Gamma'(at\_\ell_1) \;\leq 10,$$

establishing the last conjunct of q'.

A Rule for Timed Waiting-For

The following rule can be used to establish the P_T-validity of the waiting-for formula $p \Rightarrow \varphi\, \mathcal{W}\, q$ for timed assertions p, φ, and q, over a given timed transition system P_T.

$$\begin{array}{llccc} \text{T-WAIT} & \text{W1.} & p & \rightarrow & q \vee \varphi \\ & \text{W2.} & \{\varphi\} & \mathcal{T} & \{q \vee \varphi\}_{T} \\ & \text{W3.} & \{\varphi\} & tick & \{q \vee \varphi\} \\ \hline & & p & \Rightarrow & \varphi\, \mathcal{W}\, q \end{array}$$

Rule T-WAIT can be used in conjunction with the monotonicity rule W-MON (applied to timed assertions). Together they form a complete system for proving timed waiting-for formulas over timed assertions p, φ, and q.

For example, we can use rule T-WAIT to prove the formula

$$\underbrace{at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10}_{p} \quad \Rightarrow$$

$$\overbrace{\Big(at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\Big)}^{\varphi} \ \mathcal{W} \ \underbrace{\Big(at\_\ell_0 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 10 + \Gamma(at\_\ell_0) \leq 15\Big)}_{q}$$

for program ANY-Y_T.

We take p and φ to be $at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10$ and q to be $at\_\ell_0 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 10 + \Gamma(at\_\ell_0) \leq 15$. Premise W1 is trivial since $p = \varphi$. For premise W2 we have to prove

$$\Big\{at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\Big\} \ \tau \ \left\{\begin{array}{l} at\_\ell_0 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 10 + \Gamma(at\_\ell_0) \leq 15 \\ \quad \vee \\ at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10 \end{array}\right\}$$

for every $\tau \in \{\tau_I, m_0, \ell_0, \ell_0, \ell_1\}$. For $\tau = \ell_1$ we have proven

$$\{at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\} \ \ell_1 \ \{at\_\ell_0 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 10 + \Gamma(at\_\ell_0) \leq 15\}_T$$

above. For all other transitions, it is straightforward to prove

$$\{at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\} \ \tau \ \{at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\}_T$$

Premise W3 follows from $\{\varphi\}$ *tick* $\{\varphi\}$,

$$\{at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\} \ tick \ \{at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\}$$

which has also been proven above.

We conclude that the formula

$$at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10 \quad \Rightarrow$$

$$\Big(at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 5 + \Gamma(at\_\ell_1) \leq 10\Big) \ \mathcal{W} \ \Big(at\_\ell_0 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ T \leq 10 + \Gamma(at\_\ell_0) \leq 15\Big)$$

is valid for program ANY-Y_T.

Using Proof Diagrams

To present more elaborate proofs, we may use proof diagrams whose nodes are labeled by timed assertions.

We add to the definition of a sound (and valid) proof diagram the requirement

- For every $n_i \in N$ it is required that the tick verification condition

  $$\{\varphi_i\}\ tick\ \{\varphi_i\}$$

  is valid, implying that each of the assertions is preserved under the progress of time.

Let us consider again the bounded response property which, in MTL, is specified as

$$p \Rightarrow \underset{\leq d}{\diamondsuit}\, q.$$

In this form, as well as in the explicit clock style

$$p \,\wedge\, T = t_0 \quad \Rightarrow \quad \diamondsuit(q \wedge T \leq t_0 + d),$$

this property appears to be a response property. However, this property is actually a *safety* property. We refer the reader to [Hen92] and [Pnu92] where it is pointed out that many liveness properties become safety properties when we consider their bounded version.

In fact, the TL_Γ specification $\Box(\Gamma\big((\neg q)\,\mathcal{S}\,(p \wedge \neg q)\big) \;\leq\; d)$ identifies bounded response as a safety property. Here, however, we prefer to work with another equivalent form

$$p \,\wedge\, T = t_0 \quad \Rightarrow \quad (T \leq t_0 + d)\,\mathcal{W}\,q.$$

This waiting-for formula states that, following every occurrence of p at time t_0, time cannot progress beyond $t_0 + d$ without encountering an occurrence of q.

Let us show that this formula specifies bounded response within delay of at most d. Since time is required to diverge, the waiting-for formula cannot be satisfied by $T \leq t_0 + d$ holding forever. Thus, there must exist a position j such that $T \leq t_0 + d$ still holds at situation $\langle s_j, t_j\rangle$, i.e., $t_j \leq t_0 + d$, while q holds at the next situation $\langle s_{j+1}, t_{j+1}\rangle$. Note that q is an assertion depending only on the state s_{j+1} but not on time. According to the definition of a computation, there are two cases: either $t_j = t_{j+1}$ or $t_j < t_{j+1}$ but then $s_j = s_{j+1}$. In the first case, $t_{j+1} \leq t_0 + d$ so q occurs while the time is still within the bound d. In the second case, q is also satisfied at situation $\langle s_j, t_j\rangle$ and, again, q occurs within bound d.

Let us prove termination of program ANY-Y_T within 15 time units. Using rule T-INIT, it is sufficient to prove

$$\Theta_T \quad \Rightarrow \quad (T \leq 15)\,\mathcal{W}\,(at\_\ell_2 \wedge at\_m_1).$$

The proof diagram presented in Fig. 10 provides a proof for this statement.

It is not difficult to show that the diagram is sound. For example, the verification conditions for φ_1 (including its preservation under a tick step) have been proven above. It

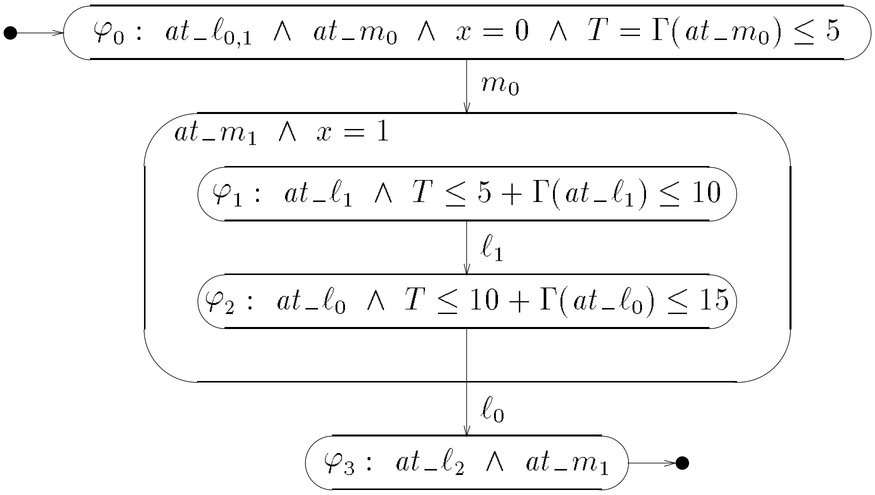


Figure 10: A timed waiting-for proof diagram.

remains to show that the diagram is valid with respect to Θ_T, $T \leq 15$, and $at\_\ell_2 \wedge at\_m_1$. The condition for Θ_T is

$$\underbrace{\pi = \{\ell_0, m_0\} \wedge x = 0 \cdots \wedge T = 0}_{\Theta_T} \quad \rightarrow \quad \underbrace{at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0 \;\wedge\; T = \Gamma(at\_m_0) \leq 5}_{\varphi_I}.$$

Obviously, $\pi = \{\ell_0, m_0\}$ implies $at\_\ell_{0,1} \wedge at\_m_0$, and $T = 0$ implies, by axiom AGE-RANGE, $T = \Gamma(at\_m_0) = 0 \leq 5$. The condition for $T \leq 15$ is

$$\underbrace{\Big(\overbrace{\cdots \wedge T = \cdots \leq 5}^{\varphi_0}\Big) \vee \Big(\overbrace{\cdots \wedge T \leq \cdots \leq 10}^{\varphi_1}\Big) \vee \Big(\overbrace{\cdots \wedge T \leq \cdots \leq 15}^{\varphi_2}\Big)}_{\varphi_M} \quad \rightarrow \quad T \leq 15,$$

which is obviously valid. The condition for $at\_\ell_2 \wedge at\_m_1$ is trivial since $\varphi_F = \varphi_3 = at\_\ell_2 \wedge at\_m_1$.

Proving Untimed Properties of Timed Systems

The previous examples concentrated on proving timed properties, i.e., properties in which time is explicitly mentioned, either via reference to T, or via age expressions, or via bound subscripts on the temporal operators. Another interesting class of properties consists of properties that do not refer to time directly but whose validity over a program P_T is a consequence of the timing constraints satisfied by the computations of P_T.

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

To prove this property, we first derive a timed version of rule INV for establishing the P_T-validity of the invariance formula $\Box\,\varphi$ for a timed assertion φ. This can be done by applying rule T-WAIT with $p = \Theta_T$ and $q = \text{F}$ and observing that $\varphi\,\mathcal{W}\,\text{F}$ is congruent to $\Box\,\varphi$. Using rule T-INIT, we obtain rule T-INV.

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

Rules T-INV and I-MON (applied to timed assertions) serve as the foundation for proving invariance properties by proof diagrams.

We say that a proof diagram is *invariance-valid* with respect to timed assertion φ if it is sound, $F = \phi$, and the following two implications are valid:

$$\begin{array}{lll} \Theta_T & \rightarrow & \varphi_I \\ \varphi_M & \rightarrow & \varphi \end{array}$$

Obviously, if there exists a diagram that is invariance-valid with respect to φ, then $\Box\,\varphi$ is P_T-valid.

In Fig. 11 we present a proof diagram that is invariance-valid with respect to the assertion $y \leq 3$.

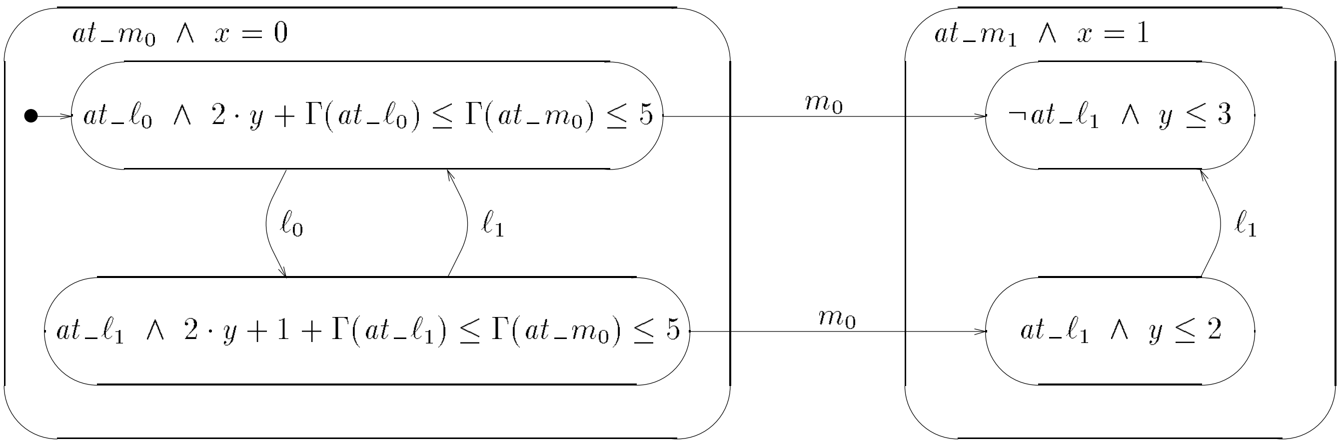


Figure 11: A timed invariance proof diagram.

