Verifying Clocked Transition Systems*

Yonit Kesten† Zohar Manna‡ 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.
- 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*).

* 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, Weizmann Institute, Rehovot, Israel, e-mail: yonit@wisdom.weizmann.ac.il

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

In [MP93], 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) [MP91b].

The approach to real time presented in [MP93] 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 [MP93], 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.

refer to the two Henzinger's papers

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 the real-time computational model of *clocked transition systems* (CTS). In Section 3, we show how programs augmented with timing bounds for the execution of statements can be represented as clocked transition systems. In Section 4, we present rules and verification diagrams for verifying safety properties of CTS's. In Section 5, we present rules and verification diagrams for establishing response properties of clocked transition systems. Finally, in Section 6, we present an approach to verifying that a given CTS is non-zeno.

2 Real-Time Systems

We now introduce a computational model for real-time systems.

2.1 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 $\tau(s) \subseteq \Sigma$.
  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.
For every $\tau \in \mathcal{T}$, it is required that

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

i.e., the master clock is modified by no transition.
- 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:

$$\rho_{tick}: \quad \exists \Delta.\, \Omega(\Delta) \ \wedge \ D' = D \ \wedge \ C' = C + \Delta,$$

where $\Omega(\Delta)$ is given by

$$\Omega(\Delta): \quad \Delta > 0 \ \wedge \ \forall t \in [0, \Delta).\, \Pi(D, C + t).$$

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 a finite or infinite sequence of states $\sigma : s_0, s_1, \ldots$ satisfying:

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

A state is called (Φ-)*accessible* if it appears in a run of Φ.
A *computation* of Φ is an infinite 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 Frequently Occurring Case

In many cases, the time-progress condition Π has the following special form

$$\Pi\colon \quad \bigwedge_{i \in I} (p_i \rightarrow t^i < E_i),$$

where I is some finite index set and, for each $i \in I$, the assertion p_i and the real-valued expression E_i do not depend on the clocks, and $t^i \in C$ is some clock. This is, for example, the form of the time-progress condition for any CTS representing a real-time program. For such cases, the formula $\Omega(\Delta)$ can be significantly simplified and assumes the following form:

$$\Omega(\Delta)\colon \quad \Delta > 0 \ \wedge \ \bigwedge_{i \in I} (p_i \rightarrow t^i + \Delta \leq E_i)$$

Note, in particular, that this simpler form does not use quantifications over t.

Non-Zeno Systems

A CTS is defined to be *non-zeno* if every finite run can be extended into a computation (see [AL94], [Hen92]). An equivalent formulation is that Φ is non-zeno if it satisfies

> A finite sequence σ is a run of Φ iff σ is a prefix of some computation of Φ.

A consequence of Φ being non-zeno is that a state s is Φ-accessible iff it appears in some computation of Φ.

Example 1. Consider the CTS's Φ_1 and Φ_2 presented in Fig 1. In Fig 2, we present these two CTS's in textual form. The predicate $pres(y, t, T)$ is an abbreviation for the assertion

$$pres(y, t, T)\colon \quad y' = y \ \wedge \ t' = t \ \wedge \ T',$$

stating that y, t, and T, are preserved by the transition.

It is not difficult to establish that both Φ_1 and Φ_2 are non-zeno CTS's. This is becaues, from any accessible state, we can always move to state ℓ_1 from which we can continue to take infinitely many time steps with increment 1.

The *tick* transitions for these two CTS's are given by

$$\rho^1_{tick}\colon \exists \Delta > 0.\, pres(\pi, y) \ \wedge \ (t', T') = (t + \Delta, T + \Delta) \ \wedge \ (\pi = 0 \rightarrow t + \Delta \leq 2)$$
$$\rho^2_{tick}\colon \exists \Delta > 0.\, pres(\pi, y) \ \wedge \ (t', T') = (t + \Delta, T + \Delta) \ \wedge \ (\pi = 0 \rightarrow t + \Delta \leq \tfrac{1}{2^{y+1}}).$$

Following is a computation of CTS Φ_1:

$$\begin{array}{l}
\langle \pi\colon 0\,,\, y\colon 0\,,\, t\colon 0\,,\, T\colon 0 \rangle \xrightarrow{tick(1)} \langle \pi\colon 0\,,\, y\colon 0\,,\, t\colon 1\,,\, T\colon 1 \rangle \xrightarrow{\tau_0} \\
\langle \pi\colon 0\,,\, y\colon 1\,,\, t\colon 1\,,\, T\colon 1 \rangle \xrightarrow{\tau_0} \langle \pi\colon 0\,,\, y\colon 2\,,\, t\colon 1\,,\, T\colon 1 \rangle \xrightarrow{tick(1)} \\
\langle \pi\colon 0\,,\, y\colon 2\,,\, t\colon 2\,,\, T\colon 2 \rangle \xrightarrow{\tau_0} \langle \pi\colon 0\,,\, y\colon 3\,,\, t\colon 2\,,\, T\colon 2 \rangle \xrightarrow{\tau_0} \\
\langle \pi\colon 0\,,\, y\colon 4\,,\, t\colon 2\,,\, T\colon 2 \rangle \xrightarrow{\tau_1} \langle \pi\colon 1\,,\, y\colon 4\,,\, t\colon 2\,,\, T\colon 2 \rangle \xrightarrow{tick(1)} \\
\langle \pi\colon 1\,,\, y\colon 4\,,\, t\colon 3\,,\, T\colon 3 \rangle \xrightarrow{tick(1)} \langle \pi\colon 1\,,\, y\colon 4\,,\, t\colon 4\,,\, T\colon 4 \rangle \xrightarrow{tick(1)} \\
\qquad \cdots
\end{array}$$

