Clocked Transition Systems* †

Zohar Manna‡ and Amir Pnueli†

Abstract. This paper presents a new computational model for real-time systems, called the *clocked transition system* model. The model is a development of our previous *timed transition* model, where some of the changes are inspired by the model of *timed automata*. The new model leads to a simpler style of temporal specification and verification, requiring no extension of the temporal language. For verifying safety properties, we present a run-preserving reduction from the new real-time model to the untimed model of fair transition systems. This reduction allows the (re)use of safety verification methods and tools, developed for untimed reactive systems, for proving safety properties of real-time systems.

1 Introduction

A formal framework for specifying and verifying temporal properties of reactive systems often contains the following components:

- 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 have considered in our previous work are all variants of temporal logic extended to deal with various aspects specific to the considered model, such as real-time and continuously changing variables.
- A *system description* language for describing systems within the model. We frequently 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.

---

* This paper was written as a tribute to Prof. C.S. Tang on his 70th birthday.

† This research was supported in part by the National Science Foundation under grant CCR-92-23226, by the Advanced Research Projects Agency under NASA grant NAG2-892, by the United States Air Force Office of Scientific Research under grant F49620-93-1-0139, and by Department of the Army under grant DAAH04-95-1-0317, by a basic research grant from the Israeli Academy of Sciences, and by the European Community ESPRIT Basic Research Action Project 6021 (REACT).

‡ Department of Computer Science, Stanford University, Stanford, CA 94305, e-mail: `manna@cs.stanford.edu`

† Department of Computer Science, Weizmann Institute, Rehovot, Israel, e-mail: `amir@wisdom.weizmann.ac.il`

- A set of *algorithmic methods* enabling a fully automatic verification of decidable subclasses of the verification problem such as the verification of finite-state systems (*model checking*).

In [MP93a], we considered a hierarchy of three models, each extending its predecessor, as follows:

- 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.

The computational model proposed for reactive systems is that of a *fair transition system* (FTS) [MP91].

The approach to real time presented in [MP93a] and [HMP94] is based on the computational model of *timed transition systems* (TTS) in which time itself is not explicitly represented but is reflected in a time stamp affixed to each state in a computation of a TTS. As a result of time not being available as a value of some variable, the requirement specification language must be based on a necessary extension of temporal logic. In [MP93a], we present two such extensions: *metric temporal logic* (MTL) which introduces bounded versions of the temporal operators, subscripted by an interval specification, and *temporal logic with age* (TL_Γ) which introduces the operator Γ, measuring the length of recent time in which a certain formula held continuously.

Consider, for example, the following 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.

In MTL, these two properties can be specified as follows:

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

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.

Using TL_Γ, the same properties can be specified by:

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

When constructing detailed proofs of such timed properties, we found it necessary to form many auxiliary invariants involving the age function Γ. Very soon, we realized that each age expression of the form $\Gamma(p)$ represents a timer that started measuring time as soon as p becomes true and is reset (or shut off) whenever p becomes false. This led to the suggestion that age expressions can be eliminated in favor of the introduction of explicit timers.

Meanwhile, the model of timed automata [AD94] has gained dominance in the arena of real-time algorithmic verification. This model uses timers which can only be reset by system transitions but increase uniformly whenever time progresses. However, aiming at algorithmic verification, this model can only capture finite-state systems.

In this paper we present a new computational model for real-time systems: *clocked transition system* (CTS). This model represents time by a set of clocks (timers) which increase uniformly whenever time progresses, but can be set to arbitrary values by system (program) transitions.

It is easy and natural to stipulate that one of the clocks T is never reset. In this case, T represents the *master clock* measuring real time from the beginning of the computation. This immediately yields the possibility of specifying timing properties of systems by unextended temporal logic, which may refer to any of the system variables, including the master clock T. Thus, the two yardstick properties considered above can be specified by the following (unextended) temporal formulas:

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

A first advantage of the new computational model over its predecessors is that it leads to a more natural style of verification proofs, stating invariants in terms of clocks, which are just another kind of system variables, instead of formulating invariants in terms of Γ-expressions. A second advantage is that we can use temporal logic without any extensions as the specification language, and can reuse many of the methods and tools developed for verifying reactive systems, under the FTS computational model, for verifying real-time systems under the CTS model. Another advantage of the CTS model is that it can be viewed as a natural first-order extension of the timed automata model [AD94].

In order to reuse FTS verification methods for verifying CTS properties, we will show that for every CTS Φ there exists a corresponding FTS P_Φ such that the runs of Φ and the runs of P_Φ coincide. It follows that any safety property is valid over the CTS Φ iff it is valid over the FTS P_Φ. This reduction from real-time systems to simpler transition systems enables us to use the deductive methodology developed for proving safety properties of reactive systems, as well as support systems for this methodology such as [MAB$^+$94], for establishing safety properties of timed systems.

It is interesting to note that the move from TTS to CTS brings us closer to the approach proposed in [AL94], which also recommends handling real time with a minimal extension of the reactive-systems formalism.

The restriction to safety properties is not as constraining as one may think. Many properties whose untimed version falls into the liveness class become safety properties when we consider their timed version. For example, termination of a program is a liveness property. However, the property of termination within 15 time units is a safety property. This indicates that, when we move to real-time systems, the set of safety properties we may want to prove constitutes an even higher portion of the total set of properties of interest than in the case of untimed reactive-systems. We refer the reader to [Hen92] and [Pnu92] for further discussions of this point.

We refer the reader to [AH89], [Ost90], [AL94], and the survey in [AH92], for additional logics, models, and approaches to the verification of real-time systems. In the process algebra school, some of the representative approaches to real time are [NSY92], [MT90], and many others are listed in [Sif91].

The paper is organized as follows. In Section 2, we present some background material, consisting of the computational models of *fair transition systems* (FTS) and *timed transition systems* (TTS). We also introduce the simple programming language SPL, in which sample programs are presented, and its transition semantics in the two models. In Section 3 we present the new real-time computational model of *clocked transition systems* (CTS), compare it to the TTS model and illustrate proofs of some timing properties. In Section 4, we review the model of hybrid systems and show how it can be viewed as a natural extension of the CTS model. A similar reduction from the hybrid-systems model to an FTS is presented and shown to preserve all system runs. This reduction can be used to verify safety properties of hybrid systems, reusing methods and tools developed for reactive systems. The methodology is illustrated on an example.

2 Background: Previous Models

In this section, we present some background material, consisting of the formal framework for reactive systems, as well as the previous real-time model of timed transition systems (TTS) as presented in [MP93a].

2.1 Formal Framework for Reactive Systems

The computational model for reactive systems is given by a *fair transition system* (FTS) $\Phi = \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ consisting of:

- $V = \{u_1, ..., u_n\}$: A finite set of *system variables*. 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*. A satisfiable assertion characterizing the initial states.

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

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

  mapping each state $s \in \Sigma$ into a (possibly empty) set of τ*-successor* states $\tau(s) \subseteq \Sigma$.
  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 system variables. An unprimed version of a system 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). Informally, 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). Informally, 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 enableness 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 $\rho_I : (V = V')$.

Runs and Computations

Let $\Phi = \left\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \right\rangle$ be a fair transition system. A *run* of Φ is an infinite sequence of states $\sigma : s_0, s_1, s_2, ...,$ satisfying:

- *Initiation*: s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution*: For each $j = 0, 1, ...,$ the state s_{j+1} is a τ-successor of the state s_j, i.e., $s_{j+1} \in \tau(s_j)$, for some $\tau \in \mathcal{T}$. In this case, we say that the transition τ is *taken* at position j in σ.

A *computation* of Φ is a run satisfying:

- *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.

Example 1. For the description of reactive systems, we use the simple programming language SPL introduced in [MP91].

Figure 1 presents a simple program consisting of two processes communicating by the shared variable x.

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

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

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

Fig. 1. Program ANY-Y

The fair transition system $\Phi_{\text{ANY-Y}}$ associated with program ANY-Y, is defined as follows:

- *System Variables:* $V = \{\pi, x, y\}$. Variable π is a control variable ranging over subsets of the locations of program ANY-Y: $\{\ell_0, \ell_1, \ell_2, m_0, m_1\}$. The value of π in a state denotes all the locations of the program in which control currently resides.
- *Initial Condition:*

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

- *Transitions:* $\mathcal{T} : \{\tau_I, \ell_0, \ell_1, m_0\}$ with transition relations:

$$\rho_{\tau_I} : \; \pi' = \pi \wedge x' = x \wedge y' = y$$

$$\rho_{\ell_0} : \; \ell_0 \in \pi \wedge \begin{pmatrix} x = 0 \;\wedge\; \pi' = \pi - \{\ell_0\} \cup \{\ell_1\} \\ \vee \\ x \neq 0 \;\wedge\; \pi' = \pi - \{\ell_0\} \cup \{\ell_2\} \end{pmatrix} \wedge x' = x \wedge y' = y$$

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

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

- Set of *Just Transitions:* $\mathcal{J} : \mathcal{T} - \{\tau_I\} = \{\ell_0, \ell_1, m_0\}$
- Set of *Compassionate transitions:* $\mathcal{C} : \phi$

Following are some runs and computations of $\Phi_{\text{ANY-Y}}$.

$$\sigma_0 : \begin{cases} \langle \pi : \{\ell_0, m_0\},\, x:0,\, y:0\rangle \xrightarrow{m_0} \langle \pi : \{\ell_0, m_1\},\, x:1,\, y:0\rangle \xrightarrow{\ell_0} \\ \langle \pi : \{\ell_2, m_1\},\, x:1,\, y:0\rangle \xrightarrow{\tau_I} \langle \pi : \{\ell_2, m_1\},\, x:1,\, y:0\rangle \xrightarrow{\tau_I} \\ \ldots \end{cases}$$

$$\sigma_1 : \begin{cases} \langle \pi : \{\ell_0, m_0\},\, x:0,\, y:0\rangle \xrightarrow{\ell_0} \langle \pi : \{\ell_1, m_0\},\, x:0,\, y:0\rangle \xrightarrow{\ell_1} \\ \langle \pi : \{\ell_0, m_0\},\, x:0,\, y:1\rangle \xrightarrow{m_0} \langle \pi : \{\ell_0, m_1\},\, x:1,\, y:1\rangle \xrightarrow{\ell_0} \\ \langle \pi : \{\ell_2, m_1\},\, x:1,\, y:1\rangle \xrightarrow{\tau_I} \langle \pi : \{\ell_2, m_1\},\, x:1,\, y:1\rangle \xrightarrow{\tau_I} \\ \ldots \end{cases}$$

$$\sigma_2 : \begin{cases} \langle \pi : \{\ell_0, m_0\},\, x:0,\, y:0\rangle \xrightarrow{\ell_0} \langle \pi : \{\ell_1, m_0\},\, x:0,\, y:0\rangle \xrightarrow{\ell_1} \\ \langle \pi : \{\ell_0, m_0\},\, x:0,\, y:1\rangle \xrightarrow{\ell_0} \langle \pi : \{\ell_1, m_0\},\, x:0,\, y:1\rangle \xrightarrow{\ell_1} \\ \langle \pi : \{\ell_0, m_0\},\, x:0,\, y:2\rangle \xrightarrow{\ell_0} \ldots \end{cases}$$

Both σ_0 and σ_1 are computations of program ANY-Y. The infinite sequence σ_2 in which m_0 is never taken, is a run but not a computation. This is because transition m_0 is continually enabled but not taken in σ_2, violating the requirement of justice towards m_0. ▟

Specification Language

To specify properties of reactive systems, we use the language of temporal logic, as presented in [MP91]. We use only the following:

- State formulas (assertions) - any first-order formula. For locations ℓ_i, ℓ_j, and ℓ_k, we denote:

$$\begin{aligned} at\_\ell_i &: \ell_i \in \pi \\ at\_\ell_{j,k} &: at\_\ell_j \vee at\_\ell_k \end{aligned}$$

- $\square\, p$ - Always p, where p is an assertion.
- $\diamondsuit\, p$ - Eventually p, where p is an assertion.