Let us present some arguments for the validity of this diagram. First, let us check that all assertions appearing in the diagram are preserved by a tick step. The only interesting cases are assertions containing age expressions. The relevant expressions are

$$\begin{array}{ll} 2 \cdot y + \Gamma(at\_\ell_0) \leq \Gamma(at\_m_0) \leq 5 & \text{while } at\_\ell_0 \wedge at\_m_0 \text{ holds,} \\ 2 \cdot y + 1 + \Gamma(at\_\ell_1) \leq \Gamma(at\_m_0) \leq 5 & \text{while } at\_\ell_1 \wedge at\_m_0 \text{ holds.} \end{array}$$

In both cases, when time progresses from T to $T' > T$, both sides of the inequality increase by $T' - T$. Furthermore, $\Gamma'(at\_m_0) \leq 5$ follows from the definition of ρ_{tick}.

Next, we should check the verification conditions corresponding to the transitions labeling edges in the diagram. The verification condition corresponding to ℓ_0 is

$$\begin{array}{l} \rho^*_{\ell_0} \;\wedge\; at\_\ell_0 \;\wedge\; 2 \cdot y + \Gamma(at\_\ell_0) \leq \Gamma(at\_m_0) \leq 5 \\ \qquad \rightarrow \quad (at\_\ell_1)' \;\wedge\; 2 \cdot y' + 1 + \Gamma'(at\_\ell_1) \leq \Gamma'(at\_m_0) \leq 5 \end{array}$$

Clearly, ρ_{ℓ_0}, which is part of $\rho^*_{\ell_0}$, implies $(at\_\ell_1)'$ and $y' = y$. The rest of $\rho^*_{\ell_0}$ implies $\Gamma(at\_\ell_0) \geq 1$ and $T' = T$, which together with at_m_0 leads to $\Gamma'(at\_m_0) = \Gamma(at\_m_0)$. By axiom AGE-FALSE and the definition of Γ', $\Gamma'(at\_\ell_1) = 0$. We thus have

$$2 \cdot y' + 1 + \Gamma'(at\_\ell_1) \;=\; 2 \cdot y + 1 \;\leq\; 2 \cdot y + \Gamma(at\_\ell_0) \;\leq\; \Gamma(at\_m_0) \;=\; \Gamma'(at\_m_0) \;\leq\; 5.$$

On taking transition m_0 from any of the assertions appearing on the left of the diagram, we have that either $2 \cdot y \leq 5$ or $2 \cdot y + 1 \leq 5$. In both cases, this implies $y \leq 2.5 \leq 3$.

It is also straightforward to show the condition

$$\Theta_T \;\rightarrow\; at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x = 0 \;\wedge\; 2 \cdot y + \Gamma(at\_\ell_0) \leq \Gamma(at\_m_0) \leq 5,$$

since Θ_T implies $at\_\ell_0 \wedge at\_m_0 \wedge x = 0$ and $y = \Gamma(at\_\ell_0) = \Gamma(at\_m_0) = 0$.

This shows that the diagram is invariance-sound with respect to $y \leq 3$ and establishes that the formula

$$\square(y \leq 3)$$

is valid over program ANY-Y$_T$.

A Time Dependent Mutual Exclusion Algorithm

As a final example, we present a fragment of a mutual exclusion algorithm, due to M. Fischer, which functions properly only due to the timing constraints associated with the statements in the language. Similar proofs to the one we will present here are given in [SBM92], [AL92], and [MMP92].

The algorithm is presented in Fig. 12. Each of the processes can progress to its second location (ℓ_1 or m_1, respectively) only when $x = 0$. Then, process P_i sets x to $i = 1, 2$. It delays for one instruction time at the next statement which is *skip*. The next statement checks whether x still equals i and, if it does, the process proceeds to its critical section. Of course, in some executions, P_1 may set x to 1 at statement ℓ_1 but find its value to be 2 at ℓ_3, because P_2 has set x to 2 between the execution of these two statements.

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

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

Figure 12: Program MUTEX: Coordination by timing.

The main verification problem associated with this program is the following:

Claim 5 *Assuming all transitions in program* MUTEX$_T$ *are assigned uniform time bounds* L, U, *where* $2 \cdot L > U$, *then the property of mutual exclusion, specifiable as*

$$\square\,\neg(at\_\ell_4 \wedge at\_m_4),$$

is valid for MUTEX$_T$.

In Fig. 13, we present a proof diagram for this property. This diagram employs an additional convention by placing a tabular grid over some of the nodes. The interpretation of the grid is that all nodes belonging to a row of the table have the assertion appearing on the left of that row as a common additional conjunct. In a similar way, all nodes appearing in a column of the table share as a common conjunct the assertion appearing at the top (or bottom) of that column. For example, the full assertion associated with the node appearing at the top left corner of the diagram is:

$$x = 1 \;\wedge\; \underbrace{at\_\ell_4 \;\wedge\; \Gamma(x \neq 0) \geq 2 \cdot L}_{\text{row's common conjunct}} \;\wedge\; \underbrace{at\_m_0}_{\text{column's}}$$

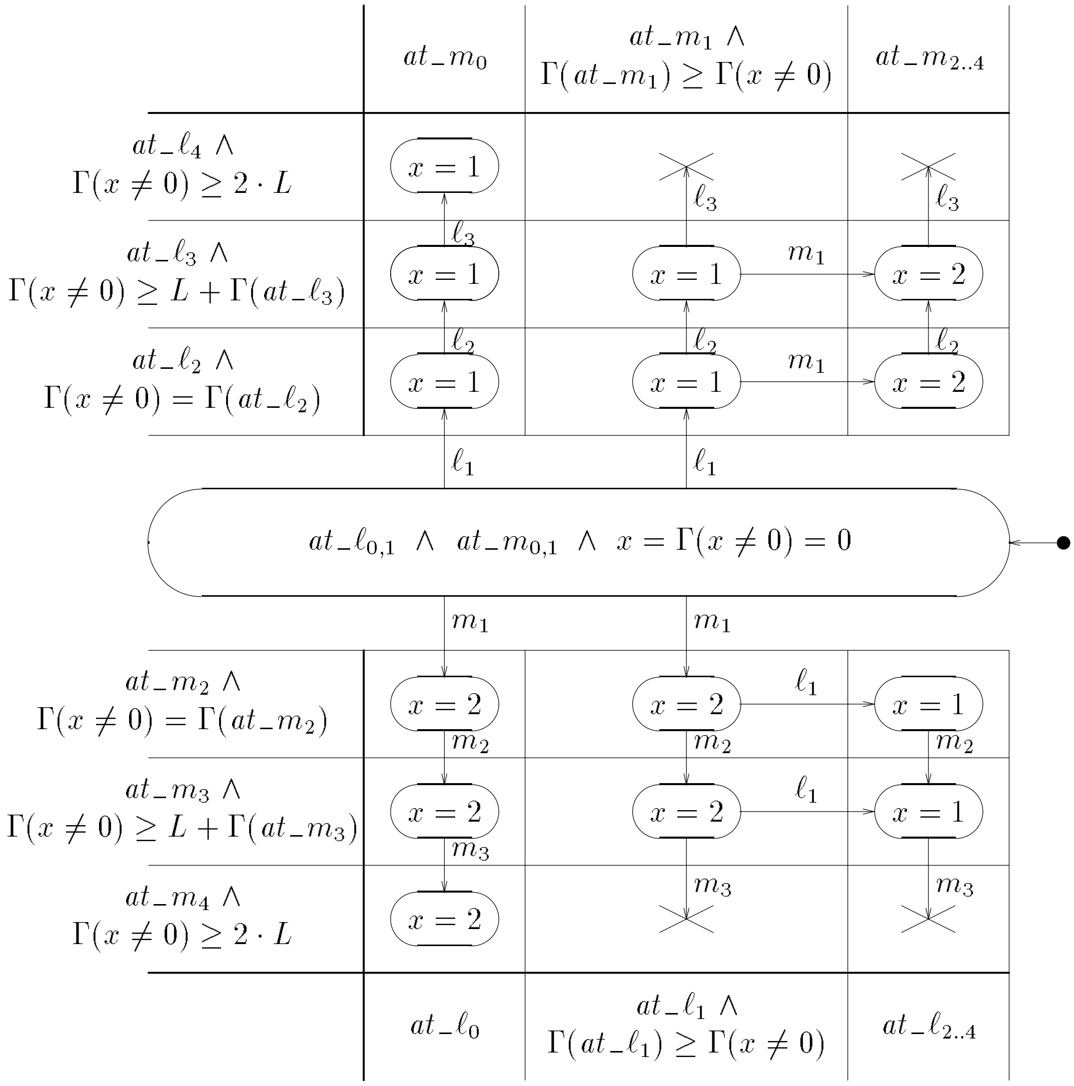


Figure 13: A proof diagram for program MUTEX$_T$.

It is possible to check that the diagram is invariance-valid with respect to the assertion $\neg(at\_\ell_4 \wedge at\_m_4)$. All assertions are preserved by the progress of time. For the initial assertion, this is true since $x \neq 0$ is false as long as the control stays at $\ell_{0,1}, m_{0,1}$. All other assertions refer to time only by equalities or inequalities that either contain age expressions that grow at the same rate on both sides of the equality/inequality, or are of the form $\Gamma(x \neq 0) \geq 2 \cdot L$, in which the left-hand side grows with time, while the right-hand side is constant.

The diagram identifies four transitions as impossible. They are transition ℓ_3 from the two nodes satisfying $at\_\ell_3 \wedge (at\_m_1 \vee (at\_m_{2..4} \wedge x = 2))$ and transition m_3 from the nodes satisfying $at\_m_3 \wedge (at\_\ell_1 \vee (at\_\ell_{2..4} \wedge x = 1))$. There are many other transitions that are impossible, for example m_0 from a node satisfying $at\_\ell_3 \wedge at\_m_0$, which is impossible due to $x = 1$. The reason we singled out these four is that they form the most direct threat to mutual exclusion. That is, if they were possible then mutual exclusion could have been violated.

Transitions ℓ_3 from an $(at\_m_{2..4} \wedge x = 2)$-state and m_3 from an $(at\_\ell_{2..4} \wedge x = 1)$-state are impossible because they are disabled on these states. Taking ℓ_3 from an at_m_1-state or m_3 from an at_ℓ_1-state is impossible due to timing considerations.

Consider, for example, taking transition ℓ_3 from a state (at column 2 and row 2 from the top) satisfying

$$x = 1 \;\wedge\; at\_m_1 \;\wedge\; \Gamma(at\_m_1) \geq \Gamma(x \neq 0) \;\wedge\; at\_\ell_3 \;\wedge\; \Gamma(x \neq 0) \geq L + \Gamma(at\_\ell_3).$$

Combining the two inequalities, such states also satisfy

$$\Gamma(at\_m_1) \;\geq\; L + \Gamma(at\_\ell_3)$$

Transition ℓ_3 can be taken only when $\Gamma(at\_\ell_3) \geq L$ which would lead to

$$\Gamma(at\_m_1) \;\geq\; 2 \cdot L \;>\; U,$$

which violates axiom AGE-RANGE for transition m_1.

The considerations leading to the impossibility of taking m_3 from an at_ℓ_1-state are similar. We conclude that the diagram is valid with respect to $\neg(at\_\ell_4 \wedge at\_m_4)$ and therefore

$$\Box\, \neg(at\_\ell_4 \wedge at\_m_4)$$

is valid for program MUTEX_T.

4 Hybrid Systems

The last model presented here is that of *hybrid systems*. Hybrid systems are systems that combine discrete and continuous components. To represent the continuous components, the hybrid system model contains activities that modify their variables continuously over intervals of positive duration, in addition to the familiar transitions that change the values of variables in zero time, representing the discrete components. The model presented here was first introduced in [MMP92].