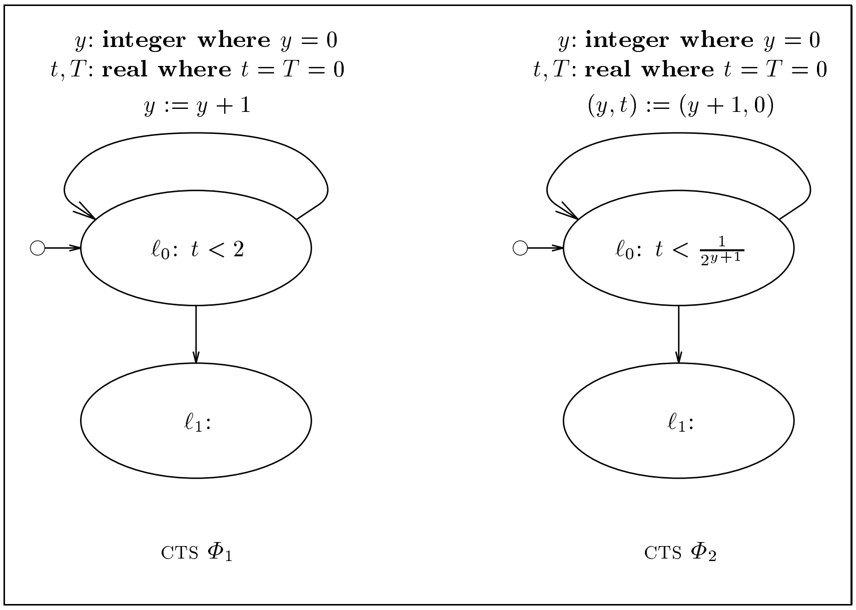

Fig. 1. Two CTS's.

Note that to be a computation, time must grow beyond any bounds. Since, at location ℓ_0 of CTS time cannot grow beyond 2, any computation of Φ_1 must eventually move to location ℓ_1, where time can grow beyond any bounds. ▪

3 Programs as Clocked Transition Systems

In this section we show how to represent real-time programs as clocked transition systems.

In [MP91b], we introduced a simple programming language SPL. Here we consider a subset of the language, restricting our attention to the following statements:

> *skip, assignment, await, noncritical, critical, conditional, concatenation, selection, while,* and *block*.

Not including the *cooperation* statement has the effect that parallelism is allowed only at the top level of the program. That is, a program is a parallel copmposition of processes, each of which is sequential.

We refer the reader to Chapter 0 of [MP95] for the construction of a fair transition system F_P corresponding to an SPL program P. This construction associates a single transition with each *executable* statement, which include all statements except *concatenation*, *selection*, and *block*. Here, we deviate from the

V: $\underbrace{\{\pi:\{0,1\}; y:\mathbf{integer}\}}_{D} \cup \underbrace{\{t,T:\mathbf{real}\}}_{C}$ Θ: $\pi = y = t = T = 0$ $\mathcal{T}$: $\{\tau_0, \tau_1\}$with transition relations τ_0: $\pi = \pi' = 0 \wedge y' = y + 1 \wedge pres(t,T)$ τ_1: $\pi = 0 \wedge \pi' = 1 \wedge pres(y,t,T)$ Π: $\pi = 0 \rightarrow t < 2$ CTS Φ_1	V: $\underbrace{\{\pi:\{0,1\}; y:\mathbf{integer}\}}_{D} \cup \underbrace{\{t,T:\mathbf{real}\}}_{C}$ Θ: $\pi = y = t = T = 0$ $\mathcal{T}$: $\{\tau_0, \tau_1\}$with transition relations τ_0: $\pi = \pi' = t' = 0 \wedge y' = y + 1 \wedge pres(T)$ τ_1: $\pi = 0 \wedge \pi' = 1 \wedge pres(y,t,T)$ Π: $\pi = 0 \rightarrow t < \frac{1}{2^{y+1}}$ CTS Φ_2

Fig. 2. The two CTS's in textual form.

construction of [MP95] in the transition associated with the *await* statement. Instead, we associate with each statement of the form

ℓ: **await** c; $\quad \widehat{\ell}$:

a transition τ_ℓ, whose transition relation ρ_ℓ is given by

$$\rho_\ell: \quad \ell \in \pi \wedge \begin{pmatrix} c \wedge \pi' = \pi - \{\ell\} \cup \{\widehat{\ell}\} \\ \vee \\ \neg c \wedge \pi' = \pi \end{pmatrix} \wedge pres(V - \{\pi\}).$$

Otherwise than the *await* statement, we follow [MP95] in the association of transitions with executable statements.

Another difference from [MP95] is that we only associate transitions with the executable statements and do not add the idling transition τ_I.

Let P be an SPL program. To obtain a real-time program, we associate with each S, an executable statement of P, a pair of values $[l_S, u_S]$, called the *lower* and *upper* bounds of S. These values, satisfying $0 \leq l_S \leq u_S \leq \infty$, are intended to provide a lower and upper bound on the length of time the statement can be enabled without being taken. We refer to a program with an assignment of time bounds as an SPL$_T$ program, and view it as a real-time program.

Note that assigning time bounds to each executable statement of P is equivalent to assigning time bounds $[l_\tau, u_\tau]$ to each transition τ associated with an executable statement.

3.1 The CTS corresponding to an SPL$_T$ program

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

As a first step, we construct the FTS $F_P: \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ corresponding to program P. Note that if program P uses only the statements in the previously presented subset, then $\mathcal{J} = \mathcal{T} - \{\tau_I\}$ and $\mathcal{C} = \emptyset$. Assume that program P is a parallel composition of m processes. That is, P has the form

declarations; $\quad P_1 \parallel \cdots \parallel P_m,$

where each P_i is a (sequential) statement.