For a state s and assertion p, we write $s \models p$ to indicate that p holds (is true) over s. Let $\sigma : s_0, s_1 \ldots$ be an infinite sequence of states. We say that $\sigma \models \square\, p$ iff for *all* $i \geq 0$, $s_i \models p$. We say that $\sigma \models \diamondsuit\, p$ iff for *some* $i \geq 0$, $s_i \models p$.

A temporal formula φ is said to be *valid over* program P (or P-*valid*) if $\sigma \models \varphi$ for every computation σ of P. We write $P \models \varphi$ to denote this fact.

Specification and Verification of Invariance Properties

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

Here we will consider only *invariance* properties, namely, properties that can be expressed by the formula $\Box\, p$, for some assertion p.

We now present proof rules for establishing the P-validity of an invariance formula. We focus our attention on a particular program P, specified by the components $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C}\rangle$. 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, $\{p\}\tau\{q\}$ for each $\tau \in T$.

Basic Invariance Rule

The basic tool for establishing invariance properties is the following rule B-INV.

Rule B-INV

$$\begin{array}{ll} \text{B1.} & \Theta \;\rightarrow\; p \\ \text{B2.} & \{p\}\,\tau\,\{p\} \text{ for every } \tau \in \mathcal{T} \\ \hline & P \models \Box\, p \end{array}$$

Premise B1 of the rule ensures that all initial states satisfy the assertion p. Premise B2 ensures that the successor of each p-state in every run (and hence in every computation) is also a p-state. By induction on the states in a run, it follows that p holds on all states of all runs. Consequently, if premises B1 and B2 are valid then p is an invariant of the considered program. In the application of rule B-INV, it is not necessary to check premise B2 for the idling transition τ_I, since the idling transition trivially preserves every assertion.

For example, we may use rule B-INV to establish the invariance of the assertion

$$p: \quad (x = 0 \wedge at\_\ell_{0,1}) \;\vee\; at\_m_1,$$

over program ANY-Y (Fig. 1). This assertion claims that, at every state in the execution of program ANY-Y, either control of process P_1 is at ℓ_0 or ℓ_1 and $x = 0$, or control of P_2 is at m_1. In particular, it implies that if P_1 is at ℓ_2 then P_2 has already arrived in m_1.

Applying rule B-INV to this choice of the assertion p, premise B1 assumes the form

$$\underbrace{\pi = \{\ell_0, m_0\} \;\wedge\; x = 0 \;\wedge\; \cdots}_{\Theta} \quad \rightarrow \quad \underbrace{(x = 0 \wedge at\_\ell_{0,1}) \vee \cdots}_{p}$$

which is obviously valid.

Premise B2 has to be checked for each $\tau \in \{\tau_I, \ell_0, \ell_1, m_0\}$. For example, premise B2 for ℓ_0 assumes the form

$$\left(\begin{array}{l} \underbrace{\cdots \wedge \begin{pmatrix} x = 0 \;\wedge\; \pi' = \pi - \{\ell_0\} \cup \{\ell_1\} \\ \vee \\ x \neq 0 \;\wedge\; \pi' = \pi - \{\ell_0\} \cup \{\ell_2\} \end{pmatrix} \wedge\; x' = x \;\wedge\; \cdots}_{\rho_{\ell_0}} \\ \wedge \quad \underbrace{\big(x = 0 \wedge (\ell_0 \in \pi \vee \ell_1 \in \pi)\big) \;\vee\; m_1 \in \pi}_{p} \end{array}\right) \rightarrow$$

$$\underbrace{\big(x' = 0 \wedge (\ell_0 \in \pi' \vee \ell_1 \in \pi')\big) \;\vee\; m_1 \in \pi'}_{p'}.$$

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

The Need for a Stronger Assertion

In some cases, rule B-INV is not strong enough to prove the invariance of an assertion. Consider, for example, the assertion

$$\varphi\colon \quad at\_\ell_{0,1} \;\vee\; at\_m_1.$$

It is obvious that assertion φ which is implied by the previously considered assertion $p\colon (x = 0 \wedge at\_\ell_{0,1}) \;\vee\; at\_m_1$ is also an invariant of program ANY-Y. Unfortunately, this cannot be proven by rule B-INV. If we try to apply rule B-INV to assertion φ, we encounter the following instance of premise B2:

$$\underbrace{\cdots \wedge \big(\cdots \vee (x \neq 0 \wedge \pi' = \pi - \{\ell_0\} \cup \{\ell_2\})\big) \;\wedge\; \cdots}_{\rho_{\ell_0}}$$

$$\wedge \quad \underbrace{(\ell_0 \in \pi \vee \ell_1 \in \pi) \;\vee\; m_1 \in \pi}_{\varphi} \quad \rightarrow \quad \underbrace{(\ell_0 \in \pi' \vee \ell_1 \in \pi') \;\vee\; m_1 \in \pi'}_{\varphi'}$$

which is falsified by $\pi = \{\ell_0, m_0\}, x = 1$.

To verify the invariance of an assertion such as φ, we use the property that the $\Box$ operator is monotonic. This can be stated by the following rule:

Rule $\Box$-MON	
M1.	$p \rightarrow \varphi$
M2.	$P \models \Box\, p$
	$P \models \Box\, \varphi$

According to this rule, if assertion p is an invariant of program P, and p implies the assertion φ, then φ is also an invariant of P. Thus, to prove the invariance of assertion φ, we strengthen it into the assertion p and prove the invariance of p by rule B-INV, as done above.

Invariance Verification Diagram

An effective way of presenting the verification of an invariance property by a combination of rules B-INV and $\Box$-MON is provided by the graphical formalism of *verification diagrams* [MP94]. An (invariance) verification diagram is a directed labeled graph, constructed as follows:

- *Nodes* in the graph are labeled by assertions $\varphi_1, \ldots, \varphi_m$. We will often refer to a node by the assertion labeling it.
- *Edges* in the graph represent transitions between assertions. Each edge departs from one node, connects to another, and is labeled by the name of a transition in the program. We refer to an edge labeled by τ as a *τ-edge.*
- Some of the nodes are designated as *initial nodes*. They are annotated by an entry arrow .

For example, in Fig. 2 we present a verification diagram for program ANY-Y.

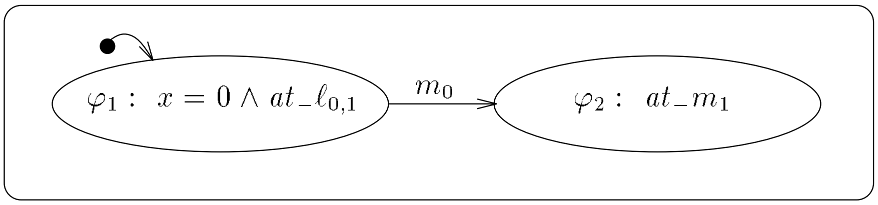

Fig. 2. Verification Diagram D_1

Verification Conditions for Diagrams

With each verification diagram, we associate the following *verification conditions*:

- Let φ_1 be the (single) initial node in the diagram. The verification condition corresponding to premise B1 of rule B-INV is given by

  $$\Theta \ \rightarrow \ \varphi_1.$$

- Let φ be a node in the graph, τ be a non-idling transition in the program, and let $\varphi_1, \ldots, \varphi_k$ be the nodes reached by τ-edges departing from φ. The case that no τ-edges depart from φ, i.e., $k = 0$, is also included. The verification condition associated with φ and τ is given by

  $$\{\varphi\}\ \tau\ \{\varphi \vee \varphi_1 \vee \ldots \vee \varphi_k\}$$

For example, the verification conditions associated with diagram D_1 (Fig. 2), are:

$$
\begin{array}{ll}
\text{B1:} & \Theta \rightarrow\ x = 0 \wedge at\_\ell_{0,1} \\
\text{B2 for } \varphi_1 \text{ and } \ell_0: & \{x = 0 \wedge at\_\ell_{0,1}\}\, \ell_0\, \{x = 0 \wedge at\_\ell_{0,1}\} \\
\text{B2 for } \varphi_1 \text{ and } \ell_1: & \{x = 0 \wedge at\_\ell_{0,1}\}\, \ell_1\, \{x = 0 \wedge at\_\ell_{0,1}\} \\
\text{B2 for } \varphi_1 \text{ and } m_0: & \{x = 0 \wedge at\_\ell_{0,1}\}\, m_0\, \{(x = 0 \wedge at\_\ell_{0,1})\ \vee\ at\_m_1\} \\
\text{B2 for } \varphi_2 \text{ and } \ell_0: & \{at\_m_1\}\, \ell_0\, \{at\_m_1\} \\
\text{B2 for } \varphi_2 \text{ and } \ell_1: & \{at\_m_1\}\, \ell_1\, \{at\_m_1\} \\
\text{B2 for } \varphi_2 \text{ and } m_0: & \{at\_m_1\}\, m_0\, \{at\_m_1\}
\end{array}
$$

We say that a verification diagram is *valid* if all the verification conditions associated with the diagram are valid.

Let D_P be a verification diagram associated with a program P. Let $\varphi_1, \ldots, \varphi_m$ be the assertions labeling the nodes of D_P.

Claim 1 *If the verification diagram D_P is valid, then*

$$P \models \Box \bigvee_{i=1}^{m} \varphi_i$$

If, in addition, $\varphi_i \rightarrow\ p$ for every $i = 1, \ldots, m$, then

$$P \models \Box\, p.$$

For example, all the verification conditions associated with diagram D_1 are valid and the verification diagram D_1 establishes

$$\text{ANY-Y} \models (at\_\ell_{0,1}\ \vee\ at\_m_1) \tag{1}$$

Encapsulation Conventions

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

We use the following conventions:

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

With these conventions we can draw a more detailed verification diagram establishing (1), as shown in Fig. 3.