It is obvious that many systems that interact with a physical environment, such as a digital module controlling a process or a manufacturing plant, a digital-analog guidance of transport systems, or a control of a robot, can benefit from the more detailed modeling proposed by the comprehensive framework of the hybrid model.

4.1 Computational Model: Phase Transition System

A *phase transition system* (PTS) $\Phi = \langle V, \Theta, \mathcal{T}, \mathcal{A}, l, u, \mathcal{I} \rangle$ consists of:

- $V = \{u_1, ..., u_n\}$: A finite set of *state variables*. The set $V = V_d \cup V_c$ is partitioned into V_d the set of *discrete variables* and V_c the set of *continuous variables*. Continuous variables always have the type *real* (or type *complex*). The discrete variables can be of any type. A state is any type consistent interpretation of V. The set of all states is denoted by Σ.

- Θ : The *initial condition*. A satisfiable assertion characterizing the initial states.

- $\mathcal{T}$: A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is a function

  $$\tau : \Sigma \mapsto 2^{\Sigma},$$

  defined by a transition relation $\rho_\tau(V, V')$. A transition can also change the value of a continuous variable.

  As mentioned earlier, the enabledness of a transition τ can be expressed by the formula

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

  which is true in s iff s has some τ-successor. The enabling condition of a transition τ can always be written as $\delta \wedge \kappa$, where δ is the largest subformula that does not depend on continuous variables. We call κ the *continuous part* of the enabling condition and denote it by $En_c(\tau)$.

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

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

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

  It is required that the activation conditions of the activities that govern the same variable x be exhaustive and exclusive, i.e., exactly one of them holds on any state.

- A *minimal delay* $l_\tau \in \mathsf{R}^+$ for every transition $\tau \in \mathcal{T}$.

- A *maximal delay* $u_\tau \in \mathsf{R}^\infty$ for every transition $\tau \in \mathcal{T}$. We require that $u_\tau \geq l_\tau$ for all $\tau \in \mathcal{T}$.

- A set of *important events* $\mathcal{I}$. This is a finite set of assertions that includes at least the assertions $En_c(\tau)$, for each $\tau \in \mathcal{T}$. These are assertions such that changes in their truth values must be observable. Usually, $\mathcal{I}$ includes, in addition to the assertions $\{En_c(\tau)\}$, all the assertions that appear in specifications of the system.

For simplicity, we require that transitions whose enabling condition depends on a continuous variable be immediate. We also require that every transition is self-disabling.

As in the real-time case, we only consider progressive systems, i.e., systems that do not admit Zeno sequences.

Activity Successors

Consider a phase transition system Φ, and Let $\langle s_1, t_1\rangle$ and $\langle s_2, t_2\rangle$ be two situations of Φ with $t_1 < t_2$. An *evolution* from $\langle s_1, t_1\rangle$ to $\langle s_2, t_2\rangle$ consists of a set of functions $F : \{f_x(t) \mid x \in V_c\}$ that are differentiable in the interval $[t_1, t_2]$ and satisfy the following requirements:

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

- $s_1[y] = s_2[y]$ for every $y \in V_d$. That is, states s_1 and s_2 agree on the values of all discrete variables.

- For every activity $p \rightarrow \dot{x} = e$ that governs x, if p holds at s_1, then f_x satisfies the differential equation
$$\dot{f}_x(t) \quad = \quad e(F)$$
in the interval $[t_1, t_2]$, where the expression $e(F)$ is obtained from e by replacing each occurrence of a variable $y \in V_c$ by the function $f_y(t)$.

- For every assertion $\varphi \in \mathcal{I}$, $\varphi(t)$ has a uniform truth value for all $t \in (t_1, t_2)$, which equals either $\varphi(t_1)$ or $\varphi(t_2)$.

The last requirement ensures that the truth value of every important assertion $\varphi \in \mathcal{I}$ is uniform throughout the interior of the evolution interval, and matches its value at one of the endpoints of the interval. In particular, it disallows a change in the truth value of φ in an internal point. It also guarantees that any value assumed by φ at internal points is also represented at one of the endpoints. This implies that φ cannot be true at both endpoints but false in the middle, nor false at both endpoints but true in the middle.

If such an evolution exists, we say that the situation $\langle s_2, t_2\rangle$ is an *activity successor* of the situation $\langle s_1, t_1\rangle$. Assuming that the differential equations satisfy some reasonable healthiness conditions, such as the Lipschitz condition, there exists at most one evolution from $\langle s_1, t_1\rangle$ to $\langle s_2, t_2\rangle$. In fact, the functions F are uniquely determined by the situation $\langle s_1, t_1\rangle$.

We denote by $\mathcal{A}(\langle s_1, t_1\rangle)$ the set of all activity successors of $\langle s_1, t_1\rangle$.

Consider, for example, a trivial phase transition system with a single (continuous) state variable x, no transitions, a single activity α given by $\alpha : \ \dot{x} = -1$, and an empty $\mathcal{I}$. Then, the following are exmples of a situation and its activity successor:

$$\begin{aligned}
s_1 : \langle x : \ \ 0\,,\, T : 1\rangle &\in \mathcal{A}(s_0 : \langle x : 1\,,\, T : 0\rangle)\\
s_2 : \langle x : -1\,,\, T : 2\rangle &\in \mathcal{A}(s_1 : \langle x : 0\,,\, T : 1\rangle)\\
s_2 : \langle x : -1\,,\, T : 2\rangle &\in \mathcal{A}(s_0 : \langle x : 1\,,\, T : 0\rangle)
\end{aligned}$$

Sampling Computations

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

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

satisfying:

- *Initiation:* $s_0 \models \Theta$ and $t_0 = 0$.
- *Consecution:* For each $j = 0, 1, ...,$
  - Either $t_j = t_{j+1}$ and $s_{j+1} \in \tau(s_j)$ for some transition $\tau \in \mathcal{T}$ — transition τ is taken at j, or
  - $\langle s_{j+1}, t_{j+1} \rangle$ is an activity successor of $\langle s_j, t_j \rangle$ (implying $t_j < t_{j+1}$) — a continuous phase takes place at step j.
- *Lower bound:* For every transition $\tau \in \mathcal{T}$ and position $j \geq 0$, if τ is taken at j, there exists a position i, $i \leq j$, such that $t_i + l_\tau \leq t_j$ and τ is enabled on $s_i, s_{i+1}, \ldots, s_j$.

  This implies that τ must be continuously enabled for at least l_τ time units before it can be taken.
- *Upper bound:* For every transition $\tau \in \mathcal{T}$ and position $i \geq 0$, if τ is enabled at position i, there exists a position j, $i \leq j$, such that $t_i + u_\tau \geq t_j$ and τ is disabled on s_j.

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

Example

Consider a simple phase transition system Φ_1 given by:

State Variables $V = V_c$	:	$\{x\}$
Initial Condition Θ	:	$x = 1$
Transitions $\mathcal{T}$	:	$\{\tau\}$, where $\rho_\tau : (x \leq -1) \wedge (x' = 1)$
Activities $\mathcal{A}$	:	$\{\alpha\}$, where $\alpha : \dot{x} = -1$
Bounds	:	$l_\tau = u_\tau = 0$.
Important events $\mathcal{I}$	:	$\{En_c(\tau) : x \leq -1\}$

Fig. 14 depicts the full behavior of system Φ_1 as a function from $T \in \mathsf{R}^+$ to the value of x. Note that this is not really a function because at $T = 2$ (and all other even positive integers) x has two values: -1 and $+1$. The value -1 is attained at the end of the continuous phase, while $+1$ is the result of taking transition τ at this point.

There are (uncountably) many sampling computations that correspond to this full behavior.

For example, the sampling computation

$$\begin{array}{ll} \sigma_1 : & s_0 : \langle x : 1 \, , T : 0 \rangle \xrightarrow{\alpha} \; s_1 : \langle x : -1 \, , T : 2 \rangle \xrightarrow{\tau} \\ & s_2 : \langle x : 1 \, , T : 2 \rangle \xrightarrow{\alpha} \; s_3 : \langle x : -1 \, , T : 4 \rangle \xrightarrow{\tau} \\ & \cdots \end{array}$$

corresponds to sampling the full behavior as shown in Fig. 15.

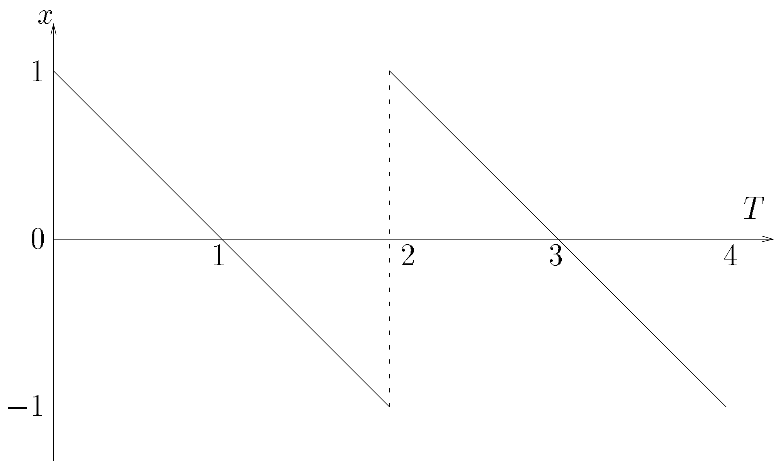


Figure 14: Full behavior of hybrid system ϕ_1.

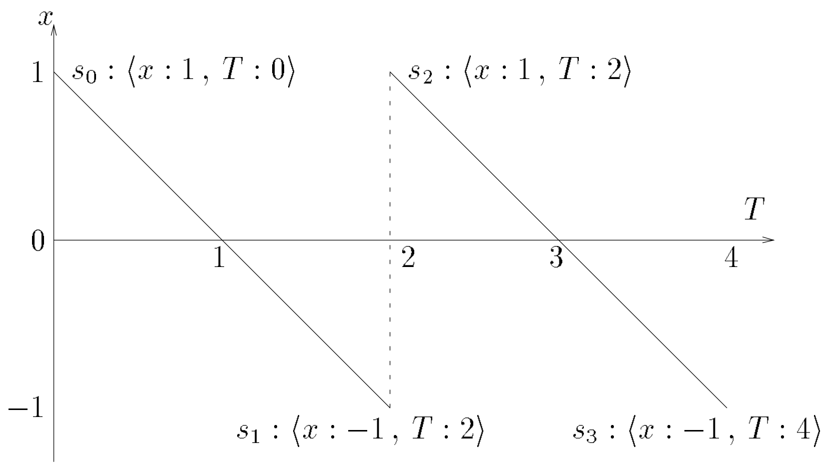


Figure 15: Sampling computation σ_1.

A more frequent sampling leads to the computation

$$\begin{array}{lllll}
\sigma_2: & s_0 : \langle x : \ \ 1\,,\, T : 0\rangle & \xrightarrow{\alpha} & s_1 : \langle x : \ \ 0\,,\, T : 1\rangle & \xrightarrow{\alpha} \\
 & s_2 : \langle x : -1\,,\, T : 2\rangle & \xrightarrow{\tau} & s_3 : \langle x : \ \ 1\,,\, T : 2\rangle & \xrightarrow{\alpha} \\
 & s_4 : \langle x : \ \ 0\,,\, T : 3\rangle & \xrightarrow{\alpha} & s_5 : \langle x : -1\,,\, T : 4\rangle & \xrightarrow{\tau} \\
 & \cdots & & &
\end{array}$$

whose sampling points are shown in Fig. 16.