The CTS corresponsing to P is given by $\Phi_P: \langle \widetilde{V}, \widetilde{\Theta}, \widetilde{\mathcal{T}}, \Pi \rangle$, where

- *System Variables:* $\widetilde{V} = V \cup \{t_1, \ldots, t_m, T\}$.
  Thus, $\widetilde{V}$ consists of the system variables of F_P, to which we add $m+1$ clocks, one clock t_i for each process P_i, $i = 1, \ldots, m$, plus the master clock T.
- *Initial Condition:* $\widetilde{\Theta}$: $\quad \Theta \ \wedge \ t_1 = \cdots = t_m = T = 0.$
- *Transitions:* $\widetilde{\mathcal{T}}: \{\widetilde{\tau} \mid \tau \in \mathcal{T} - \{\tau_I\}\}$.
  Let $\tau \in \mathcal{T} - \{\tau_I\}$ be a transition corresponding to an executable statement in process P_i. The transition relation for transition $\widetilde{\tau}$ is given by:

  $$\widetilde{\rho_\tau}: \quad \rho_\tau \ \wedge \ t_i \geq l_\tau \ \wedge \ t_i' = 0 \ \wedge \ \mathit{pres}(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m, T).$$

  Thus, the transition can be taken only when t_i, the clock correponding to process P_i, is not below l_τ, the lower bound associated with τ. When taken, the transition resets clock t_i to 0.
- *Time-progress condition Π:* For each executable statement

  $$\ell: S$$

  in process P_i, Π includes the conjunct

  $$\ell \in \pi \quad \rightarrow \quad t_i < u_S,$$

  where u_S is the upper bound associated with statement S. This ensures that control cannot wait at location ℓ for more than u_S without the transition associated with S (or another transition causing control to move away from ℓ) being taken.

Note that the lower bounds of statements are added as constraints to transitions, while the upper bounds are added as constraints to the time-progress condition Π.

Example 2. Figure 3 presents a simple program consisting of two processes communicating by the shared variable x.

To make it an SPL$_T$ program, we uniformly associate each of its executable statements with the time bounds $[3,5]$. 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:*

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

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

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

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

Fig. 3. Program ANY-Y.

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

$$\rho_{\ell_0} : \ \ell_0 \in \pi \ \wedge \left(\begin{array}{c} x = 0 \ \wedge \ \pi' = \pi - \{\ell_0\} \cup \{\ell_1\} \\ \vee \\ x \neq 0 \ \wedge \ \pi' = \pi - \{\ell_0\} \cup \{\ell_2\} \end{array}\right) \wedge \ t_1 \geq 3 \ \wedge \ t_1' = 0$$
$$\wedge \, pres(V - \{\pi, t_1\})$$

$$\rho_{\ell_1} : \ \ell_1 \in \pi \ \wedge \ \pi' = \pi - \{\ell_1\} \cup \{\ell_0\} \ \wedge \ y' = y + 1 \ \wedge \ t_1 \geq 3 \ \wedge \ t_1' = 0$$
$$\wedge \ pres(V - \{\pi, y, t_1\})$$

$$\rho_{m_0} : m_0 \in \pi \ \wedge \ \pi' = \pi - \{m_0\} \cup \{m_1\} \ \wedge \ x' = 1 \ \wedge \ t_2 \geq 3 \ \wedge \ t_2' = 0$$
$$\wedge \ pres(V - \{\pi, x, t_2\}).$$

- *Time-progress condition:*

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

The *tick* transition for this system is given by

$$\begin{array}{l} tick\text{:} \quad \exists \Delta > 0.\ pres(\pi, x, y) \ \wedge \ (t_1', t_2', T') = (t_1 + \Delta, t_2 + \Delta, T + \Delta) \ \wedge \\ \qquad\qquad (at\_\ell_{0,1} \ \rightarrow \ t_1 + \Delta \leq 5) \ \wedge \ (at\_m_0 \ \rightarrow \ t_2 + \Delta \leq 5) \end{array} \quad \blacksquare$$

3.2 Specification Language

To specify properties of reactive systems, we use the language of temporal logic, as presented in [MP91b]. We use only the following:

- *State formulas* (*assertions*) - any first-order formula. For locations ℓ_i, ℓ_j, and ℓ_k, we denote:
$$\begin{array}{l} at\_\ell_i \quad : \ell_i \in \pi \\ at\_\ell_{j,k} : at\_\ell_j \vee at\_\ell_k \end{array}$$
- $\Box\, p$ — Always p, where p is an assertion. We refer to such a formula as an *invariance formula*.
- $p \Rightarrow (q\, \mathcal{W}\, r)$ — p entails q waiting for r, where p, q, and r are assertions. We refer to such a formula as a *waiting-for formula*.
- $p \Rightarrow \Diamond\, r$ — p entails eventually r, where p and r are assertions. We refer to such a formula as a *response formula*.

For a state s and assertion p, we write $s \Vdash p$ to indicate that p holds (is true) over s. Let $\sigma : s_0, s_1 \ldots$ be an infinite sequence of states, to which we refer as a *model*. For an assertion p, we say that $j \geq 0$, is a *p-position* if $s_j \models p$. Satisfaction of (the three considered) temporal formulas over a model σ is defined as follows:

- A model σ satisfies the invariance formula $\square\, p$, written $\sigma \models \square\, p$, if all positions within σ are p-positions.
- A model σ satisfies the waiting-for formula $p \Rightarrow (q\, \mathcal{W}\, r)r$, written $\sigma \models p \Rightarrow (q\, \mathcal{W}\, r)$, if every p-position i within σ initiates an interval of positions, all of which satisfy q. This continuous-q interval can either extend to infinity or terminate in an r-position which is not in the interval. That is,

$$\begin{array}{lll} \sigma[i] \Vdash p & \text{implies} & \sigma[j] \Vdash q \text{ for all } j \geq i, \text{ or} \\ & & \sigma[k] \Vdash r \text{ for some } k \geq i \text{ and } \sigma[j] \Vdash q \text{ for all } j,\, i \leq j < k. \end{array}$$

- A model σ satisfies the response formula $p \Rightarrow \diamondsuit\, r$, written $\sigma \models p \Rightarrow \diamondsuit\, r$, if every p-position i within σ is followed by an r-position $j \geq i$.