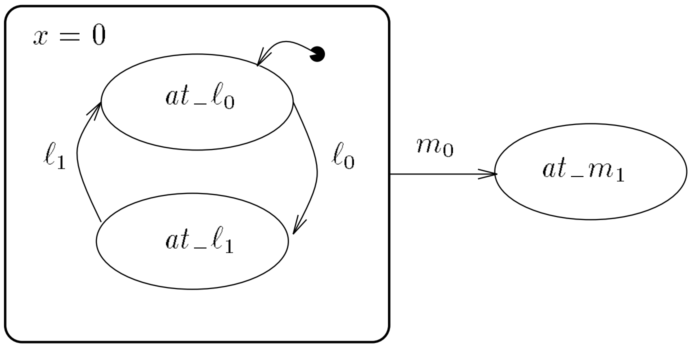

Fig. 3. A more detailed diagram, using encapsulation conventions

In the diagram of Fig. 3, the single assertion $x = 0 \wedge at\_\ell_{0,1}$, has been broken into the two sub-cases: $x = 0 \wedge at\_\ell_0$ and $x = 0 \wedge at\_\ell_1$, explicitly displaying the fact that transitions ℓ_0 and ℓ_1 cause the system to move between these two sub-cases.

2.2 Timed Transition Systems

Next, we review the model of *timed transition systems* (TTS), which was the computational model proposed in [MP93a] for modeling real-time systems.

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 *system 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 the first three components of a TTS are idntical to the first three components of an FTS, but that in the TTS we eliminate the fairness-related components of justice and compassion and replace them by the specification of lower and upper bounds. The role of the justice requirements in an FTS is to guarantee that transitions that have been enabled long enough are taken sufficiently frequently. The time-bounds components impose a quantified version of the justice requirements by demanding that transition τ cannot be continually enabled for more than u_τ without being taken. Thus, in the presence of the more stringent time-bounds requirements, justice is superfluous. It is possible to retain the requirement of compassion but, in the simpler model presented here, we decided to omit this requirement too.

We introduce a special variable T, sometimes called the *master 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.

Runs and Computations

A *run* of a timed transition system $S : \langle V, \Theta, \mathcal{T}, l, u\rangle$ is an infinite sequence of situations

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

satisfying:

- *Initiation:* $s_0 \models \Theta$ and $t_0 = 0$.
- *Consecution:* For each $j = 0, 1, \ldots$,
  - Either $t_j = t_{j+1}$ and $s_{j+1} \in \tau(s_j)$ for some transition $\tau \in \mathcal{T}$, or
  - $s_j = s_{j+1}$ and $t_j < t_{j+1}$. We refer to this step as a *tick step*, implying that time has progressed.
- *Lower bound:* For every transition $\tau \in \mathcal{T}$ and position $j \geq 0$, if τ is taken at j, there exists a position i, $i \leq j$, such that $t_i + l_\tau \leq t_j$ and τ is enabled on $s_i, s_{i+1}, \ldots, s_j$. This implies that τ must be continuously enabled for at least l_τ time units before it can be taken.
- *Upper bound:* For every transition $\tau \in \mathcal{T}$ and position $i \geq 0$, if τ is enabled at position i, there exists a position j, $i \leq j$, such that $t_i + u_\tau \geq t_j$ and τ is disabled on s_j. In other words, τ cannot be continuously enabled for more than u_τ time units without being taken.

A *computation* of S is a run satisfying:

- *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 $j - 1$.

As shown in [HMP94], 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 2. Consider the simple timed transition system given by:

- System 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.

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

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.

$$\sigma_2 : \ \langle x:0\,,\, y:0\,,\, T:0\rangle \xrightarrow{tick} \langle x:0\,,\, y:0\,,\, T:2\rangle \xrightarrow{\tau_0} \langle x:1\,,\, y:0\,,\, T:2\rangle \xrightarrow{tick}$$
$$\langle x:1\,,\, y:0\,,\, T:3\rangle \xrightarrow{\tau_2} \langle x:1\,,\, y:1\,,\, T:3\rangle \xrightarrow{tick} \cdots$$

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 [AL94].

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 system 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 TTS S is called *non-zeno* if every prefix of a run of S can be extended to a computation of S. From now on, we restrict our attention to non-zeno timed transition systems.

Timed Extension of the Textual Language

In [MP93a], we presented the system description language of *timed statecharts* and showed how to interpret it as a TTS. Here we will only consider the TTS corresponding to the textual language SPL, extended to deal with real time.

Earlier on, we introduced the simple programming language SPL for the untimed 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 section we will show how such properties are specified and verified.

To distinguish between the interpretation of a program P as a fair transition system and its interpretation as a timed transition system (when given 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 is meaningless for ANY-Y, whose computations as a fair transition system do not contain any timing information. We define an SPL_T program to be any SPL program with time bounds associated with its transitions, such as ANY-Y_T.

3 Real-Time Systems

We now introduce our new computational model for real-time systems, intended to replace the TTS model.

3.1 Computational Model: Clocked Transition System

Real-time systems are modeled as *clocked transition systems* (CTS). A clocked transition system $\Phi = \langle V, \Theta, \mathcal{T}, \Pi \rangle$ consists of:

- $V = \{u_1, \ldots, u_n\}$: A finite set of *system variables*. The set $V = D \cup C$ is partitioned into D the set of *discrete variables* and C the set of *clocks*. Clocks always have the type *real*. The discrete variables can be of any type. We introduce a special clock $T \in C$, representing the *master clock*, as one of the system variables.
- Θ : The *initial condition*. A satisfiable assertion characterizing the initial states. It is required that

$$\Theta \quad \rightarrow \quad T = 0,$$

  i.e., $T = 0$ at all initial states.
- $\mathcal{T}$: 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 $\tau(s) \subseteq \Sigma$. As before, the successor function for τ is defined by a transition relation $\rho_\tau(V, V')$, which may refer to V and modify $V - \{T\}$. For every $\tau \in \mathcal{T}$, it is required that

$$\rho_\tau \quad \rightarrow \quad T' = T.$$

- Π : The *time-progress condition.* An assertion over V. The assertion is used to specify a global restriction over the progress of time.

Extended Transitions

Let $\Phi : \langle V, \Theta, \mathcal{T}, \Pi \rangle$ be a clocked transition system. We define the set of *extended transitions* $\mathcal{T}_T$ associated with Φ as follows:

$$\mathcal{T}_T = \mathcal{T} \cup \{tick\}.$$

Transition *tick* is a special transition intended to represent the passage of time. Its transition relation is given by:

$$\rho_{tick} : \quad \exists \Delta > 0. \begin{pmatrix} (D' = D) \wedge (C' = C + \Delta) \\ \wedge \\ \forall t \in [0, \Delta).\ \Pi(D, C + t) \end{pmatrix}$$

Let $D = \{u_1, \ldots, u_m\}$ be the set of discrete variables of Φ and $C = \{c_1, \ldots, c_k\}$ be the set of its clocks. Then, the expression $C' = C + \Delta$ is an abbreviation for

$$c'_1 = c_1 + \Delta \ \wedge \ \cdots \ \wedge \ c'_k = c + \Delta,$$

and $\Pi(D, C + t)$ is an abbreviation for $\Pi(u_1, \ldots, u_m, c_1 + t, \ldots, c_k + t)$.

Runs and Computations

Let $\Phi : \langle V, \Theta, \mathcal{T}, \Pi \rangle$ be a clocked transition system. A *run* of Φ is an infinite sequence of states $\sigma : s_0, s_1, \ldots$ satisfying:

- *Initiation:* $s_0 \models \Theta$
- *Consecution:* For each $j = 0, 1, ..., s_{j+1} \in \tau(s_j)$, for some $\tau \in \mathcal{T}_T$.

A *computation* of Φ is a run satisfying:

- *Time Divergence:* The sequence $s_0[T], s_1[T], \ldots$ grows beyond any bound. That is, as i increases, the value of T at s_i increases beyond any bound.

A temporal formula φ is said to be *valid over* CTS Φ (Φ*-valid*) if φ holds over all computations of Φ.

Non-Zeno Systems

Similarly to the TTS case, we say that a CTS is *non-zeno* if every finite sequence of states $\sigma : s_0, \ldots, s_k$ satisfying the requirements of initiation and consecution can be extended into a computation. In the following, we restrict our attention to non-zeno CTS's.

3.2 The Fair Transition System induced by a CTS

Let $\Phi: \langle V, \Theta, \mathcal{T}, \Pi \rangle$ be a non-zeno CTS. Consider the following FTS:

$$P_\Phi: \quad \langle V, \Theta, \mathcal{T}_T, \emptyset, \emptyset \rangle,$$

in which the justice and compassion sets are taken to be empty. We refer to P_Φ as the FTS *induced by the* CTS Φ. An important relation between a CTS Φ and the FTS it induces is identified by the following claim:

Claim 2 *A non-zeno* CTS Φ *and the* FTS P_Φ *it induces have identical sets of runs.*

Justification: The claim is a direct consequence of the definition of runs for Φ and P_Φ.

Corollary 3 *An invariance formula* $\Box p$ *is valid over a non-zeno* CTS Φ *iff it is valid over the induced* FTS P_Φ.

Justification: An invariance formula $\Box p$ is falsified by a model σ (i.e., $\sigma \not\models \Box p$) iff σ has a prefix σ' such that all infinite extensions of σ' falsify $\Box p$. We observe that the set of prefixes of all computations of a non-zeno CTS Φ is equal to the set of prefixes of all runs of Φ which, in turn, is equal to the set of prefixes of all runs of P_Φ and, hence, is equal to the set of prefixes of all computations of P_Φ. It follows that the invariance formula $\Box p$ is Φ-valid iff it is P_Φ-valid. ▟

Corollary 3 is not restricted to invariance formulas but holds for all safety properties. It reduces the problem of proving that a safety formula φ (a formula specifying a safety property) is valid over a given (non-zeno) CTS Φ to proving the validity of φ over the FTS P_Φ. Thus, we may use all the techniques and tools developed for verifying safety properties of FTS's for proving safety properties of CTS's.

3.3 From Timed Transition Systems to Clocked Transition Systems

The CTS model is at least as expressive as the TTS model. To see this, we present a sketch of a general translation from a TTS S to a CTS Φ_S.

Recall that a TTS-transition τ is associated with lower and upper time bounds $[L, U]$ (possibly dependent on τ), with the following implications:

- Transition τ can be taken only after being continuously enabled for at least L time units.
- Transition τ cannot be continuously enabled for more than U time units without being taken.

We start by showing how to translate each TTS-transition into a CTS-transition. We will proceed in order of increasing complexity.