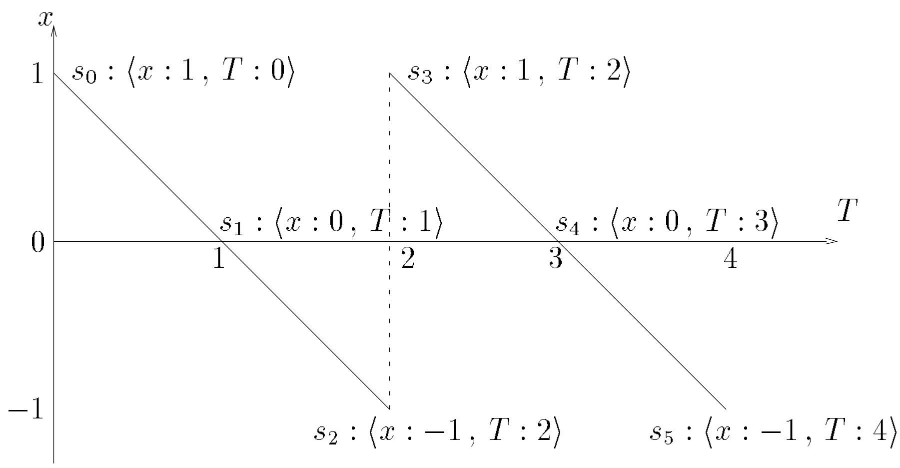


Figure 16: Sampling computation σ_2.

In comparison, consider phase transition system Φ_2 which is identical to Φ_1 in all components, except for $\mathcal{I}$, which is given by

$$\text{Important events } \mathcal{I}_2 \quad : \quad \{x \leq -1,\ x = 0\}$$

Thus, system Φ_2 considers the assertion $x = 0$ to be an important event in addition to $x \leq -1$, which is the enabling condition of transition τ. The situation sequence σ_2 is a sampling computation of Φ_2 as well. However, the sequence σ_1 is not. Informally, this is because σ_1 fails to observe the (infinitely many) points at which x becomes 0. Formally, $s_1 : \langle x : -1\,,\, T : 2\rangle$ is no longer an activity successor of $s_0 : \langle x : 1\,,\, T : 0\rangle$ because the evolution from s_0 to s_1 did not respect the condition that important assertions do not change their truth value in the middle of a continuous step.

Super-Dense Computations

In addition to sampling computations, [MMP92] presents another class of computations, based on the notion of *hybrid traces*. Sampling computations, similar to computations of timed transition systems, have the signature $\mathsf{N} \mapsto \Sigma \times \mathsf{R}^+$, that is, each natural number $j = 0, 1, \ldots$ is mapped to a pair consisting of a state s_j and a real time stamp t_j, i.e., a situation.

In contrast, the other type of computations presented in [MMP92], to which we refer here as *super-dense computations*, have the signature $\mathsf{R}^+ \times \mathsf{N} \mapsto \Sigma$, that is, each pair $\langle t, i\rangle$, where $t \in \mathsf{R}^+$ and $i \in \mathsf{N}$, is mapped to a state $s \in \Sigma$. The pair $\langle t, i \rangle$ identifies a time stamp t and a step number i. The step numbers correspond to the transitions that are taken at time instant t.

For example, the (single) super-dense computation produced by phase transition system Φ_1 is given by a function $x(t, i)$ from $\mathsf{R}^+ \times \mathsf{N}$ to R defined as

$$x(t,i) = \begin{cases} 1 & \text{for} \quad t = 0 \quad \text{and} \quad i \geq 0 \\ 1 - t & \text{for} \quad 0 < t < 2 \quad \text{and} \quad i \geq 0 \\ -1 & \text{for} \quad t = 2 \quad \text{and} \quad i = 0 \\ 1 & \text{for} \quad t = 2 \quad \text{and} \quad i \geq 1 \\ 3 - t & \text{for} \quad 2 < t < 4 \quad \text{and} \quad i \geq 0 \\ -1 & \text{for} \quad t = 4 \quad \text{and} \quad i = 0 \\ 1 & \text{for} \quad t = 4 \quad \text{and} \quad i \geq 1 \\ & \vdots \end{cases}$$

An argument, offered in [MMP92], claims that the super-dense semantics provides a more precise representation of the behavior of hybrid systems than the semantics of sampling computations. The main criticism of sampling computations complains that some important events may fail to be observed, such as the event of x becoming 0, to which computation σ_1 is oblivious.

This problem of undersampling is solved here by the introduction of the important event component. Consequently, in this paper we continue to use the sampling-computation semantics. The advantages of the sampling semantics are that it is simpler than the super-dense semantics and conforms better with sequence based verification methods.

4.2 System Description by Hybrid Statecharts

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

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

The statechart in Fig. 17 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 with states *Cat.rest* and *Mouse.rest* active, and variables x_m and x_c set to the initial value X_0. The mouse proceeds immediately to the state of running, in which its 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

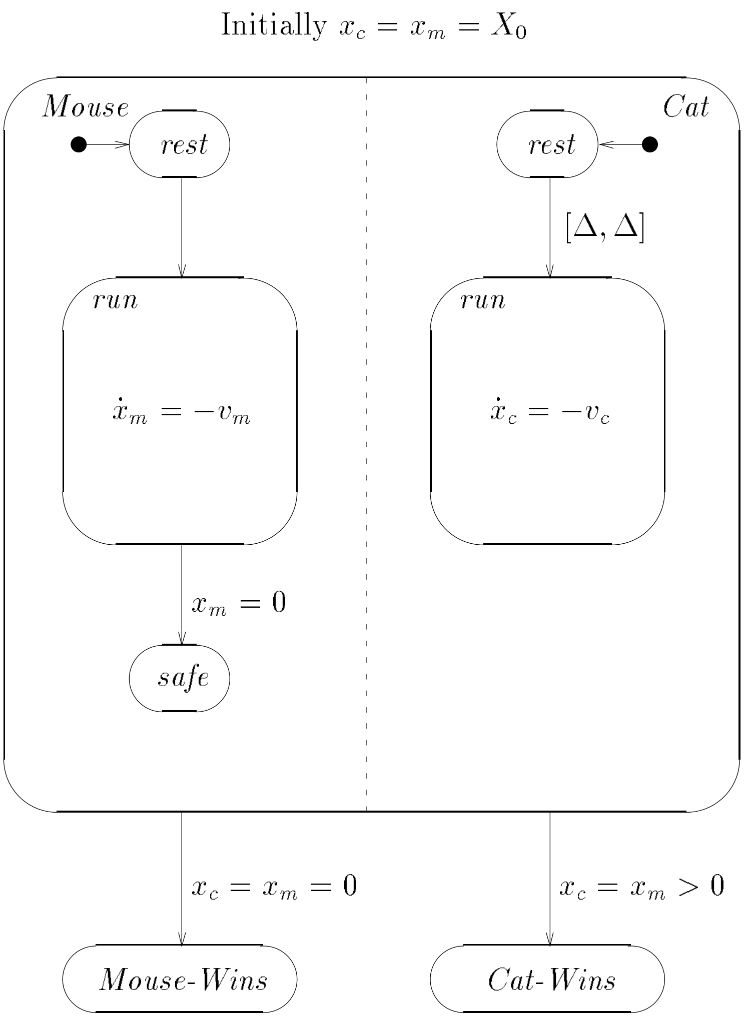


Figure 17: Specification of Cat and Mouse.

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 $x_c = x_m = 0$ 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 stop running and the system moves to state *Cat-Wins*. The compound conditions $x_c = x_m = 0$ and $x_c = x_m > 0$ stand for the conjunctions $x_c = x_m \wedge x_m = 0$ and $x_c = x_m \wedge x_m > 0$, respectively.

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

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 identify the phase transition system underlying the picture of Fig. 17.

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

- *Initial Condition:* Given by

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

- *Transitions:* Listed together with the transition relations associated with them.

$$\begin{array}{rcl}
M.rest\text{-}run & : & Mouse.rest \in \pi \ \wedge \ \pi' = \pi - \{Mouse.rest\} \cup \{Mouse.run\} \\
C.rest\text{-}run & : & Cat.rest \in \pi \ \wedge \ \pi' = \pi - \{Cat.rest\} \cup \{Cat.run\} \\
M.run\text{-}safe & : & Mouse.run \in \pi \ \wedge \ x_m = 0 \ \wedge \\
 & & \qquad \pi' = \pi - \{Mouse.run\} \cup \{Mouse.safe\} \\
M.win & : & (Active \cap \pi) \neq \phi \ \wedge \ x_c = x_m = 0 \ \wedge \ \pi' = \{Mouse\text{-}Wins\} \\
C.win & : & (Active \cap \pi) \neq \phi \ \wedge \ x_c = x_m > 0 \ \wedge \ \pi' = \{Cat\text{-}Wins\}
\end{array}$$

  The set *Active* stands for the set of basic states

$$\{Mouse.rest, Mouse.run, Mouse.safe, Cat.rest, Cat.run\}.$$

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

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

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

- *Important Events:* Given by

$$\mathcal{I} \ : \ \{x_m = 0, \ x_c = x_m = 0, \ x_c = x_m > 0\}$$

System Description by Textual Programs

It is possible to extend the simple programming language SPL to represent timed and hybrid systems as well. The resulting language is a subset of the language *Statext* introduced in [KP92b], which is shown there to have expressive power equal to that of hybrid statecharts.

We extend SPL by the following statements:

- **skip**

This statement serves as a filler. It does nothing and terminates in a single execution step.

- **idle**

  Like the *skip* statement, this statement does not change the data state. However, unlike the *skip* statement, the *idle* statement never terminates. The only way to get out of an *idle* statement is by *preemption*, which is another important construct of the extended language introduced later.
- **delay**$[l, u]$

  This statement delays for a time lying between l and u. Its semantics is given by a transition with time bounds $[l, u]$.
- *Selection*

  For statements S_1 and S_2,

  $S_1 \sqcup S_2$

is a *selection* statement. Its intended meaning is that, as a first step, one of S_1 and S_2, which is currently enabled, is selected and the first step in the selected statement is executed. Subsequent steps proceed to execute the rest of the selected substatement. If both S_1 and S_2 are enabled, the selection is non-deterministic. If both S_1 and S_2 are currently disabled, then so is the selection statement.
- *Cooperation*

  For S_1 and S_2 statements,

  $S_1 \parallel S_2$

is a *cooperation* statement. It calls for parallel execution of S_1 and S_2. The cooperation statement terminates when both S_1 and S_2 have terminated.
- *Preemption*

  For statements S_1 and S_2,

  $S_1 \ \mathsf{U} \ S_2$,

is a *preemption* statement. Steps in the execution of this statement are either steps in the execution of S_1 taken forever or till S_1 terminates, or zero or more steps in the execution of S_1 followed by steps in the execution of S_2. Thus, the intended meaning of the *preemption* statement is

> Execute S_1 forever or until it terminates,
> or execute S_1 until a first step of S_2 is taken, and then continue to execute S_2.

As usual, if a transition τ in S_2 has been continuously enabled for u_τ time units, then τ must be taken and execution switches to S_2 before time can progress.
Consider, for example, the statement

$$\left[\begin{array}{l} \textbf{while } \mathrm{T} \textbf{ do} \\ \quad \left[\begin{array}{l} \textbf{delay } [3,3] \\ y := y+1 \end{array}\right] \end{array}\right] \quad \mathsf{U} \quad \textbf{await } y > 2$$

Assuming the *await* statement to be immediate (assigned maximal delay 0), this statement terminates as soon as y grows above 2, even though the *while* statement, when standing alone, does not terminate.
- *Differential Equations*

  Differential equations are also acceptable as statements of the extended language. A differential equation statement never terminates, and the only way to get out of it is by preemption, using the U construct. Statements consisting of differential equations are associated with activities, in contrast to all other statements which are associated with transitions. Thus, the statement

$$\ell: \ \dot{x} = e$$

gives rise to the activity

$$\alpha_\ell: \quad \ell \in \pi \ \rightarrow \ \dot{x} = e.$$

We refer to this activity as an *explicit activity* since it corresponds to an explicit statement in the program.