A temporal formula φ is said to be *valid over* CTS Φ (or *Φ-valid*) if $\sigma \models \varphi$ for every computation σ of Φ. We write $\Phi \models \varphi$ to indicate this fact. An assertion p is called *Φ-state valid* if it holds at every Φ-accessible state. We write $\Phi \Vdash p$ to indicate that assertion p is Φ-state valid.

In the case that CTS Φ is derived from an SPL$_T$ program P, we use the terms P-valid, P-state valid, and P-accessible as synonymous to Φ_P-valid, Φ_P-state valid, and Φ_P-accessible. We also write $P \models \varphi$ and $P \Vdash p$ to denote $\Phi_P \models \varphi$ and $\Phi_P \Vdash p$, respectively.

4 Verifying Safety Properties of Clocked Transition Systems

In this section, we present methods for verifying safety properties of clocked transitions systems.

4.1 The Invariance Rule

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 *invariance* properties, namely, properties that can be expressed by the formula $\square\, p$, for some assertion p.

For assertions φ and p,

$$\begin{array}{ll} \text{A1.} & \varphi \rightarrow p \\ \text{A2.} & \Theta \rightarrow \varphi \\ \text{A3.} & \rho_\tau \wedge \varphi \rightarrow \varphi' \quad \text{for every } \tau \in \mathcal{T}_T \\ \hline & \Phi \models p \end{array}$$

Fig. 4. Rule ACC (Φ-state validity of assertion p).

The Accessibility Rule

As a preliminary step, we introduce a rule that establishes the Φ-validity of an assertion p. This is rule ACC, presented in Fig. 4.

The rule uses an auxiliary assertion φ. Premise A1 of rule ACC requires that the auxiliary assertion φ implies assertion p, whose Φ-state validity we wish to prove. Premise A2 of the rule requires that Θ, the initial condition of P, implies the auxiliary assertion φ. Premise A3 requires that all transitions in $\mathcal{T}_T^\Phi$ (extended transitions of Φ) preserve φ. Premises A2 and A3 state that φ holds at the initial state of every run and that it propagates from any state to its τ-successor, for every transition $\tau\mathcal{T}_T$ of the system. Thus, every state in each run of Φ satisfies φ. Due to the implication A1, every such state also satisfies p. It follows that p holds on every accessible state of system Φ and, therefore, assertion p is Φ-state valid.

Rule INV

Assume that we have shown that assertion p is Φ-state valid, i.e., every accessible state of Φ satisfies p. Since all states appearing in a computation are accessible (every computation is a run), all states in every computation satisfy p. It follows that every computation of Φ satisfies $\Box\, p$. Thus, we may use the premises of rule ACC to establish that the temporal formula $\Box\, p$ is Φ-valid. Consequently, we propose rule INV, presented in Fig. 5, as the main tool for verifying invariance properties of a CTS Φ.

Example 3. We use rule INV to establish the invariance of the assertion

$$p: \quad at\_\ell_{0,1} \vee at\_m_1$$

over program ANY-Y (Fig. 3). 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, 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.

We apply rule INV to this choice of p, taking

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

For assertions φ and p,

$$\begin{array}{ll} \text{I1.} & \varphi \;\rightarrow\; p \\ \text{I2.} & \Theta \;\rightarrow\; \varphi \\ \text{I3.} & \rho_\tau \;\wedge\; \varphi \quad\rightarrow\quad \varphi' \quad \text{for every } \tau \in \mathcal{T}_T \\ \hline & \Phi \models \Box\, p \end{array}$$

Fig. 5. Rule INV (invariance) applied to CTS Φ.

as the auxiliary assertion.

Premise I1 assumes the form

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

which is obviously valid.

Premise I2 assumes the form

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

which is obviously valid.

Premise I3 has to be checked for each $\tau \in \{\ell_0, \ell_1, m_0, tick\}$. For example, premise I3 for ℓ_0 assumes the form

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

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

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

This establishes the P-validity of the invariant $\Box\, p$, i.e.,

$$P \models \Box(at\_\ell_{0,1} \;\vee\; at\_m_1). \quad ■$$

For assertions p and q and transition τ, we introduce the abbreviation

$$\{p\}\tau\{q\} \qquad \text{standing for} \qquad \rho_\tau \wedge p \;\rightarrow\; q'.$$

Invariance Verification Diagrams

An effective way of presenting the verification of an invariance property by rule INV 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. 6 we present a verification diagram for program ANY-Y.

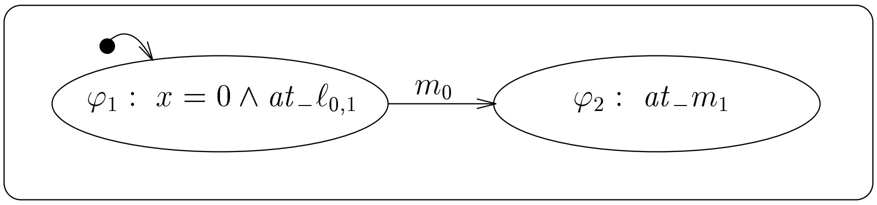

Fig. 6. Verification Diagram D_1

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 I2 of rule INV is given by

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

- Let φ be a node in the graph, τ be a 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 (corresponding to premise I3) 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. 6), are:

I2:	$\Theta \rightarrow\ x = 0 \wedge at\_\ell_{0,1}$
I3 for φ_1 and ℓ_0 :	$\{x = 0 \wedge at\_\ell_{0,1}\}\, \ell_0\, \{x = 0 \wedge at\_\ell_{0,1}\}$
I3 for φ_1 and ℓ_1 :	$\{x = 0 \wedge at\_\ell_{0,1}\}\, \ell_1\, \{x = 0 \wedge at\_\ell_{0,1}\}$
I3 for φ_1 and m_0 :	$\{x = 0 \wedge at\_\ell_{0,1}\}\, m_0\, \{(x = 0 \wedge at\_\ell_{0,1}) \vee at\_m_1\}$
I3 for φ_1 and *tick* :	$\{x = 0 \wedge at\_\ell_{0,1}\}\, tick \{x = 0 \wedge at\_\ell_{0,1}\}$
I3 for φ_2 and ℓ_0 :	$\{at\_m_1\}\, \ell_0\, \{at\_m_1\}$
I3 for φ_2 and ℓ_1 :	$\{at\_m_1\}\, \ell_1\, \{at\_m_1\}$
I3 for φ_2 and m_0 :	$\{at\_m_1\}\, m_0\, \{at\_m_1\}$
I3 for φ_2 and *tick* :	$\{at\_m_1\}\, tick \{at\_m_1\}$

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. 7.

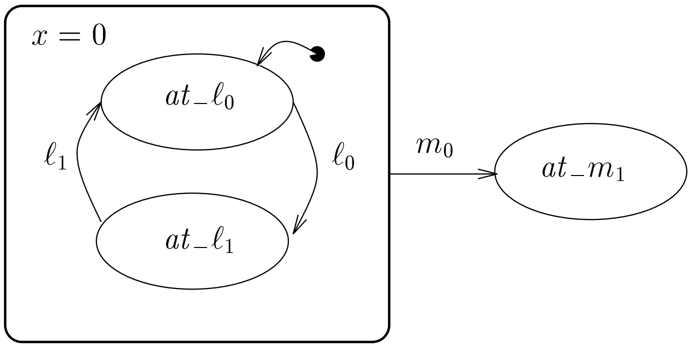

Fig. 7. A more detailed diagram, using encapsulation conventions

In the diagram of Fig. 7, 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.

We also use invariance diagrams to provide succinct representations of proofs by rule INV.

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

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

This formula claims that every accessible state in which the program has not terminated yet (i.e., $at\_\ell_2 \wedge at\_m_1$ does not hold), can only be observed when the master clock T has not yet passed 15. It follows that any state observed later than $T = 15$ must be a termination state.

The proof is presented by the invariance diagram in Fig. 8.
The assertions are indexed in descending order from φ_3 on top to φ_0 at bottom in order to keep compatibility with the chain diagrams introduced in Section 5.

Note that no edge in the diagram is labeled by the *tick* transition. This implies that all verification conditions involving the *tick* transition are of the form $\{\varphi_i\}\ tick\,\{\varphi_i\}$ claiming that the *tick* transition preserves each of the assertions appearing in the diagram.

As an example, consider the verification condition claiming that assertion

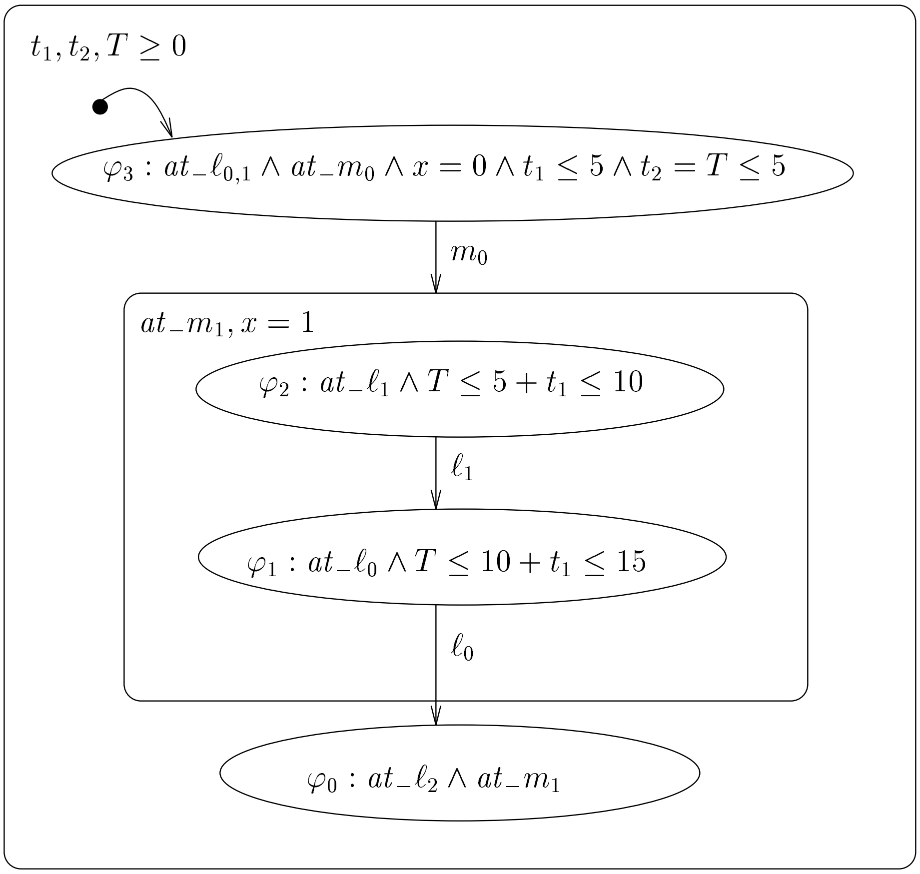

Fig. 8. Termination of ANY-Y within 15 time units.