Data-independent transitions

Consider first the case of TTS-transition τ^{TTS} whose enableness condition depends only on control but not on data. In timed statecharts ([MP93a]) such a transition can be drawn as shown at the TTS of Fig. 4. This transition is enabled whenever

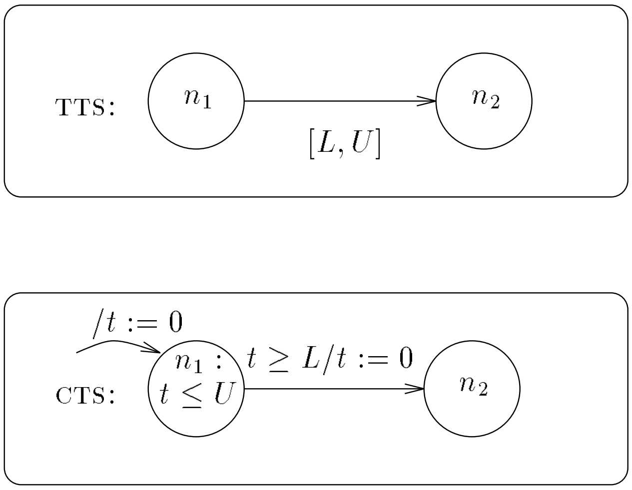

Fig. 4. Translation of a data-independent transition.

control is at node n_1. It can be taken only after control has stayed at n_1 for at least L time units. On the other hand, control cannot stay at n_1 for more than U time units, without this or another transition departing from n_1 being taken. Thus, the transition relation for the displayed TTS-transition is given by

$$\rho_\tau^{\text{TTS}}: \quad n_1 \in \pi \wedge \pi' = \pi - \{n_1\} \cup \{n_2\} \wedge \mathit{pres}(V - \{\pi\}),$$

where we assume that π is a control variable containing at any point the names of the nodes in which control currently resides. For a set of variables $U \subseteq V$, the formula $\mathit{pres}(U)$ stands for the conjunction

$$\mathit{pres}(U): \quad \bigwedge_{y \in U} y' = y,$$

implying that all variables in U are preserved by the transition.

The translation of this transition is also presented in Fig. 4. As we see in the figure, a clock t is allocated to measure the duration of time control stayed continually at n_1. This clock is reset on any transition entering node n_1. The corresponding CTS transition relation is given by

$$\rho_\tau^{\text{CTS}}: \quad n_1 \in \pi \wedge t \geq L \wedge (\pi' = \pi - \{n_1\} \cup \{n_2\}) \wedge t' = 0 \wedge \mathit{pres}(V - \{\pi, t\}).$$

Transition τ^{CTS} is enabled whenever control is at n_1 and the clock t is not lower than L, which implies that control has stayed at n_1 for at least L time units. On taking this transition, control moves to n_2 while resetting the clock t to 0. In addition to the generation of the CTS-transition τ^{CTS}, the TTS-transition τ^{TTS} also contributes to the time-progress condition Π of the CTS Φ_S the following conjunct:

$$n_1 \in \pi \quad \rightarrow \quad t \leq U$$

This conjunct disallows the progress of time when control is at n_1 while $t \geq U$, forcing transition τ^{CTS} to be taken before time can progress. In the diagram of Fig. 4, this conjunct is represented by the condition $t \leq U$ inscribed inside node n_1, ensuring that the upper bound for transition τ^{CTS} is respected.

Immediate data-dependent transitions

Next, we consider the case of a TTS-transition τ^{TTS} whose enableness condition depends on the data variables, but whose time bounds satisfy $L = U = 0$. Transitions with such time bounds are called *immediate*. In Fig. 5, we present such a transition and its CTS translation. The label c represents the data condition

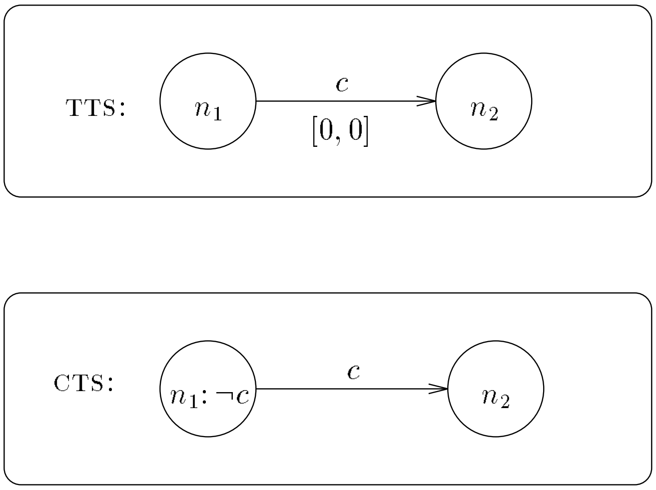

Fig. 5. Translation of immediate data-dependent transition.

under which the transition is enabled. The transition relation for τ^{CTS} is given by

$$\rho_\tau^{\text{CTS}}: \quad n_1 \in \pi \wedge c \wedge (\pi' = \pi - \{n_1\} \cup \{n_2\}) \wedge t' = 0 \wedge pres(V - \{\pi, t\}).$$

Transition τ^{TTS} also contributes the following conjunct to the time-progress condition.

$$n_1 \in \pi \quad \rightarrow \quad \neg c,$$

which allows time to progress while control is at n_1 only when c is false (and therefore τ^{CTS} is disabled). This identifies transition τ^{CTS} as *urgent* in the sense that, when the transition is enabled, time cannot progress before it is taken.

Non-immediate data-dependent transitions

The most complex case is that of a non-immediate data-dependent transition. In Fig. 6, we present the translation of such TTS-transition. The translation uses

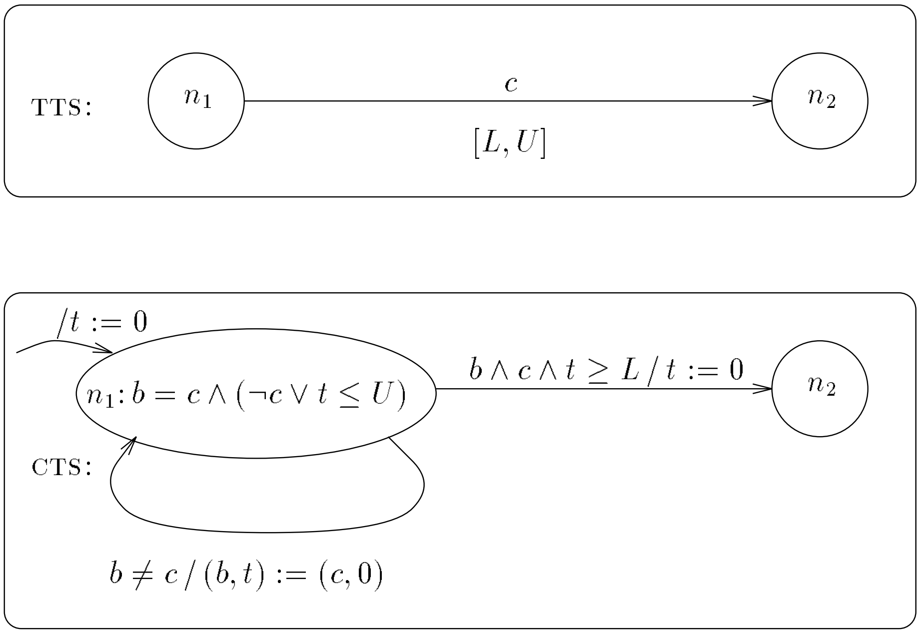

Fig. 6. Non-immediate transition, with condition.

a clock t to measure the length of time that control has been continually at n_1 while c was continuously true. An additional boolean variable b is introduced to guarantee that the system senses recent changes in the value of the condition c. Part of the time-progress condition associated with node n_1 is that b equals c. If this is not the case, then time cannot progress and the self-transition drawn from n_1 to itself must be taken. This transition is enabled whenever b is different from c, and its effect is to set b to the current value of c and reset the clock t to zero. The intention is that t measures the time since the last change in the value of c. The transition from n_1 to n_2 which is intended to translate the original TTS transition, can be taken only if both b and c are true, and $t \geq L$, implying that the time since c last changed to T is at least L.

Formally, the translation introduces a clock t, a boolean variable b, and two transitions τ^{CTS} and *sense*, whose transition relations are given by

$$\rho_\tau^{\text{CTS}}: \quad n_1 \in \pi \wedge b \wedge c \wedge t \geq L \wedge (\pi' = \pi - \{n_1\} \cup \{n_2\}) \wedge t' = 0 \wedge pres(V - \{\pi, t\})$$

and

$$\rho_{sense}: \quad n_1 \in \pi \ \wedge\ b \neq c \ \wedge\ b' = c \ \wedge\ t' = 0 \ \wedge\ pres(V - \{b, t\}),$$

respectively. Also, the following conjunct is added to the time-progress condition:

$$n_1 \in \pi \quad \rightarrow \quad b = c \ \wedge\ (\neg c \vee t \leq U).$$

The rest of the translation

Having sketched how each TTS-transition can be translated to one or more CTS-transitions, we can complete the description of a translation from a general TTS $S = \langle V, \Theta, \mathcal{T}, l, u \rangle$ into a corresponding CTS $\Phi = \langle \widetilde{V}, \widetilde{\Theta}, \widetilde{\mathcal{T}}, \widetilde{\Pi} \rangle$.

- For system variables we take $\widetilde{V} = V \cup \{b_1, \ldots, b_k\} \cup \{t_1, \ldots, t_m\} \cup \{T\}$, where $b_1, \ldots, b_k$ are the auxiliary boolean variables introduced by translations of non-immediate data-dependent transitions, $t_1, \ldots, t_m$ are the clocks introduced by any of the transition translations, and T is the master clock, measuring time from the beginning of the computation.
- The initial condition is given by

  $$\widetilde{\Theta}: \quad \Theta \ \wedge\ t_1 = 0 \ \wedge\ \cdots \ \wedge\ t_m = 0.$$

- The transitions $\widetilde{\mathcal{T}}$ include all the CTS-transitions generated by translating the TTS-transitions in $\mathcal{T}$.
- The time-progress condition $\widetilde{\Pi}$ is the conjunction of all the time-progress clauses generated by translating the TTS-transitions in $\mathcal{T}$.

The Relation between a TTS and its CTS translation

It would be nice if we could claim that the set of runs of a TTS S coincides with the set of runs of its translated CTS Φ (ignoring the additional variables and clocks introduced in Φ). Unfortunately, this is not the case.