Besides the explicit activities, each continuous variable $x \in V_c$ also has an implicit *default activity* α_x^{off} which controls its continuous change when none of the explicit activities governing x is active. If $\ell_1, \ldots, \ell_m$ are all the statements giving rise to explicit activities for x, then its default activity is given by

$$\alpha_x^{\mathit{off}}: \quad \{\ell_1, \ldots, \ell_m\} \cap \pi = \phi \ \rightarrow \ \dot{x} = 0.$$

In addition to these new statements, we assign time bounds to each transition in the language.

We refer the reader to [KP92b] for a sampling-computation semantics of this extension of SPL.

As an example, consider the statement

$$\left(\begin{bmatrix} \ell_0: & \textbf{while } \mathrm{T} \textbf{ do} \\ & \begin{bmatrix} \ell_1: & \textbf{delay } [3,3] \\ \ell_2: & y := y+1 \end{bmatrix} \end{bmatrix} \ \sqcup \ m_0: \ \textbf{await } y > 2 \right) \quad ; \quad k: \ldots$$

The transition relation associated with m_0 is

$$\rho_{m_0}: \quad m_0 \in \pi \ \wedge \ y > 2 \ \wedge \ \pi' = \pi - \{m_0, \ell_0, \ell_1, \ell_2\} \cup \{k\}.$$

Thus, on executing m_0, control discards any locations within statement ℓ_0.

Cat and Mouse in Extended SPL

The top level of the *hybrid* SPL specification is

$$\mathit{Spec} :: \begin{bmatrix} & x_c, x_m: \ \textbf{real} \ \ \textbf{where } x_c = x_m = X_0 \\ [\mathit{Mouse} \parallel \mathit{Cat}] \ \sqcup & \begin{bmatrix} \textbf{await } x_c = x_m > 0 \ ; & \mathit{Cat\text{-}Wins}: \\ \bigsqcup & \\ \textbf{await } x_c = x_m = 0; & \mathit{Mouse\text{-}Wins}: \end{bmatrix} \end{bmatrix}$$

Mouse and *Cat* are processes defined as follows:

$$
\begin{array}{ll}
\mathit{Mouse} :: & \left[\begin{array}{l} \dot{x}_m = -v_m \\ \quad \mathsf{U} \\ \textbf{await } x_m = 0;\ \textbf{idle} \end{array} \right] \\[2ex]
\mathit{Cat} :: & \left[\ \textbf{delay}[\Delta, \Delta]; \quad \dot{x}_c = -v_c \ \right]
\end{array}
$$

The *idle* statement at the end of process *Mouse* corresponds to state *Mouse.safe* in the statechart.

The transitions associated with the statements of this extended SPL program are all immediate, except for the transition associated with the delay statement, which is assigned the time bounds $[\Delta, \Delta]$.

The Sharpness Condition

Requiring that systems be progressive still does not guarantee that every phase transition system has a computation. Consider, for example, a system described by the statechart of Fig. 18.

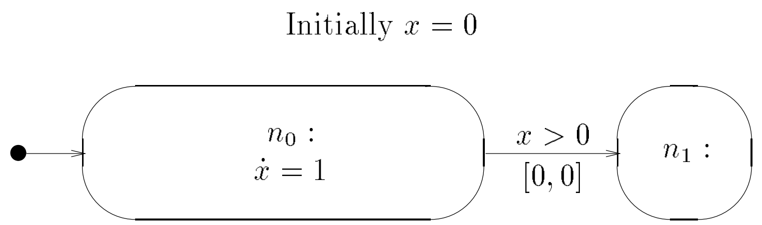


Figure 18: A system with no computations.

Any possible computation must start at the initial situation $s_0 : \langle \pi : n_0,\ x : 0,\ T : 0 \rangle$. Unfortunately, there is no way to proceed from this situation. The transition from n_0 to n_1 cannot be taken since x is not positive. Time cannot progress by even a small amount $\epsilon > 0$, because that would cause the transition from n_0 to n_1 to be continuously enabled for ϵ time units which is higher than its upper bound 0. It follows that this system has no computations.

Obviously, the problem can be traced to the nature of the condition $x > 0$ which, when x increases continuously, has no definite point in time at which the condition becomes true. To prevent such situations, we define a subset of formulas to which we refer as *sharp formulas* and which always have definite points at which they become true. A formula p is defined to be sharp if:

- p depends only on discrete variables, or
- p has the form $t_1 \leq t_2$ for some terms t_1 and t_2, or
- p has the form $q \wedge r$ or $q \vee r$ for some sharp formulas q and r.

A phase transition system is called *sharp* if each immediate transition τ is a member of a set of immediate transitions $\{\tau_1, \ldots, \tau_k\}$ (possibly consisting of τ_1 alone) such that the disjunction

$$En(\tau_1) \vee \cdots \vee En(\tau_k)$$

is equivalent to a sharp formula. This ensures that when τ becomes enabled as a result of a continuous evolution, then the disjunction $En(\tau_1) \vee \cdots \vee En(\tau_k)$ which is sharp also becomes true. It follows that there is a first moment at which the disjunction becomes true and one of the transitions in the set $\{\tau_1, \ldots, \tau_k\}$ can be immediately taken.

Obviously, the system of Fig. 18 is not sharp and this explains the problems associated with it. On the other hand, consider the *Cat and Mouse* system of Fig. 17. The enabling condition of transition $C.win$ is $(Active \cap \pi) \neq \phi \wedge x_c = x_m \wedge x_m > 0$, which is not a sharp formula. However, when we consider the larger set of immediate transitions $\{C.win, M.win\}$, the disjunction of the enabling conditions of its members we obtain

$$(Active \cap \pi) \neq \phi \;\wedge\; x_c = x_m \;\wedge\; x_m \geq 0,$$

which is a sharp formula. Consequently, the *Cat and Mouse* system is sharp.

From now on, we will only consider sharp phase transition systems.

4.3 Requirement Specification Languages

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

For example, to specify that the mouse will always escape the cat, for the system of Fig. 17, we can write the invariance formula

$$\Box(Cat.run \wedge (x_c = x_m) \quad \rightarrow \quad x_m = 0),$$

where we use names of states in a statechart as control predicates. Of course, such a property will not be valid over all computations of the cat and mouse system, unless some relation is established among the problem parameters X_0, Δ, v_c, and v_m. Indeed, a sufficient condition for this property to be valid is:

$$\frac{X_0}{v_m} \;<\; \Delta + \frac{X_0}{v_c}.$$

4.4 Verification of Age Formulas

Here we only consider verification of TL_Γ formulas over hybrid systems. The age axioms and rule T-INIT, presented for the real-time model, hold here as well. The major difference is in the verification conditions.

Verification Conditions for Hybrid Systems

The verification condition $\{p\}\,\tau\,\{q\}_T$ remains unchanged. However, instead of using the verification condition $\{p\}\,tick\,\{q\}$, we define a new condition

$$\{p\}\ cont\ \{q\}$$

which is intended to ensure that every continuous phase leads from a p-situation to a q-situation.

To formulate the verification condition over continuous steps, we consider an evolution from a situation that can be described as $\langle V, T\rangle$ to the situation $\langle V', T'\rangle$, assuming that $T' > T$.

An *activity selection* is a mapping $g : V_c \mapsto \mathcal{A}$, assigning to each continuous variable $x \in V_c$ an activity $g(x)$ in its governing set. Assume that the activity selected by g for each continuous state variable $x \in V_c$ is

$$p_x^g \quad \rightarrow \quad \dot{x} = e^g.$$

Let $F^g = \{f_x^g(t)\}$ be a set of functions, one for each continuous variable $x \in V_c$, such that

- $f_x^g(T) = x$, for every $x \in V$.
- The equation

  $$\dot{x}_x^g(t) \quad = \quad e^g(F^g)$$

  is satisfied in the range $t \in [T, T']$, where $e^g(F^g)$ is obtained from e^g by replacing each occurrence of $y \in V_c$ by $f_y^g(t)$.

We assume that we know how to express the functions f_x^g in a closed form, referring to x, T, and t. For example, if g selects for $x \in V_c$ the activity $\dot{x} = 2$, then $f_x^g(t)$ is given by

$$f_x^g(t) \quad = \quad x + 2 \cdot (t - T).$$

The Condition

With each possible activity selection function g, we associate a verification condition $\{p\}\,g\,\{q\}$, which is given by

$$\rho_{cont}^g \ \wedge\ p \quad \rightarrow \quad q',$$

where the relation ρ_{cont}^g stands for

$$\bigwedge_{x \in V_d} x' = x \ \wedge \ \bigwedge_{x \in V_c} f_x^g(T') = x' \ \wedge \ \bigwedge_{x \in V_c} p_x^g \ \wedge \ T' > T \ \wedge \ \bigwedge_{\tau \in \mathcal{T}} \Big(\Gamma'(En(\tau)) \leq u_\tau\Big) \ \wedge$$

$$\bigwedge_{\varphi \in \mathcal{I}} \left(\begin{array}{l} \Big(\forall t : (T < t \leq T') : \quad \varphi^g(t)\Big) \ \wedge \ \Gamma'(\varphi) = \Gamma(\varphi) + T' - T \\ \qquad \vee \\ \Big(\forall t : (T \leq t < T') : \quad \varphi^g(t)\Big) \ \wedge \ \neg\varphi' \ \wedge \ \Gamma'(\varphi) = 0 \\ \qquad \vee \\ \Big(\forall t : (T < t < T') : \neg\varphi^g(t)\Big) \ \wedge \ (\neg\varphi \vee \neg\varphi') \ \wedge \ \Gamma'(\varphi) = 0 \end{array} \right)$$

The first conjunct of the formula states that all discrete variables are not changed in a continuous step. The next conjunct states that the value of $f_x^g(T')$ agrees with x'. The third conjunct requires that the activation condition p_x^g holds at the pre-state V. The fourth conjunct requires that time progresses by a positive amount. The last conjunct in the first line ensures that the progress of time cannot cause any transition to remain continuously enabled longer than is allowed by its upper bound u_τ.

The next line requires that every important assertion is either true throughout the full evolution interval, or is false throughout the full evolution interval, except possibly at one of its boundaries. The formula $\varphi^g(t)$ is obtained from φ by replacing each occurrence of $y \in V_c$ by $f_y^g(t)$.

This line also defines the value of $\Gamma'(r)$ after a continuous phase takes place. In order to ensure that this definition actually measures the age of an assertion, we restrict the use of the expression $\Gamma(r)$ to assertions r that appear in the important event set $\mathcal{I}$.