φ_1: $at\_\ell_1 \wedge T \leq 5 + t_1 \leq 10$ is preserved by the *tick* transition.

$$\underbrace{at\_\ell_1 \wedge T \leq 5 + t_1 \leq 10}_{\varphi_1} \quad \wedge$$
$$\underbrace{\Delta > 0 \wedge \pi' = \pi \wedge T' = T + \Delta \wedge t_1' = t_1 + \Delta \wedge (at\_\ell_1 \rightarrow t_1 + \Delta \leq 5) \wedge \cdots}_{\rho_{tick}}$$
$$\rightarrow \quad \underbrace{at'\_\ell_1 \wedge T' \leq 5 + t_1' \leq 10}_{\varphi_1'}$$

It is not difficult to see that this implication is state-valid. ▟

Example 5. As a second example, we present 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. 9 under the name of program MUTEX. By assigning all statements in MUTEX the uniform time bounds $[L, U]$, we obtain the timed program $\text{MUTEX}_{[L,U]}$. Assuming that $2L > U$, we prove that the mutual exclusion property

$$\square \neg(at\_\ell_8 \wedge at\_m_8)$$

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

```
                         x: integer where x = 0

 ℓ0 : loop forever do                      m0 : loop forever do
    ℓ1 : noncritical                          m1 : noncritical
    ℓ2 : skip                                 m2 : skip
    ℓ3 : while x ≠ 1 do                       m3 : while x ≠ 2 do
        ℓ4 : await x = 0                  ||      m4 : await x = 0
        ℓ5 : x := 1                               m5 : x := 2
        ℓ6 : skip                                 m6 : skip
        ℓ7 : if x = 1 then                        m7 : if x = 2 then
               ℓ8 : critical                             m8 : critical
    ℓ9 : skip                                 m9 : skip
    ℓ10 : x := 0                              m10 : x := 0

            – P1 –                                    – P2 –
```

Fig. 9. Program MUTEX, implementing Fischer's protocol.

Let us explain informally why mutual exclusion for program MUTEX is guaranteed and the role of the *skip* statements at ℓ_6 and m_6. Assume a violation of mutual exclusion in which process P_1 ewntered its critical section (location ℓ_8) first and, while P_1 is still there, process P_2 enters m_8. When P_1 entered ℓ_8, x was 1. For P_2 to enter m_8 later, it was necessary for it to set x to 2 first, which can only be done at m_5. Since after P_1 set x to 1 at ℓ_5, P_2 cannot pass the test for $x = 0$ at m_4, the only possibility is that P_2 kept waiting at m_5 while P_1 executed ℓ_6 and the test at ℓ_7. This must have taken P_1 at least $2L$, since L is the lower bound for execution of a statement. However, P_2 cannot wait at m_5 for as long as $2L$ because $2L > U$ and no process can delay the execution of a statement for more than U. It follows that the described scenario in which P_2 keep waiting at m_5 until P_1 enters ℓ_8 is impossible. The role of the *skip* statement at ℓ_6 is therefore to introduce an additional delay of at least L time units.

The *skip* statements at ℓ_9 and m_9 are necessary to guarantee the response properties of this algorithm, and we will discuss them in the next section. The *skip* statements at ℓ_2 and m_2 represent exits out of the noncritical sections (at ℓ_1 and m_1, respectively) and intention to enter the critical sections.

The property of mutual exclusion for program MUTEX can be stated by the invariance formula

$$\square \neg(at\_\ell_8 \ \wedge \ at\_m_8).$$

A formal proof of this invariant is presented by the invariance diagram of Fig. 10.

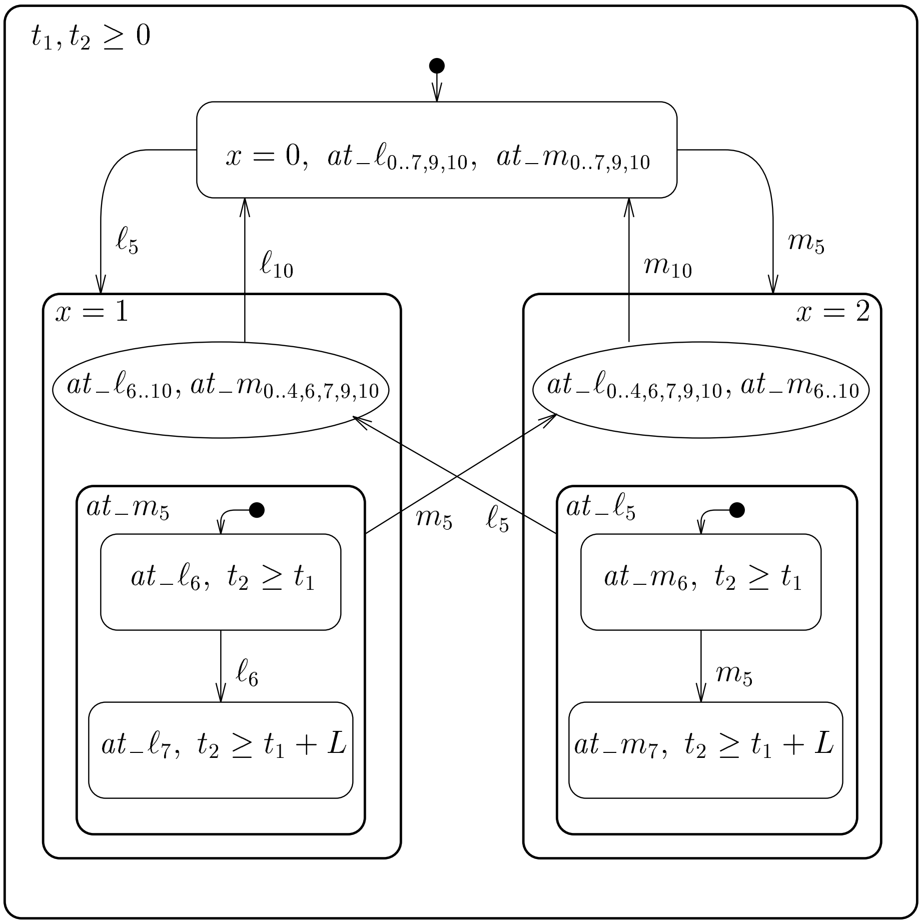

Fig. 10. Mutual exclusion for program MUTEX.