Consider, for example, the TTS S_1, presented in Fig. 7. Apart from different sampling points, system S_1 has essentially one computation, which is represented by the following sequence of situations:

$$\begin{aligned}
\sigma_1 :\ & \langle \pi\colon \{\ell_0, m_0\},\ x\colon 0;\ T\colon 0 \rangle \xrightarrow{tick} \langle \pi\colon \{\ell_0, m_0\},\ x\colon 0;\ T\colon 1 \rangle \xrightarrow{m_0} \\
& \langle \pi\colon \{\ell_0, m_1\},\ x\colon 1;\ T\colon 1 \rangle \xrightarrow{m_1} \\
& \langle \pi\colon \{\ell_0, m_0\},\ x\colon 0;\ T\colon 1 \rangle \xrightarrow{tick} \langle \pi\colon \{\ell_0, m_0\},\ x\colon 0;\ T\colon 2 \rangle \xrightarrow{m_0} \\
& \langle \pi\colon \{\ell_0, m_1\},\ x\colon 1;\ T\colon 2 \rangle \xrightarrow{m_1} \cdots
\end{aligned}$$

An obvious fact about this system is that control can never reach node ℓ_1 in the upper level of the diagram. The reason for this is that transition ℓ_0 is never

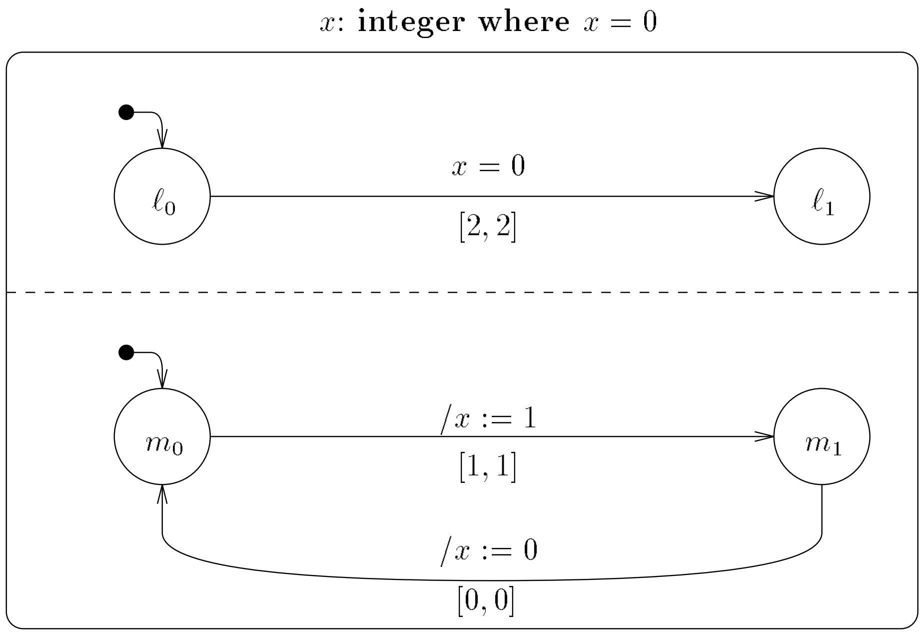

Fig. 7. TTS S_1.

continuously enabled for 2 time units. This is because every time unit, the lower part of the diagram sets x to 1 and then back to 0 again.

On the other hand, consider CTS Φ_1, presented in Fig. 8 which was obtained by the translation of S_1 according to the recipe given above. In the translation we took the liberty of defining b to be an integer rather than a boolean variable. System Φ_1 has a computation corresponding to computation σ_1 of S_1, in which the upper component periodically detects that $x = 1$ every single time unit, and therefore can never take transition ℓ_0. On the other hand, it also has the following computation

$$\begin{array}{l}
\langle \pi\colon \{\ell_0, m_0\},\ x\colon 0,\ b\colon 0,\ t_1\colon 0, t_2\colon 0\rangle \ \xrightarrow{tick} \langle \pi\colon \{\ell_0, m_0\},\ x\colon 0,\ b\colon 0,\ t_1\colon 1, t_2\colon 1\rangle \ \xrightarrow{m_0} \\
\langle \pi\colon \{\ell_0, m_1\},\ x\colon 1,\ b\colon 0,\ t_1\colon 1, t_2\colon 0\rangle \ \xrightarrow{m_1} \\
\langle \pi\colon \{\ell_0, m_0\},\ x\colon 0,\ b\colon 0,\ t_1\colon 1, t_2\colon 0\rangle \ \xrightarrow{tick} \langle \pi\colon \{\ell_0, m_0\},\ x\colon 0,\ b\colon 0,\ t_1\colon 2, t_2\colon 1\rangle \ \xrightarrow{\ell_0} \\
\langle \pi\colon \{\ell_1, m_0\},\ x\colon 0,\ b\colon 0,\ t_1\colon 2, t_2\colon 1\rangle \ \xrightarrow{m_0} \cdots,
\end{array}$$

in which control does get to ℓ_1. The reason that this is possible in CTS Φ_1 is that the mechanism within node ℓ_0 which is supposed to detect an instance of $x \neq 0$ is allowed to ignore such instances if their duration is 0.

To see how this is possible, consider state $\langle \pi\colon \{\ell_0, m_1\},\ x\colon 1,\ b\colon 0,\ t_1\colon 1, t_2\colon 0\rangle$. At this point in the execution, both transition $\ell_0 \to \ell_0$, which detects that x equals 1, and transition $m_1 \to m_0$ are enabled, and the time-progress condition

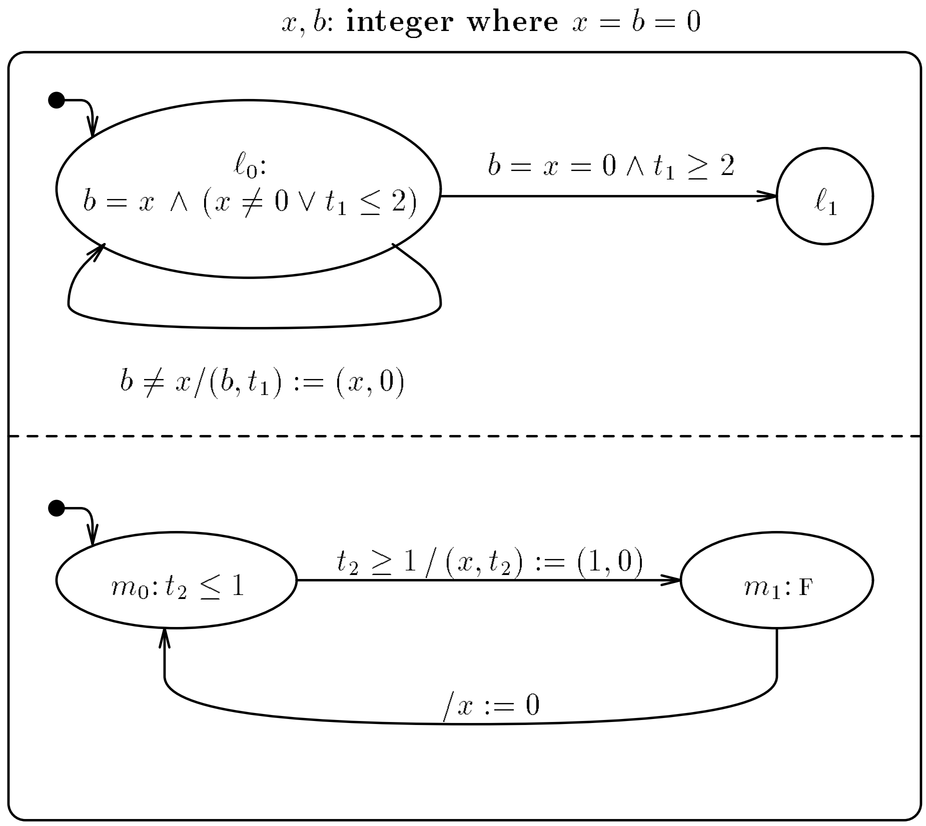

Fig. 8. CTS Φ_1, translating TTS S_1.

within ℓ_0 is false (as $b \neq x$). This means that, presently, time cannot progress and one of the enabled transitions must be taken. If $\ell_0 \rightarrow \ell_0$ is taken, then the system detects an instance of $x = 1$ and resets clock t_1. On the other hand, if transition $m_1 \rightarrow m_0$ is taken, as is the case in the computation above, transition $\ell_0 \rightarrow \ell_0$ is disabled again and the time-progress condition within ℓ_0 becomes true again. This means that under this nondeterministic choice, the system has lost the opportunity to detect an instance of $x = 1$.

However, apart from differences in the treatment of states that persist for zero durations, the behavior of a TTS is similar to the behavior of its corresponding translated CTS.

3.4 The CTS corresponding to an SPL$_T$ program

Let P_T be an SPL$_T$ program. That is, we are given an SPL program P with time bounds $[l_\tau, u_\tau]$ associated with each transition $\tau \in \mathcal{T}_P$. We will show how to construct a CTS Φ_{P_T} corresponding to the SPL$_T$ program P_T.

The construction can be described as a multi-stage process, which can be summarized by the following sequence:

$$\text{Program } P \longrightarrow \text{FTS } F_P \xrightarrow{+\text{ timing information}} \text{TTS } S_{P_T} \xrightarrow{\text{construction 3.3}} \text{CTS } \Phi_{P_T}$$

Example 3. We illustrate the construction on the SPL$_T$ program ANY-Y$_{[3,5]}$. This consists of program ANY-Y of Fig. 1 with the time bounds $[3,5]$ uniformly assigned to each of its statements. The CTS $\Phi_{\text{any-y}_{[3,5]}}$ associated with ANY-Y$_{[3,5]}$ is defined as follows:

- *System Variables:* $V = \{\pi, x, y, t_1, t_2, T\}$. In addition to the control variable π and data variables x and y, the system variables also include clock t_1, measuring delays in process P_1, clock t_2, measuring delays in process P_2, and the master clock T, measuring time from the beginning of the computation.
- *Initial Condition:*

$$\Theta: \quad \pi = \{\ell_0, m_0\} \wedge x = y = 0 \wedge t_1 = t_2 = T = 0.$$

- *Transitions:* $\mathcal{T} : \{\ell_0, \ell_1, m_0\}$ with transition relations:

$$\begin{aligned}
\rho_{\ell_0} &: \ \ell_0 \in \pi \wedge \begin{pmatrix} x = 0 \ \wedge \ \pi' = \pi - \{\ell_0\} \cup \{\ell_1\} \\ \vee \\ x \neq 0 \ \wedge \ \pi' = \pi - \{\ell_0\} \cup \{\ell_2\} \end{pmatrix} \wedge t_1 \geq 3 \wedge t_1' = 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\, pres(V - \{\pi, t_1\}) \\
\rho_{\ell_1} &: \ \ell_1 \in \pi \wedge \pi' = \pi - \{\ell_1\} \cup \{\ell_0\} \wedge y' = y + 1 \wedge t_1 \geq 3 \wedge t_1' = 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ pres(V - \{\pi, y, t_1\}) \\
\rho_{m_0} &: m_0 \in \pi \wedge \pi' = \pi - \{m_0\} \cup \{m_1\} \wedge x' = 1 \wedge t_2 \geq 3 \wedge t_2' = 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ pres(V - \{\pi, x, t_2\}). \ \blacksquare
\end{aligned}$$

- *Time-progress condition:*

$$\Pi: \quad (at\_\ell_{0,1} \ \rightarrow \ t_1 \leq 5) \wedge (at\_m_0 \ \rightarrow \ t_2 \leq 5)$$

Faithfulness of the Model

In view of our previous discussions of the differences between the TTS and CTS interpretations of non-immediate data-dependent transitions, one may be concerned that a translation that includes a TTS$\rightarrow$CTS link may involve some distortion of the semantics of statements that give rise to non-immediate data-dependent transitions such as the *await* statement. In fact, the converse is true, i.e., rather than distorting the faithful TTS modeling of timed programs, the CTS modeling rectifies some anomalies present in the TTS models. For example, a deeper analysis shows that the CTS interpretation of the *await* statement is more realistic than the TTS interpretation of that statement.