In comparison, When a transition is taken, the value of $\Gamma'(r)$ is determined by the following axiom:

$$\Gamma'(r) \quad = \quad \textbf{if } r' \textbf{ then } \Gamma(r) \textbf{ else } 0.$$

Consider, for example, the condition $\{-1 \leq x \leq 1\}\, g\, \{-1 \leq x \leq 1\}$ for system Φ_1. Since there is only one activity, there is only one activity selection g, i.e., the one that selects this activity. The evolution function f_x^g is given by $f_x^g(t) = x - (t - T)$. Consequently, the verification condition is given (after some simplifications) by:

$$\underbrace{\left[\begin{array}{ll} & \pi' = \pi \ \wedge \ x' = x - (T' - T) \ \wedge \ T' > T \\ \wedge & \Gamma'(x \leq -1) \leq 0 \\ \wedge & \left(\begin{array}{ll} & (\forall t : (T < t \leq T') : x - (t - T) \leq -1) \\ \wedge & \Gamma'(x \leq -1) = \Gamma(x \leq -1) + T' - T \\ & \qquad \vee \\ & (\forall t : (T \leq t < T') : x - (t - T) \leq -1) \\ \wedge & x' > -1 \wedge \Gamma'(x \leq -1) = 0 \\ & \qquad \vee \\ & (\forall t : (T < t < T') : x - (t - T) > -1) \\ \wedge & (x > -1 \vee x' > -1) \wedge \Gamma'(x \leq -1) = 0 \end{array}\right) \end{array}\right]}_{\rho^g_{cont}} \wedge \underbrace{-1 \leq x \leq 1}_{p} \quad \rightarrow \quad \underbrace{-1 \leq x' \leq 1}_{q}$$

First, we show that each of the three possibilities allowed by the last conjunct of the formula implies $x' \geq -1$.

- The case $(\forall t : (T \leq t < T') : x - (t - T) \leq -1)$ is impossible since, as $\Gamma(x \leq -1) \geq 0$ and $T' > T$, this leads to $\Gamma'(x \leq -1) > 0$ which violates the conjunct in the second line of the formula.

- The case $(\forall t : (T \leq t < T') : x - (t - T) \leq -1)$ explicitly requires $x' > -1$.

- The case $(\forall t : (T < t < T') : x - (t - T) > -1)$ implies that $x - (t - T) > -1$ holds for all $t \in (T, T')$. It follows that, in the limit of t approaching T', $x' = x - (T' - T) \geq -1$.

For the other inequality, from $T' > T$ it follows that $x' = x - (T' - T) < x \leq 1$ and therefore $x' \leq 1$.

Finally, we define the verification condition over a continuous step to be

$$\{p\}\, cont\, \{q\} : \quad \bigwedge_g \{p\}\, g\, \{q\},$$

where the conjunction of the individual conditions $\{p\}\, g\, \{q\}$ is taken over all possible activity selection functions g.

Rules for Waiting-For and Invariance Formulas

Having defined the two verification conditions, we can immediately formulate two rules for proving waiting-for and invariance formulas over hybrid systems.

$$\begin{array}{|llccc|}
\hline
\text{H-WAIT} & \text{W1.} & p & \rightarrow & q \vee \varphi \\
 & \text{W2.} & \{\varphi\} & \mathcal{T} & \{q \vee \varphi\}_{\mathcal{T}} \\
 & \text{W3.} & \{\varphi\} & cont & \{q \vee \varphi\} \\
\cline{2-5}
 & & p & \Rightarrow & \varphi\, \mathcal{W}\, q \\
\hline
\end{array}
\qquad
\begin{array}{|lll|}
\hline
\text{H-INV} & \text{I1.} & \Theta_T \rightarrow \varphi \\
 & \text{I2.} & \{\varphi\}\, \mathcal{T}\, \{\varphi\}_{\mathcal{T}} \\
 & \text{I3.} & \{\varphi\}\, cont\, \{\varphi\} \\
\cline{2-3}
 & & \square\, \varphi \\
\hline
\end{array}$$

We will illustrate the use of these rules on several examples.

For the most trivial example, consider system Φ_1. We will prove that the invariant $\square(-1 \leq x \leq 1)$ is valid over all sampling computations of this system. We use rule H-INV with $\varphi : -1 \leq x \leq 1$.

Premise I1 assumes the form

$$\underbrace{x = 1 \ \wedge \ T = 0}_{\Theta_T} \quad \rightarrow \quad \underbrace{-1 \leq x \leq 1}_{\varphi},$$

which is obviously valid.

Premise I2 assumes the form

$$\underbrace{\cdots \wedge (x' = 1) \wedge \cdots}_{\rho^*_{\mathcal{T}}} \ \wedge \ \underbrace{\cdots}_{\varphi} \quad \rightarrow \quad \underbrace{-1 \leq x' \leq 1}_{\varphi'}$$

which is also obviously valid.

Premise I3 requires showing that $-1 \leq x \leq 1$ is preserved under a continuous step. However, this has been verified above.

Proof of a Hybrid Version of Program ANY-Y

In Fig. 19, we present an extended SPL program ANY-Y$_H$ that can be viewed as a hybrid version of program ANY-Y.

In this program, process P_2 represents a continuous component that lets x grow linearly from 0 until it reaches a value $x \geq 1$. At that point, statement m_1 intervenes and shuts off the continuous process. Process P_1 is very similar to process P_1 in program ANY-Y. It loops, incrementing y, as long as $x < 1$. Once process P_1 detects that $x \geq 1$, it terminates. Note that x never exceeds 1, so that all references to $x \geq 1$ can be replaced by $x = 1$.

$$y\text{: }\mathbf{integer\ where}\ y = 0$$
$$x\text{: }\mathbf{real\ \ where}\ x = 0$$

$$\left[\begin{array}{l} \ell_0 : \ \ \mathbf{while}\ x < 1\ \mathbf{do} \\ \qquad \left[\ell_1 :\ y := y+1\right] \\ \ell_2 : \end{array}\right] \quad \| \quad \left[\begin{array}{l} [m_0 :\ \ \dot{x} = 0.2] \quad \mathsf{U} \quad [m_1 :\ \ \mathbf{await}\ x \geq 1] \\ m_2 : \end{array}\right]$$

$$- \ \ P_1 \ \ - \qquad\qquad\qquad\qquad - \ \ P_2 \ \ -$$

Figure 19: Program ANY-Y$_H$: A hybrid textual program.

The time bounds associated with this program identify transition m_1 as immediate (i.e., time bounds [0,0]), and assign uniform time bounds $[1, 5]$ to all other transitions.

To fully comprehend the behavior of this program, we present two possible computations of ANY-Y$_H$. The first computation attempts to maximize the value of y on termination.

$$\begin{array}{llll}
\langle \pi : \{\ell_0, m_0, m_1\}, x : 0.0, y : 0, T : 0\rangle & \xrightarrow{cont} & \langle \pi : \{\ell_0, m_0, m_1\}, x : 0.2, y : 0, T : 1\rangle & \xrightarrow{\ell_0} \\
\langle \pi : \{\ell_1, m_0, m_1\}, x : 0.2, y : 0, T : 1\rangle & \xrightarrow{cont} & \langle \pi : \{\ell_1, m_0, m_1\}, x : 0.4, y : 0, T : 2\rangle & \xrightarrow{\ell_1} \\
\langle \pi : \{\ell_0, m_0, m_1\}, x : 0.4, y : 1, T : 2\rangle & \xrightarrow{cont} & \langle \pi : \{\ell_0, m_0, m_1\}, x : 0.6, y : 1, T : 3\rangle & \xrightarrow{\ell_0} \\
\langle \pi : \{\ell_1, m_0, m_1\}, x : 0.6, y : 1, T : 3\rangle & \xrightarrow{cont} & \langle \pi : \{\ell_1, m_0, m_1\}, x : 0.8, y : 1, T : 4\rangle & \xrightarrow{\ell_1} \\
\langle \pi : \{\ell_0, m_0, m_1\}, x : 0.8, y : 2, T : 4\rangle & \xrightarrow{cont} & \langle \pi : \{\ell_0, m_0, m_1\}, x : 1.0, y : 2, T : 5\rangle & \xrightarrow{m_1} \\
\langle \pi : \{\ell_0, m_2\} \quad\ \ , x : 1.0, y : 2, T : 5\rangle & \xrightarrow{\ell_0} & \langle \pi : \{\ell_2, m_2\} \quad\ \ , x : 1.0, y : 2, T : 5\rangle & \xrightarrow{cont} \\
\langle \pi : \{\ell_2, m_2\} \quad\ \ , x : 1.0, y : 2, T : 6\rangle & \xrightarrow{cont} & \cdots &
\end{array}$$

The second computation attempts to minimize the value of y on termination.

$$\begin{array}{llll}
\langle \pi : \{\ell_0, m_0, m_1\}, x : 0.0, y : 0, T : 0\rangle & \xrightarrow{cont} & \langle \pi : \{\ell_0, m_0, m_1\}, x : 1.0, y : 0, T : 5\rangle & \xrightarrow{m_1} \\
\langle \pi : \{\ell_0, m_2\} \quad\ \ , x : 1.0, y : 0, T : 5\rangle & \xrightarrow{\ell_0} & \langle \pi : \{\ell_2, m_2\} \quad\ \ , x : 1.0, y : 0, T : 5\rangle & \xrightarrow{cont} \\
\langle \pi : \{\ell_2, m_2\} \quad\ \ , x : 1.0, y : 0, T : 6\rangle & \xrightarrow{cont} & \cdots &
\end{array}$$

We will now prove that program ANY-Y$_H$ also terminates within 15 time units. Using rule T-INIT, it is sufficient to prove

$$\Theta_T \quad \Rightarrow \quad (T \leq 15)\, \mathcal{W}\, (at\_\ell_2 \wedge at\_m_2).$$

The proof uses rule H-WAIT and monotonicity. The main constituents of the proof are presented in Fig. 20. It is not difficult to check that this diagram is valid with respect to Θ_T, $T \leq 15$, and $at\_\ell_2 \wedge at\_m_2$. The only new element is checking that assertion φ_0 is preserved under a continuous step. The activity set for this program consists of the two activities:

$$\begin{array}{rrcl}
\alpha_1 : & at_m_0 & \rightarrow & \dot{x} = 0.2 \\
\alpha_2 : & \neg at_m_0 & \rightarrow & \dot{x} = 0.
\end{array}$$

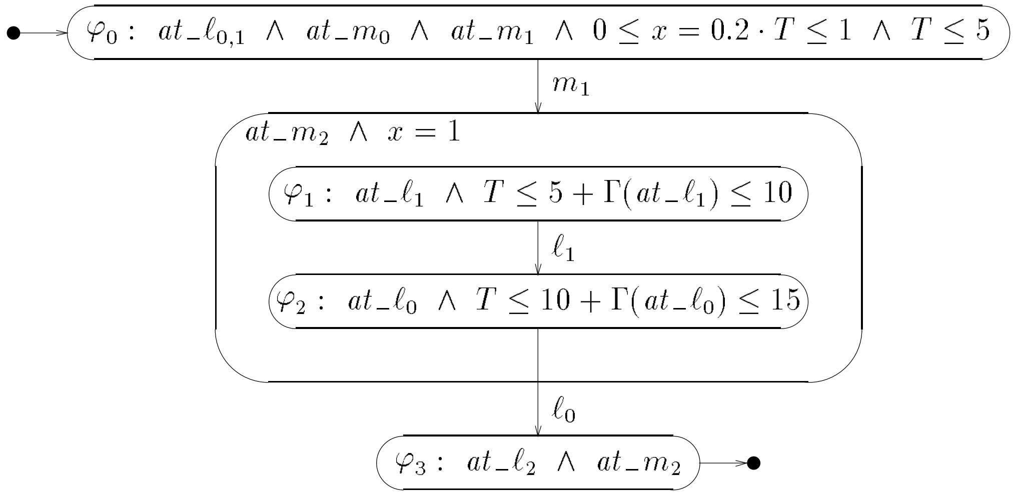


Figure 20: A hybrid waiting-for proof diagram.