Note that the edge labeled ℓ_5 reaches the compound node labeled $x = 1$. By the encapsulation conventions, this is equivalent to having this edge connect to the basic node labeled $at\_\ell_{6..10},\ at\_m_{0..4,6,7,9,10}$ and the compound node labeled at_m_5. While reaching this compound node, it should enter some of its contained basic nodes. The default-entry arrow marks the basic node labeled $at\_\ell_6,\ t_2 \geq t_1$ as the single node entered by the ℓ_5 edge. Thus, the ℓ_5 edge connects to the basic node labeled $at\_\ell_{6..10},\ at\_m_{0..4,6,7,9,10}$ and the one labeled $at\_\ell_6,\ t_2 \geq t_1$. ▟

Completeness of Rule INV

Rule INV is complete for verifying invariances of fair transition systems. This is stated by the following claim.

Claim 2 *If formula* $\square\, p$ *is valid over non-zeno* CTS P*, then there exists an assertion* φ *such that premises* I1-I3 *of rule* INV *are state-valid.*

Justification The basic idea of the proof is the construction of an assertion acc_P that holds in a state s iff s is P-accessible, i.e., appears in some run of P. We then show that if $\square\, p$ is P-valid, then the premises of rule INV are state-valid (implying that they are also P-state valid) when taking acc_P for φ. Since we are only proving *relative* completeness, it is enough to show validity of the premises, assuming an oracle that provides proofs or otherwise verifies all generally valid assertions.

For the construction of acc_P, we refer the reader to [MP91a] or Chapter 2 of [MP95]. Assume that we have constructed an assertion acc_P such that

$$s \models acc_P \qquad \text{iff} \qquad s \text{ is a } P\text{-accessible state.}$$

We show that acc_P, when substituted for φ, validates the three premises of rule INV.

I1. $acc_P \rightarrow p$
By our assumption that $\square\, p$ is P-valid, it follows that each state appearing in some computation satisfies p. As P is non-zeno, every accessible state appears in some computation and, hence, every accessible state satisfies p. Since acc_P characterizes precisely the accessible states, the premise follows.

I2. $\Theta \rightarrow acc_P$
It is obvious that every state satisfying Θ is initial and is, therefore, P-accessible. Consequently, such a state must satisfy acc_P, which characterizes all P-accessible states.

I3. $\rho_\tau(U,\widetilde{U}) \wedge acc_P(U) \quad \rightarrow \quad acc_P(\widetilde{U})$, for each $\tau \in \mathcal{T}$ and every values of U and $\widetilde{U}$.
Let U and $\widetilde{U}$ be two lists of values, which can be viewed as values of the system variables V, such that both $\rho_\tau(U,\widetilde{U})$ and $acc_P(U)$ are true. We will show that $acc_P(\widetilde{U})$ is also true.
Let s and $\widetilde{s}$ be two arbitrary states such that $s[V] = U$ and $\widetilde{s}[V] = \widetilde{U}$. Since $acc_P(U) = \text{T}$, we have that $s \models acc_P$. By the meaning of acc_P, it follows that s is P-accessible. From $\rho_\tau(U,\widetilde{U}) = \text{T}$, it follows that $\langle s, \widetilde{s}\rangle \models \rho_\tau(V,V')$ and, therefore, that $\widetilde{s}$ is a τ-successor of s. Since s is accessible, it follows that also $\widetilde{s}$ is accessible and, hence $\widetilde{s} \models acc_P$, leading to $acc_P(\widetilde{U}) = \text{T}$

This concludes the proof of completeness of rule INV. ∎

The following example demonstrates that the assumption that P is non-zeno is essential to the completeness of the rule.

Example 6. Consider the CTS Φ_3 which is presented graphically in Fig. 11 and textually in Fig. 12.

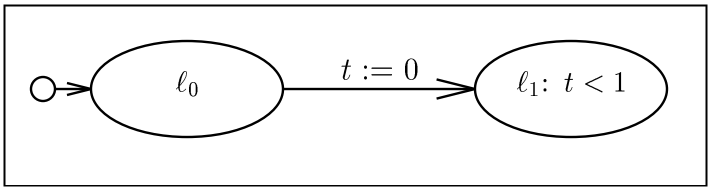

Fig. 11. CTS Φ_3.

Clearly, CTS Φ_3 is a possibly-zeno system because, once a run enters location ℓ_1, time is bounded and cannot diverge. Thus, a run entering location ℓ_1 cannot be extended into a computation of Φ_3. A result of the fact that no computation of Φ_3 ever enters location ℓ_1 is that the invariance formula $\square\ at\_\ell_0$ is Φ_3-valid. However, examination of the arguments for the soundness of rule INV shows

$V\colon \{\pi\colon \{0,1\}; t, T\colon \mathbf{real}\}$
$\Theta\colon \pi = t = T = 0$
$\mathcal{T}\colon \{\tau_{01}\}$ with transition relations
  $\tau_{01}\colon \pi = 0 \,\wedge\, \pi' = 1 \,\wedge\, t' = 0 \,\wedge\, T' = T$
$\Pi\colon \pi = 1 \rightarrow t < 1$

Fig. 12. CTS Φ_3 in textual form.

that any assertion p, whose invariance is proven by rule INV, must hold on all accesible states. The state $\langle \pi\colon 1\rangle$ is accessible for CTS Φ_3 but does not satisfy at_ℓ_0. Therefore, the Φ_3 validity of $\square\ at\_\ell_0$ cannot be proven by rule INV. ∎

4.2 Verifying Waiting-for Formulas

The invariance formula $\square\, q$ states that q holds continuously from the beginning of the computation to infinity. In comparison, the waiting-for formula

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

also states the continuous holding of q but only for an interval that is initiated by an occurrence of p and may be terminated by an occurrence of r. Such formulas are useful for expressing timing properties, where time is measured since an occurrence of an event, rather than from the beginning of a computation.