To sketch the issues, assume that process P_1 contains an ℓ: **await** c statement, and a concurrent process causes c to become false for intervals of zero time such that every time interval of length L (the lower bound assigned to ℓ), contains at least one zero-length lapse in the continuous holding of c. According to the TTS semantics, statement ℓ can never be fully executed, i.e., process P_1 will never progress beyond location ℓ. The CTS semantics is more liberal. It allows computations in which P_1 never progresses beyond ℓ but it also allows computations in which P_1 manages to ignore these zero duration interruptions in c and does progress beyond ℓ. Consequently, we believe the CTS interpretation of timed programs to be an improvement on their TTS interpretation.

Verifying Invariance Properties over CTS

As previously explained, to verify that the invariance formula $\square\, p$ is valid over a CTS Φ, one may prove instead that $\square\, p$ is valid over P_Φ, the FTS induced by Φ. However, the explicit reduction is not necessary and we may, instead, apply the following direct rule C-INV:

Rule C-INV

$$\begin{array}{ll} \text{C1.} & \Theta \;\rightarrow\; p \\ \text{C2.} & \{p\}\, \tau \,\{p\} \text{ for every } \tau \in \mathcal{T}_T \\ \hline & \Phi \models \square\, p \end{array}$$

As is the case with rule B-INV, rule C-INV is sound and (relatively) complete for proving all invariance properties of non-zeno CTS's.

In a similar way, we can adapt all proof rules for proving safety properties of fair transition systems ([MP95]) to apply to verification of the corresponding safety property over a non-zeno CTS. The only modification necessary is to consider all transitions in $\mathcal{T}_T$, wherever the FTS rule considers all transitions in $\mathcal{T}$.

Example 4. We use rule C-INV to prove that program ANY-Y$_{[3,5]}$ terminates within 15 time units, as specified by the following invariance formula:

$$\square\big(T \leq 15 \;\vee\; (at\_\ell_2 \wedge at\_m_1)\big)$$

The proof is represented by the verification diagram in Fig. 9. ▟

Example 5. As a second 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. Similar proofs to the one we will present here are given in [SBM92], [AL94], and [MMP92].

The algorithm is presented in Fig. 10 under the name of program MUTEX. By assigning all statements in MUTEX the uniform time bounds L and U, we obtain the timed program MUTEX$_{[L,U]}$. Assuming that $2L > U$, we prove that the mutual exclusion property

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

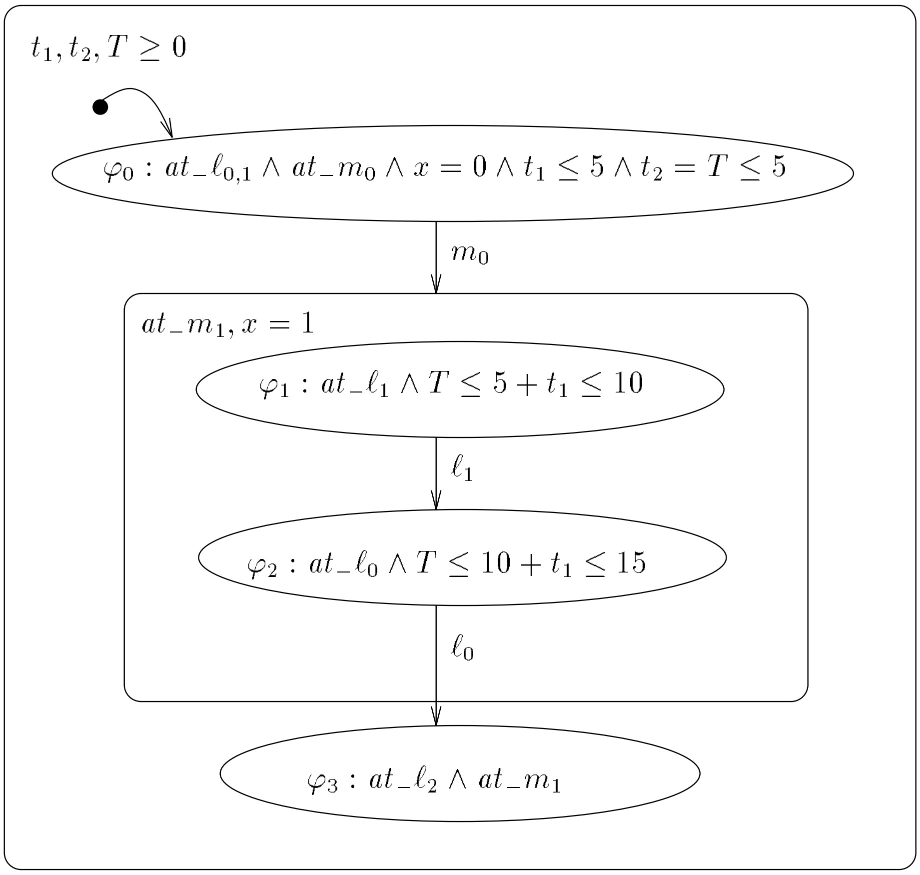

Fig. 9. Termination of ANY-Y within 15 time units.

is valid for $\text{MUTEX}_{[L,U]}$.

Premise $C1$ of rule C-INV, having the form

$$\underbrace{at\_\ell_0 \wedge at\_m_0 \wedge \ldots}_{\Theta} \rightarrow \underbrace{\neg(at\_\ell_4 \wedge at\_m_4)}_{p},$$

is obviously valid.

Premise $C2$ is proven as follows: The situations that appear to threaten the preservation of the assertion $\neg(at\_\ell_4 \wedge at\_m_4)$ are

$$at\_\ell_4 \wedge at\_m_3 \wedge x = 2 \qquad (2)$$

and

$$at\_m_4 \wedge at\_\ell_3 \wedge x = 1 \qquad (3)$$

$$
\begin{array}{c}
x\text{: } \textbf{integer where } x = 0 \\[1ex]
\left[\begin{array}{l}
\ell_0 : \textbf{await } x = 0 \\
\ell_1 : x := 1 \\
\ell_2 : \textbf{skip} \\
\ell_3 : \textbf{await } x = 1 \\
\ell_4 : \text{Critical}
\end{array}\right]
\quad \Bigg\| \quad
\left[\begin{array}{l}
m_0 : \textbf{await } x = 0 \\
m_1 : x := 2 \\
m_2 : \textbf{skip} \\
m_3 : \textbf{await } x = 2 \\
m_4 : \text{Critical}
\end{array}\right] \\[1ex]
- \ P_1 \ - \qquad\qquad\qquad - \ P_2 \ -
\end{array}
$$

Fig. 10. Program MUTEX, implementing Fischer's protocol.

which is proven symmetrically. We prove that (2) is impossible, by showing the invariance of

$$at\_\ell_4 \rightarrow \ x = 1 \tag{4}$$

The threatening situation to 4 is the following

$$at\_\ell_4 \wedge at\_m_1 \tag{5}$$

To show that (5) is impossible, we prove that process P_2 cannot wait at m_1 while process P_1 progresses from ℓ_1 to ℓ_4. This is proven in three major steps, showing the invariance of three assertions as follows:

- $at\_\ell_2 \wedge at\_m_1 \rightarrow \ t_2 \geq t_1$
  The reasoning: Initially, this implication trivially holds. Taking transition m_0 while at_ℓ_2 is impossible, since at_ℓ_2 implies $x \neq 0$. Taking transition ℓ_1 while at_m_1, implies $t'_1 = 0 \leq t_2$.
- $at\_\ell_3 \wedge at\_m_1 \rightarrow \ t_2 \geq t_1 + L$
  The initial holding of this implication is obvious. Transition m_0 cannot be taken while at_ℓ_3 holds. By the previous assertion, when transition ℓ_2 is taken from a at_m_1-state, the inequalities $t_2 \geq t_1$ and $t_1 \geq L$ hold, leading to $t_2 \geq L$. Since ρ_{ℓ_2} implies $t'_1 = 0$ and $t'_2 = t_2$, we have that $t'_2 \geq t'_1 + L$, i.e., $t_2 \geq t_1 + L$ holds after the transition.
- $at\_\ell_4 \wedge at\_m_1 \rightarrow \ t_2 \geq t_1 + 2 \cdot L$
  The invariance of this assertion is proven in a way similar to the preceding proof.

We conclude that the assumption $2 \cdot L > U$ leads to

$$at\_\ell_4 \wedge at\_m_1 \rightarrow \ t_2 > U$$

contradicting the invariance formula

$$\Box(at\_m_1 \ \rightarrow \ t_2 \leq U),$$

which can easily be shown to be valid over MUTEX$_{[L,U]}$. ▟

4 Hybrid Systems

In this section we consider the case of hybrid systems. Similarly to our treatment of real-time systems, we present a computational model for hybrid systems that admits a run-preserving reduction to an induced FTS. This enables us to use the methodologies and tools developed for fair transition systems to verify safety properties of hybrid systems.

4.1 Computation Model: Phase Transition System

Hybrid systems are modeled as phase transition systems (PTS). The PTS model was originally presented in [MMP92], [MP93b], as an extension of the TTS model. The PTS model presented here is an extension of the CTS model, and is thus slightly modified to support the desired reduction from a PTS to an induced FTS. A closely related model for hybrid systems is presented in [NOS+93].

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

- $V = \{u_1, ..., u_n\}$: A finite set of *system variables*. The set $V = D \cup I$ is partitioned into D the set of *discrete variables* and I the set of *integrators*. Integrators always have the type *real*. The discrete variables can be of any type. We introduce a special integrator $T \in I$ representing the *master clock*.
- Θ : The *initial condition*. A satisfiable assertion characterizing the initial states. It is required that

$$\Theta \quad \rightarrow \quad T = 0.$$

- $\mathcal{T}$: 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 $\tau(s) \subseteq \Sigma$. As before, the successor function for τ is defined by a transition relation $\rho_\tau(V, V')$, which may refer to V and modify $V - \{T\}$. For every $\tau \in \mathcal{T}$, it is required that

$$\rho_\tau \quad \rightarrow \quad T' = T.$$

- $\mathcal{A}$: A finite set of *activities*. Each activity $\alpha \in \mathcal{A}$ is represented by an *activity relation*:

$$p_\alpha \;\rightarrow\; I(t) = F^\alpha(V^0, t)$$

where p_α is a predicate over D called the *activation condition* of α. Activity α is said to be *active* in state s if its activation condition p_α holds on s. If p_α is *true*, it may be omitted.
Let $I = \{x_1, \ldots, x_m = T\}$ be the integrators of the system. The vector equality $I(t) = F^\alpha(V^0, t)$ is an abbreviation for the following set of individual equalities:

$$x_i(t) = F_i^\alpha(V^0, t), \quad \text{for each } i = 1, \ldots, m,$$

which define the evolution of the integrators throughout a phase of continuous change according to the activity α. The argument V^0 represents the initial values of all the system variables at the beginning of the phase.
For every $\alpha \in \mathcal{A}$ it is required that

$$F_i^\alpha(V^0, 0) = x_i^0, \quad \text{for every } i = 1, \ldots, m$$
$$F_T^\alpha(V^0, t) = F_m^\alpha(V^0, t) = T^0 + t.$$

That is, $F_i^\alpha(V^0, 0)$ agrees with the initial value of x_i, and the effect of evolution of length t on the master clock (integrator x_m) is to add t to T.
It is required that the activation conditions associated with the different activities be exhaustive and exclusive, i.e., exactly one of them holds on any state.