The only activity selection function g relevant for states satisfying φ_0 is the one that picks α_1 for x, i.e., $g(x) = \alpha_1$. For this choice of g, the evolution of x is given by $f_x^g(t) = x + 0.2 \cdot (t - T)$. Consequently, the appropriate verification condition (after some simplifications) is

$$
\begin{array}{ll}
& \left.\begin{array}{l} \pi' = \pi \ \wedge \ x' = x + 0.2 \cdot (T' - T) \ \wedge \ T' > T \ \wedge \ \cdots \\ \wedge \ \ (\forall t : T \le t < T' : x + 0.2 \cdot (t - T) < 1) \end{array}\right\} \rho^g_{cont} \\
\wedge & \underbrace{at\_\ell_{0,1} \ \wedge \ at\_m_0 \ \wedge \ at\_m_1 \ \wedge \ 0 \le x = 0.2 \cdot T \le 1 \ \wedge \ T \le 5}_{\varphi_0} \\
\rightarrow & \\
& \underbrace{\cdots}_{q'} \vee \underbrace{(at\_\ell_{0,1})' \ \wedge \ (at\_m_0)' \ \wedge \ (at\_m_1)' \ \wedge \ 0 \le x' = 0.2 \cdot T' \le 1 \ \wedge \ T' \le 5}_{\varphi_0'}
\end{array}
$$

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

- $(at\_\ell_{0,1})' \wedge (at\_m_0)' \wedge (at\_m_1)'$ follows from $\pi' = \pi$.
- $x' = 0.2 \cdot T'$ follows from $x' = x + 0.2 \cdot (T' - T)$ and $x = 0.2 \cdot T$.
- $0 \le x'$ follows from $0 \le T < T'$ and $x' = 0.2 \cdot T'$.
- Taking the limit over $(\forall t : T \le t < T' : x + 0.2 \cdot (t - T) < 1)$ as t tends to T', yields $x' = x + 0.2 \cdot (T' - T) \le 1$.
- $T' \le 5$ follows from $0.2 \cdot T' \le 1$.

This establishes that program ANY-Y$_H$ terminates within 15 time units.

Verifying a Property of the Cat and Mouse System

Consider the property that, under the assumption

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

all computations of the Cat and Mouse system satisfy

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

In Fig. 21, we present a proof diagram of this invariance property. In this diagram we

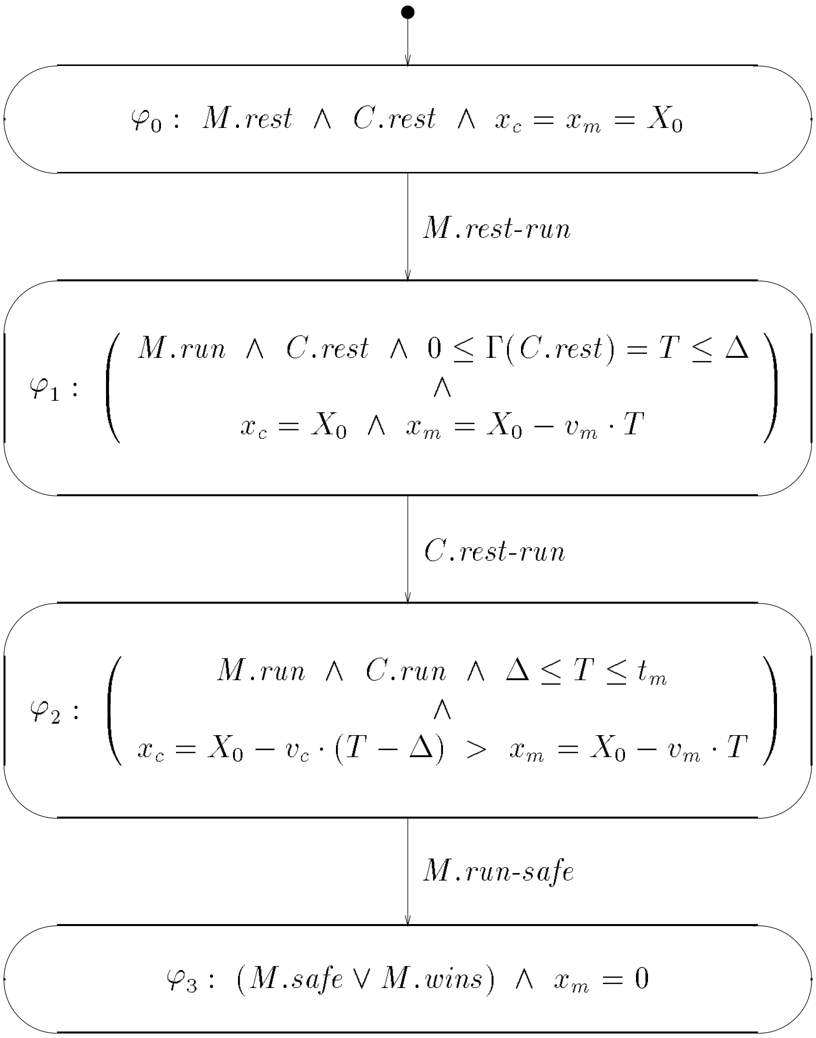


Figure 21: A hybrid invariance proof diagram.

use control assertions denoting that certain basic states are contained in π. For example, $C.run$ stands for $Cat.run \in \pi$. We also use t_m for $\frac{X_0}{v_m}$, the time it takes the mouse to run the distance X_0.

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

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

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

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

From inequality (1), we can obtain

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

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

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

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

It remains to show that

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

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

Implication (4) is obviously valid. To check implication (5), we observe that both φ_0 and φ_1 imply $\neg C.run$, φ_2 implies $x_c > x_m$, and φ_3 implies $x_m = 0$.

This shows that under the assumption (1), property

$$\square(Cat.run \wedge (x_c = x_m) \;\;\rightarrow\;\; x_m = 0)$$

is valid for the Cat and Mouse system.

4.5 The Gas Burner Example

We conclude with an example of a Gas Burner System, presented in [CHR92]. Consider the timed statechart presented in Fig. 22. This statechart represents a Gas Burner system that has three states: s_0, s_1, and s_2. There is a boolean state variable *Leak* whose value represents whether the system is currently leaking. For clarity, we have labeled each state with the value of *Leak* at the states. However, this labeling has no semantic meaning.

The verification problem posed in [CHR92] can be formulated as follows.
Assuming

1. A continuous leaking period cannot extend beyond 1 time unit.

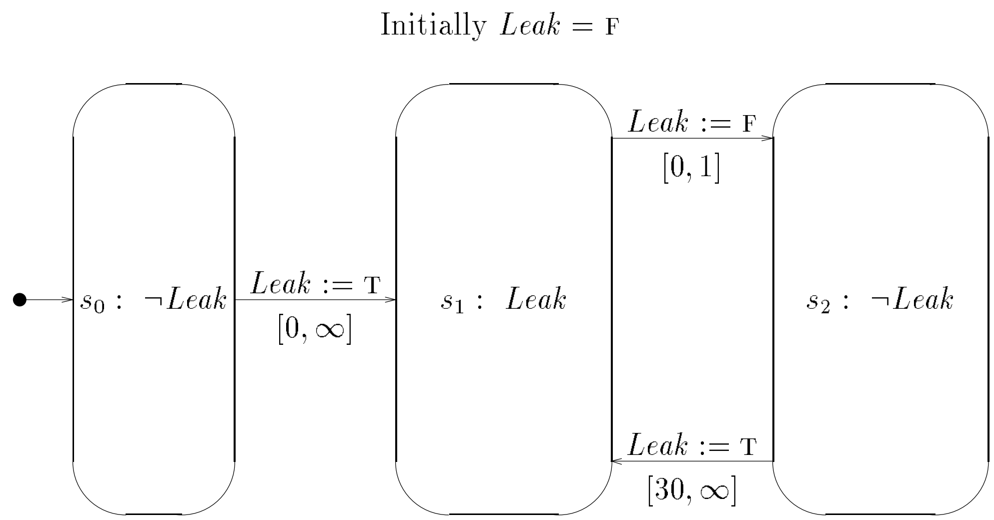


Figure 22: GAS-BURNER: A gas burner system.

2. Two disjoint leaking periods are separated by a non-leaking period extending for at least 30 time units.

Prove:

- *Safety-Critical Requirement:* In any interval longer than 60, the *accumulated* leaking time is at most 5% of the interval length.

Obviously, the timed statechart of Fig. 22 satisfies assumptions 1 and 2. The only leaking state is s_1 and it is clear that the system cannot stay continuously in s_1 for more than 1 time unit and that, between two consecutive (but disjoint) visits to s_1, the system stays at the non-leaking state s_2 for at least 30 time units.

However, the property to be proved uses the notion of *accumulated time* in which some assertion, such as *Leak*, holds. This cannot be expressed directly in TL_Γ. The calculus of durations, introduced in [CHR92], has a special *duration* operator $\int p$ that measures the accumulated time p holds. Later, we will briefly consider an extension of TL_Γ which adopts the duration operator [KP92a].

To handle this problem without extending the logic, we represent the Gas Burner system as a hybrid system, using auxiliary variables that measure the total time of an interval and the accumulated time in which variable *Leak* has been true. For simplicity, we first consider the safety-critical requirement only for *initial intervals*, i.e., intervals starting at $T = 0$. The extension of the method to arbitrary intervals is then straightforward and will be discussed later.

Consider the hybrid statechart of Fig. 23. The system presented there employs three auxiliary continuous variables as follows:

- Variable x measures the duration of time in each of the states s_0, s_1, and s_2. It is reset to 0 on entry to each of these states.

- Variable y measures the accumulated leaking time. It grows linearly in state s_2, and stays constant in any of the other states.

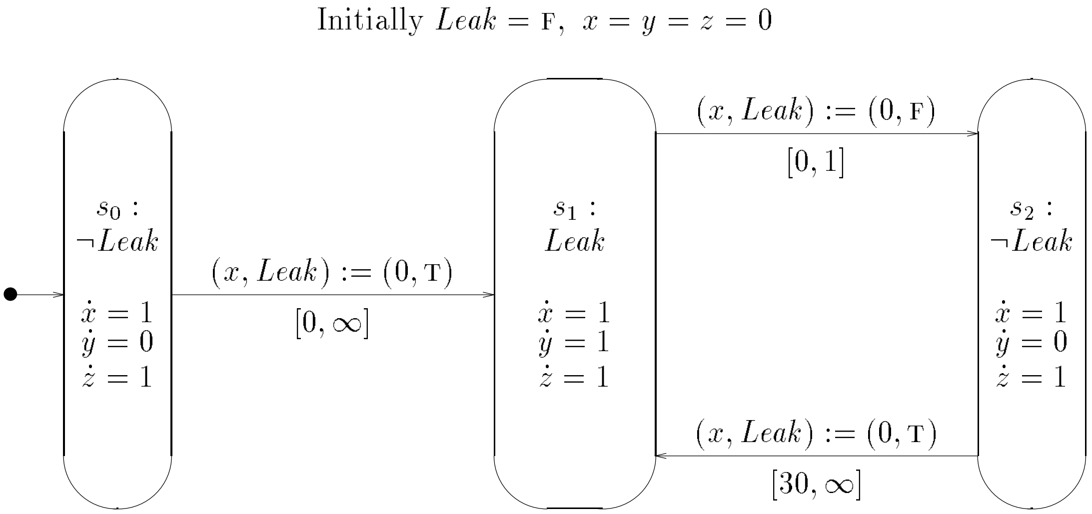


Figure 23: H-GAS: The gas burner as a hybrid system.

- Variable z measures the total elapsed time.

With these variables, we can write the requirement that the accumulated leak time does not exceed 5% of the elapsed time as $y \leq 0.05 \cdot z$ or, equivalently, as $20 \cdot y \leq z$.