- Π : The *time-progress condition*, is an assertion over V. The assertion is used to specify a global restriction over the progress of time.

The enableness 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 integrators. We call κ the *integrator part* of the enabling condition, and denote it by $En_I(\tau)$.

Example 6. Consider the hybrid system Φ_1 presented in figure 11.

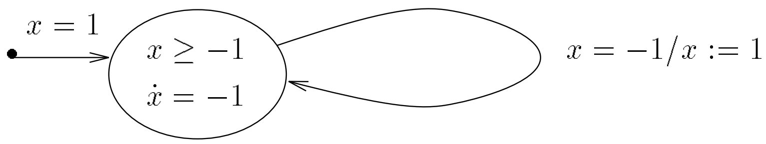

Fig. 11. A hybrid system Φ_1.

The system can be modeled by the following PTS:

$V = I : \{x, T\}$
$\Theta :$ $\quad x = 1 \wedge T = 0$
$\mathcal{T} :$ $\quad \{\tau\}$ where $\rho_\tau : x = -1 \wedge x' = 1 \wedge T' = T$
$\mathcal{A} :$ $\quad \{\alpha\}$ with activity relation (omitting the α subscript and superscript)
$$\underbrace{\mathrm{T}}_{p} \rightarrow \underbrace{x = x^0 - t}_{F(x^0, t)}$$
$\Pi :$ $\quad x \geq -1$

The behavior of this system is (informally) presented in Fig. 12. ▟

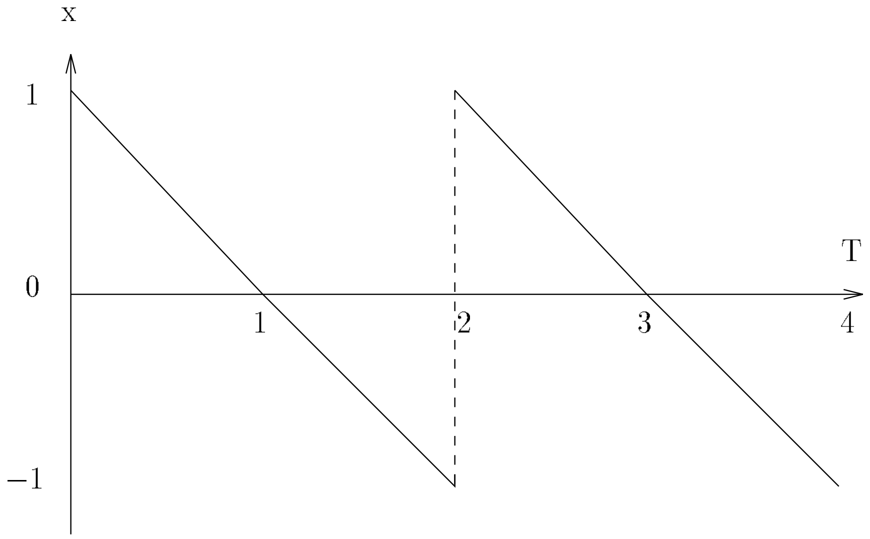

Fig. 12. Behavior of PTS Φ_1.

Extended Transitions

Let $\Phi : \langle V, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$ be a phase transition system. We define the set of *extended transitions* $\mathcal{T}_H$ associated with Φ as follows:

$$\mathcal{T}_H = \mathcal{T} \cup \mathcal{T}_\Phi, \quad \text{where} \quad \mathcal{T}_\Phi = \{\tau_\alpha \mid \alpha \in \mathcal{A}\}$$

For each $\alpha \in \mathcal{A}$, the transition relation of τ_α is given by

$$\rho_{\tau_\alpha}: \quad \exists \Delta > 0 \begin{pmatrix} D' = D \wedge p_\alpha \wedge I' = F^\alpha(V, \Delta) \\ \wedge \\ \forall t \in [0, \Delta).\ \Pi(D, F^\alpha(V, t)) \end{pmatrix}$$

The transition relation ρ_{τ_α} characterizes possible values of the system variables at the beginning and end of an α-phase, where $V = (D, I)$ denotes the values at the beginning of the phase and $V' = (D', I')$ denotes their values at the end of the phase. The formula assumes a positive time increment Δ which will be the length of the phase. It then states that the values of the discrete variables are preserved ($D' = D$), the activation condition p_α currently holds, the values of the integrators at the end of the phase are given by $F^\alpha(V, \Delta)$, and the time-progress condition Π holds for all intermediate time points within the phase, i.e., for all t, $0 \le t < \Delta$.

Runs and Computations

A *run* of a phase transition system $\Phi : \langle V, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$ is an infinite sequence of states $\sigma : s_0, s_1, \ldots$ satisfying:

- *Initiation*: $s_0 \models \Theta$
- *Consecution*: For each $j = 0, 1, ..., s_{j+1} \in \tau(s_j)$, for some $\tau \in \mathcal{T}_H$.

A *computation* of a PTS is a run satisfying:

- *Time Divergence:* The sequence $s_0[T], s_1[T], \ldots$ grows beyond any bound.

Non-Zeno Systems

As in the case of the CTS model, we restrict our attention to non-zeno PTS's. These are systems for which any prefix of a run can be extended to a computation.

System Description by Hybrid Statecharts

Hybrid systems can be conveniently described by an extension of statecharts [Har87] called *hybrid statecharts*. The main extension is

- 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. 13 describes the possible scenarios. This statechart (and the underlying phase transition system) uses the integrators x_m and x_c, measuring the distance of the mouse and the cat, respectively, from the wall. The waiting time of the cat before it starts running is measured by the master clock T. The statechart refers to the constants X_0, v_m, v_c, and Δ.

A behavior of the system starts with states $M.rest$ and $C.rest$ active, variables x_m and x_c set to the initial value X_0, and the master clock T set to 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, using the master clock T to measure this delay. There are two possible termination 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 $X_0 > x_c = x_m > 0$ 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 $X_0 > x_c = x_m > 0$ stand for the conjunctions $x_c = x_m \wedge x_m = 0$ and $X_0 > x_c \wedge x_c = x_m \wedge x_m > 0$, respectively.

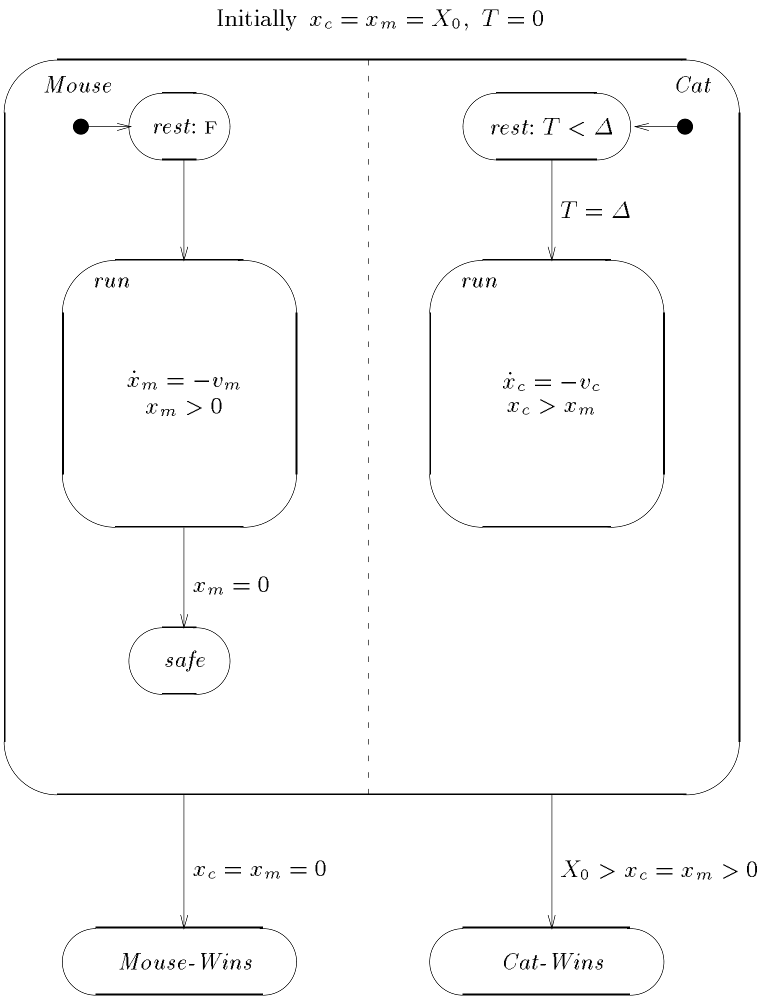

Fig. 13. Specification of Cat and Mouse.

The statechart representation of the Cat and Mouse illustrates the typical interleaving between continuous activities and discrete state changes which, in this example, only involve changes of control.

The Underlying Phase Transition System

Following the graphical representation, we identify the phase transition system underlying the picture of Fig. 13. We refer to states in the diagram that do not enclose other states as *basic states*.