Consequently, to verify that the original timed system of Fig. 22 maintains the safety-critical requirement for initial intervals, it is sufficient to prove that all computations of the hybrid system of Fig. 23 satisfy the invariance property

$$\Box\Big(z \geq 60 \quad\rightarrow\quad 20 \cdot y \leq z\Big).$$

This is the first example in which the invention of the necessary auxiliary invariants is not immediately obvious. Therefore, we will spend some time on their derivation. We try to find a relation that continuously holds between y and z and that implies the requirement

$$z \geq 60 \quad\rightarrow\quad 20 \cdot y \leq z. \tag{6}$$

Consider a finite prefix of a computation. Let v_1 denote the number of times the leaking state s_1 is visited in this prefix. Since on each visit variable y grows by at most 1 time unit, we have

$$y \leq v_1$$

at the end of the prefix. On the same prefix, variable z can be bounded from below by the sum of the accumulated time spent at s_1 and the accumulated time spent at s_2, ignoring the time spent at s_0 which can be arbitrarily short. The accumulated time spent at s_1 equals y. Since between two consecutive visits to s_1 the computation visits s_2, the number of visits to s_2 is at least $v_1 - 1$, and each of these visits lasts at least 30 time units. We thus obtain

$$z \geq 30 \cdot (v_1 - 1) + y \geq 31 \cdot y - 30,$$

where the last inequality is obtained by replacing v_1 by the smaller or equal value y. This leads to:

$$z \;\geq\; 31 \cdot y - 30. \tag{7}$$

We will show that this relation implies requirement (6), that is

$$z \geq 31 \cdot y - 30 \quad \rightarrow \quad (z \geq 60 \;\rightarrow\; 20 \cdot y \leq z),$$

or, equivalently,

$$z \geq 31 \cdot y - 30 \;\wedge\; z \geq 60 \quad \rightarrow \quad 20 \cdot y \leq z.$$

By $z \geq 60$, which can be written as $30 \leq \frac{1}{2} \cdot z$, we can replace the value 30 in $z \geq 31 \cdot y - 30$ by the bigger or equal value $\frac{1}{2} \cdot z$ and obtain

$$z \geq 31 \cdot y - \frac{1}{2} \cdot z,$$

leading to

$$\begin{array}{rcl} \frac{3}{2} \cdot z & \geq & 31 \cdot y, \\ z & \geq & \frac{62}{3} \cdot y \;\; > \;\; 20 \cdot y. \end{array}$$

We therefore start with the assumption that the inequality $z \geq 31 \cdot y - 30$ holds at all states in the computation. Working backwards, we can identify what versions of this invariant should hold on every visit to each of the states s_0, s_1, and s_2. This leads to the proof diagram presented in Fig. 24. Transitions in this diagram are identified by the names of the states in system H-GAS from which they exit. To facilitate the reading of the diagram, edges entering a node are annotated by an assertion that holds whenever this node is entered. Thus, it can be shown that $z \geq 31 \cdot y$ (which is the same as $(z - x) \geq 31 \cdot (y - x)$ since $x = 0$ on entry) holds on entering node n_1 from either n_0 or n_2. Since x, y, and z all grow at the same linear rate within state s_1 (corresponding to node n_1), the differences $z - x$ and $y - x$ maintain the values they had on entry. This explains why $(z - x) \geq 31 \cdot (y - x)$ is maintained within n_1. On exit from n_1 to n_2, $x \leq 1$, therefore, $(z - x) \geq 31 \cdot (y - x)$ implies $z \geq 31 \cdot y - 30 \cdot x \geq 31 \cdot y - 30$ on entry to n_2. Since, within n_2 both x and z grow at the same rate, while y remains the same, $(z - x) \geq 31 \cdot y - 30$ is maintained.

It is not difficult to see that the initial condition implies φ_0 and that each of φ_0, φ_1, or φ_2, implies $z \geq 31 \cdot y - 30$. Consequently, $z \geq 31 \cdot y - 30$ holds over all computations, establishing the validity of

$$\Box\Big(z \geq 60 \;\;\rightarrow\;\; 20 \cdot y \leq z\Big).$$

To generalize this analysis to arbitrary (not necessarily initial) intervals, we can add a transition that nondeterministically resets the values of y and z to 0. This will start the measurements corresponding to an interval at an arbitrary point in time. In fact, the proof diagram of Fig. 24 is also valid for this system. It can be checked that all the assertions in this diagram are preserved under simultaneous reset of y and z to 0. To ensure that this new transition is self-disabling, we make it enabled only when $y > 0$.

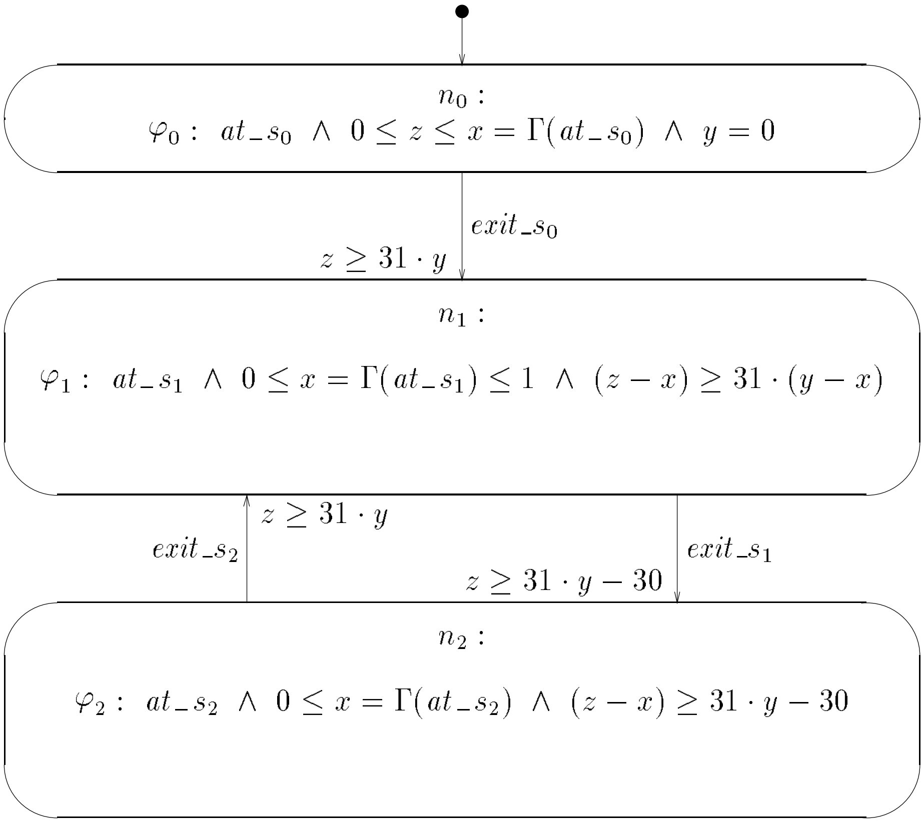


Figure 24: An invariance proof diagram for the gas burner.

Proof by an Extended Version of $\mathbf{TL}_\Gamma$

The previous proof transformed the Gas Burner problem into a hybrid system and verified the required property in the hybrid model. We will now consider an alternative approach, which does not modify the given system but uses a stronger logic. Since the original Gas Burner system as presented in Fig. 22 is a TTS rather than a hybrid system, we return to the framework of timed transition systems.

As is shown in [KP92a], it is possible to extend TL_Γ further by adding the *duration function* $\int p$, which measures the accumulated time in which p has been true up to the present. We denote the extended logic by $\mathrm{TL}_{\Gamma\int}$.

Very few extensions are needed as a result of this addition. The first extension is axiom DURATION-RANGE, which bounds the range of the duration function and also relates it to the age function.

$$\text{DURATION-RANGE}: \quad 0 \leq \Gamma(\psi) \leq \textstyle\int \psi \leq T \qquad \text{for every formula } \psi$$

Since duration expressions can appear in assertions, it is also necessary to define the primed version of a duration expression $\int r$, denoted $(\int r)'$ for some assertion r. This is given by

$$(\int r)' \;=\; \textbf{if } r \textbf{ then } \int r + T' - T \textbf{ else } \int r.$$

This definition states that, if r holds at s, then the value of $\int r$ at $\langle s', t'\rangle$ is its value at $\langle s, t\rangle$ plus the time difference $t' - t$. Otherwise, it retains the same value it has at $\langle s, t\rangle$. Since we are back in the timed transition system model, it is sufficient to check the value of r at $\langle s, t\rangle$.

Using the logic $\mathrm{TL}_{\Gamma\int}$, we can express the safety-critical requirement of system GAS-BURNER of Fig. 22 by the formula

$$\Box\Big(T \geq 60 \quad\rightarrow\quad 20 \cdot \int Leak \leq T\Big)$$

We can prove this property using rule T-INV and monotonicity. The auxiliary invariant assertion used is inspired by the proof diagram of Fig. 24 and is given by

$$\begin{array}{l}
at_s_0 \;\wedge\; \int Leak = 0 \\
\quad\vee \\
at_s_1 \;\wedge\; (T - \Gamma(at_s_1)) \geq 31 \cdot (\int Leak - \Gamma(at_s_1)) \\
\quad\vee \\
at_s_2 \;\wedge\; (T - \Gamma(at_s_2)) \geq 31 \cdot \int Leak - 30.
\end{array}$$

This proof can also be presented as a proof diagram, resembling very much the diagram of Fig. 24. The main difference is that we replace x, y, and z by $\Gamma(at\_s_i)$ (according to the node), $\int Leak$, and T, respectively.

Acknowledgement

We gratefully acknowledge the help of Luca de Alfaro, Eddie Chang, Arjun Kapur, Yonit Kesten, Oded Maler, and Narciso Marti-Oliet for their careful reading of the manuscript and thank them for many helpful suggestions.

References

[AL92] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 1–27. Springer-Verlag, 1992.

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

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

[Dij75] E.W. Dijkstra. Guarded commands, nondeterminancy, and formal derivation of programs. *Comm. ACM*, 18(8):453–457, 1975.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, New Jersey, 1976.

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

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

[Hen91] T.A. Henzinger. *The Temporal Specification and Verification of Real-time Systems.* PhD thesis, Stanford University, 1991.

[Hen92] T.A. Henzinger. Sooner is safer than later. *Info. Proc. Lett.*, 43(3):135–142, 1992.

[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.

[HMP92] T. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 226–251. Springer-Verlag, 1992.

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

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

[KP92a] Y. Kesten and A. Pnueli. Age before beauty. Technical report, Dept. of Applied Mathematics, Weizmann Institute of Science, 1992.

[KP92b] Y. Kesten and A. Pnueli. Timed and hybrid statecharts and their textual representation. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lect. Notes in Comp. Sci.*, pages 591–619. Springer-Verlag, 1992.

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

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 447–484. Springer-Verlag, 1992.

[MP83] Z. Manna and A. Pnueli. Proving precedence properties: The temporal way. In *Proc. 10th Int. Colloq. Aut. Lang. Prog.*, volume 154 of *Lect. Notes in Comp. Sci.*, pages 491–512. Springer-Verlag, 1983.

[MP91a] Z. Manna and A. Pnueli. Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.

[MP91b] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[MP92a] Z. Manna and A. Pnueli. Time for concurrency. In *INRIA's 25th Anniversary Volume*, Lect. Notes in Comp. Sci. Springer-Verlag, 1992.

[MP92b] Z. Manna and A. Pnueli. Verifying hybrid systems. In A. Ravn and H. Rischel, editors, *Workshop on Hybrid Systems*, Lect. Notes in Comp. Sci. Springer-Verlag, 1992.

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

[NSY92] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 549–572. Springer-Verlag, 1992.

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

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

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

[SBM92] F. B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 618–639. Springer-Verlag, 1992.

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