- *System Variables:* $V = D \cup I$, where $D: \{\pi\}$ and $I: \{x_c, x_m, T\}$. Variable π is a control variable whose value is the set of basic states of the statechart which are currently active.
- *Initial Condition:* Given by

$$\Theta: \quad \pi = \{M.rest, C.rest\} \ \wedge \ x_c = x_m = X_0 \ \wedge \ T = 0.$$

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

$$\begin{aligned}
M.\mathit{rest\text{-}run} &: M.rest \in \pi \ \wedge \ \pi' = \pi - \{M.rest\} \cup \{M.run\} \\
C.\mathit{rest\text{-}run} &: C.rest \in \pi \ \wedge \ T = \Delta \ \wedge \ \pi' = \pi - \{C.rest\} \cup \{C.run\} \\
M.\mathit{run\text{-}safe} &: M.run \in \pi \ \wedge \ x_m = 0 \ \wedge \ \pi' = \pi - \{M.run\} \cup \{M.safe\} \\
M.win &: (\mathit{Active} \cap \pi) \neq \phi \ \wedge \ x_c = x_m = 0 \ \wedge \ \pi' = \{\mathit{Mouse\text{-}Wins}\} \\
C.win &: (\mathit{Active} \cap \pi) \neq \phi \ \wedge \ X_0 > x_c = x_m > 0 \ \wedge \ \pi' = \{\mathit{Cat\text{-}Wins}\},
\end{aligned}$$

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

$$\mathit{Active}: \quad \{M.rest,\ M.run,\ M.safe,\ C.rest,\ C.run\}.$$

- *Activities:* It is possible to group all the activities into a single activity, given by:

$$\alpha: \quad x_m = x_m^0 - (at\_M.run)\cdot v_m t \ \wedge \ x_c = x_c^0 - (at\_C.run)\cdot v_c t \ \wedge \ T = T^0 + t.$$

In this representation, we used arithmetization of control expressions by which $at\_M.run$ equals 1 whenever $M.run \in \pi$ and equals 0 at all other instances. A less compact representation lists four activities corresponding to the four cases of: cat and mouse both resting, cat rests and mouse runs, cat runs and mouse is safe, cat and mouse both running.
- *Time Progress Condition:* Given by

$$\Pi: \begin{pmatrix} M.rest \notin \pi \ \wedge \ (C.rest \in \pi \ \rightarrow \ T < \Delta) \ \wedge \\ (M.run \in \pi \ \rightarrow \ x_m > 0) \ \wedge \ (C.run \in \pi \ \rightarrow \ x_c > x_m) \end{pmatrix}$$

4.2 The Fair Transition System induced by a PTS

Let $\Phi: \langle V, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$ be a non-zeno PTS. Consider the following FTS:

$$P_\Phi: \quad \langle V, \Theta, \mathcal{T}_H, \emptyset, \emptyset \rangle,$$

in which the justice and compassion sets are taken to be empty. We refer to P_Φ as the FTS *induced by the* PTS Φ. An important relation between a PTS Φ and the FTS it induces is identified by the following claim:

Claim 4 *A non-zeno* PTS Φ *and the* FTS P_Φ *it induces have identical sets of runs.*

Justification: The claim is a direct consequence of the definition of runs for Φ and P_Φ.

Corollary 5 *An invariance formula* $\square\, p$ *is valid over a non-zeno* PTS Φ *iff it is valid over the induced* FTS P_Φ.

Justification: Identical to that of the CTS$\rightarrow$FTS reduction.

Similarly to Corollary 3, Corollary 5 is not restricted to invariance formulas but holds for all safety properties. It reduces the problem of proving that a safety formula φ (a formula specifying a safety property) is valid over a given (non-zeno) PTS Φ to proving the validity of φ over the FTS P_Φ. Thus, we may use all the techniques and tools developed for verifying safety properties of FTS's for proving safety properties of PTS's.

Verifying Invariance Properties over PTS

As in the case of real-time systems, safety properties of hybrid systems can be verified without explicitly carrying out the reduction described above. Instead, we can use the following H-INV rule.

Rule H-INV

$$\begin{array}{ll} \text{H1.} & \Theta \;\rightarrow\; p \\ \text{H2.} & \{p\}\,\tau\,\{p\} \text{ for every } \tau \in \mathcal{T}_H \\ \hline & \qquad \Phi \models \square\, p \end{array}$$

Rule H-INV is sound and (relatively) complete for proving all invariance properties of non-zeno PTS's.

In a similar way, we can adapt all proof rules for proving safety properties of fair transition systems ([MP95]) to apply to verification of the corresponding safety property over a non-zeno PTS. The only modification necessary is to consider all transitions in $\mathcal{T}_H$, wherever the FTS rule considers all transitions in $\mathcal{T}$.

Verifying a Property of the Cat and Mouse System

Consider the property that, under the assumptions

$$\Delta > 0, \qquad \frac{X_0}{v_m} < \Delta + \frac{X_0}{v_c} \tag{6}$$

all computations of the Cat and Mouse system satisfy

$$\square\big(x_c = x_m \quad \rightarrow \quad x_c = X_0 \,\vee\, x_m = 0\big).$$

This invariant guarantees that the cat can never win.

In Fig. 14, we present a verification diagram of this invariance property. In this diagram we use control assertions indicating 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.

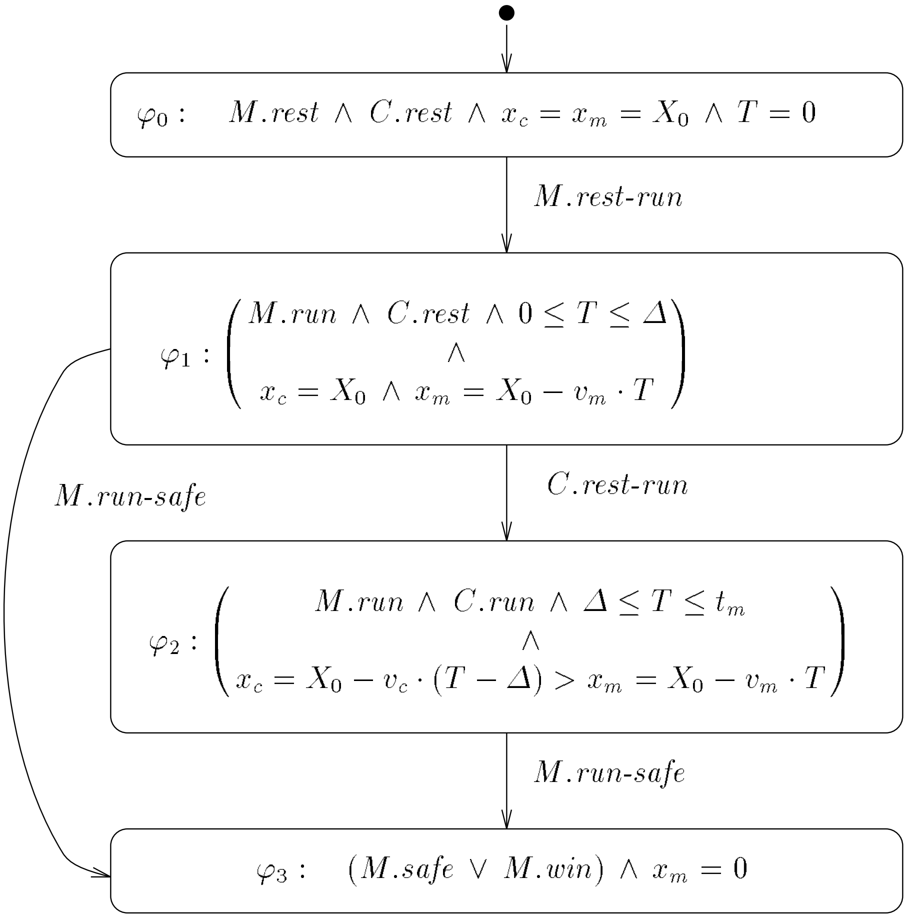

Fig. 14. A hybrid invariance proof diagram.

It is not difficult to verify that the diagram is valid, including the preservation of all assertions under the single activity-induced transition τ_α.

The part that requires some attention is showing that the φ_2 conjunct

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

is maintained until transition *M.run-safe* becomes enabled, that is, as long as x_m is nonnegative. Obviously, $x_m \geq 0$ implies $T \leq t_m$. To show that the conjunct is maintained, it is sufficient to show $v_c \cdot (T - \Delta) < v_m \cdot T$ which is equivalent to

$$\frac{v_m}{v_c} > 1 - \frac{\Delta}{T} \tag{7}$$

From inequality (6), 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

$$\frac{v_m}{v_c} > 1 - \frac{\Delta}{t_m}. \tag{8}$$

Since $T \leq t_m$, we have $1 - \dfrac{\Delta}{t_m} \geq 1 - \dfrac{\Delta}{T}$ establishing (7).

It remains to show that

$$\underbrace{M.rest \ \wedge \ C.rest \ \wedge \ x_c = x_m = X_0 \ \wedge \ y = 0}_{\Theta} \wedge T = 0 \quad \rightarrow \quad \underbrace{M.rest \ \wedge \ C.rest \ \wedge \ x_c = x_m = X_0 \ \wedge \ y = T = 0}_{\varphi_0} \tag{9}$$

$$\varphi_0 \ \vee \ \cdots \ \vee \ \varphi_3 \rightarrow \Big(x_c = x_m \ \rightarrow \ x_c = X_0 \ \vee \ x_m = 0\Big). \tag{10}$$

Implication (9) is obviously valid. To check implication (10), we observe that both φ_0 and φ_1 imply $x_c = X_0$, φ_2 implies $x_c > x_m$ (using the assumption $\Delta > 0$), and φ_3 implies $x_m = 0$.

This shows that under assumption (6), property

$$\square(x_c = x_m \quad \rightarrow \quad x_c = X_0 \vee x_m = 0)$$

is valid for the Cat and Mouse system.

Conclusions

In this paper we have presented the new real-time model of clocked transition system (CTS). This model can be viewed as an extension of the timed automata model [AD94]. In addition to algorithmic verification of finite-state systems, the CTS model can also support deductive verification. In this paper, we have restricted our attention to the verification of safety properties, and have shown a reduction from a CTS to an FTS, a reduction that preserves all the safety properties of the original system. This reduction allows us to use methodologies and tools developed for the FTS model to verify safety properties of real-time systems. In a similar way, we have shown a reduction from the phase transition system model to the FTS model, enabling the reuse of FTS-verification methods for establishing safety properties of hybrid systems.

Acknowledgment

We gratefully acknowledge the extensive help of Yonit Kesten in the preparation of earlier drafts of this paper. The careful reading and critical comments of Oded Maler, Henny Sipma, and Tomás Uribe were most helpful.

References

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theor. Comp. Sci.*, 126:183–235, 1994.

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

[AH92] R. Alur and T. Henzinger. Logics and models of real time: A survey. 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 74–106. Springer-Verlag, 1992.

[AL94] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Trans. Prog. Lang. Sys.*, 16(5):1543–1571, Sept. 1994.

[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.

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

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

[HMP94] T. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Inf. and Comp.*, 112(2):273–337, 1994.

[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.

[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.

[MAB+94] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. De Alfaro, H. Devarajan, H. Sipma, and T. Uribe. Step: The stanford temporal prover. Technical report CS-TR-94-1518, Dept. of Comp. Sci., Stanford University, Stanford, California, 1994.

[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.

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

[MP93a] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

[MP93b] Z. Manna and A. Pnueli. Verifying hybrid systems. In R.L. Grossman, A. Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lect. Notes in Comp. Sci., pages 4–35. Springer-Verlag, 1993.

[MP94] Z. Manna and A. Pnueli. Temporal verification diagrams. In T. Ito and A. R. Meyer, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lect. Notes in Comp. Sci.*, pages 726–765. Springer-Verlag, 1994.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

[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.

[NOS+93] X. Nicollin, A. Olivero, J. Sifakis, , and S. Yovine. An approach to the description and analysis of hybrid systems. In R.L. Grossman, A. Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lect. Notes in Comp. Sci., pages 149–178. Springer-Verlag, 1993.

[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.

[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.