Example 7. Consider program CYCLIC, presented in Fig. 13. This program

$$
\begin{array}{c}
x, y\text{: } \textbf{integer where } x = y = 0 \\[1em]
\left[\begin{array}{l}
\ell_0 : \ \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1 : \textbf{while } x = 0 \textbf{ do} \\
\quad \ell_2 : \ y := y + 1 \\
\ell_3 : x := 1
\end{array}\right]
\end{array}\right]
\quad \| \quad
\left[\begin{array}{l}
m_0 : \ \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
m_1 : x := 1 \\
m_2 : \textbf{await } x = 0
\end{array}\right]
\end{array}\right] \\[1em]
- \ P_1 \ - \qquad\qquad\qquad - \ P_2 \ -
\end{array}
$$

Fig. 13. Program CYCLIC.

can be viewed as a generalization of program ANY-Y, in which the basic interaction between processes P_1 and P_2 is embedded within an endless loop.

Since program CYCLIC never terminates, a relevant time-related property is that from any state in which we observe P_1 at ℓ_1 and P_2 at m_1, P_1 will reach location ℓ_3 within at most 15 time units.

This property can be stated by the following waiting-for formula:

$$at\_\ell_1 \wedge at\_m_1 \wedge x = 0 \wedge T = a \quad \Rightarrow \quad (T \leq a + 15)\, \mathcal{W}\, at\_\ell_3.$$

This formulas uses a rigid variable a to record the time at the initial observation state. It claims that from the time we observe a state satisfying $at\_\ell_1 \wedge at\_m_1 \wedge x = 0$, time will not progress by more than 15 units before we observe P_1 at ℓ_3.

To facilitate the expression of such properties, we introduce the abbreviation

$$T_a\text{:} \quad T - a.$$

With this abbreviation we can write the property of "reaching ℓ_3 within 15 time units" as

$$at\_\ell_1 \wedge at\_m_1 \wedge x = 0 \wedge T_a = 0 \quad \Rightarrow \quad (T_a \leq 15)\, \mathcal{W}\, at\_\ell_3.$$

The T_a abbreviation allows us to view T_a as a special timer, reset at the initial point of observation. ▟

A Waiting-for Rule

To establish the P-validity of a waiting-for formula, we may use rule WAIT, presented in Fig. 14.

For assertions p, q, φ, and r,

$$\begin{array}{ll} \text{U1.} & \varphi \rightarrow q \\ \text{U2.} & p \rightarrow r \vee \varphi \\ \text{U3.} & \rho_\tau \wedge \varphi \rightarrow r' \vee \varphi' \quad \text{for every } \tau \in \mathcal{T}_T \\ \hline & P \models p \quad \Rightarrow \quad q \, \mathcal{W} \, r \end{array}$$

Fig. 14. Rule WAIT (waiting-for) applied to CTS P.

Waiting Diagrams

Proofs by rule WAIT can be succinctly represented by special verification diagrams to which we refer as *waiting diagrams*.

A waiting diagram is a directed labeled graph constructed as follows:

- *Nodes* in the graph are labeled by assertions $\varphi_0, \ldots, \varphi_m$. We will often refer to a node by the assertion labeling it.
  The node labeled by φ_0 is called the *terminal node* and is graphically identified by being drawn with bold-face boundaries.
- *Edges* in the graph represent transitions between assertions. Each edge is labeled by the name of a transition in the program. We refer to an edge labeled by τ as a τ-*edge*. No edges depart form the terminal node.
- Some of the nodes are designated as *initial nodes*. They are annotated by an entry arrow.

Verification Conditions Implied by a Chain Diagram

Consider a nonterminal node labeled by assertion φ. Let $\tau \in \mathcal{T}_T$ be a transition and let $\varphi_1, \ldots, \varphi_k$, $k \geq 0$, be the successors of φ by edges labeled with τ (possibly including φ itself). With each such node and transition, we associate the following verification condition:

$$\rho_\tau \wedge \varphi \rightarrow \varphi' \vee \varphi_1' \vee \cdots \vee \varphi_k'.$$

In particular, if $k = 0$ (i.e., φ has no τ-successors), the associated verification condition is

$$\rho_\tau \wedge \varphi \rightarrow \varphi'.$$

Valid Waiting Diagrams

A waiting diagram is said to be *valid over* CTS P (P-*valid* for short) if all the verification conditions associated with the diagram are P-state valid.

The consequences of having a valid chain diagram are stated in the following claim:

Claim 3 *If D is a P-valid waiting diagram with nodes $\varphi_0, \ldots, \varphi_m$, then*

$$P \models \bigvee_{j=0}^{m} \varphi_j \quad \Rightarrow \quad (\bigvee_{j=0}^{m} \mathcal{W}\, \varphi_0$$

If, in addition, $\varphi_0 = r$,

$$\text{U1:} \quad \bigvee_{j=0}^{m} \varphi_j \quad \rightarrow \quad q, \qquad \textit{and} \qquad \text{U2:} \quad p \quad \rightarrow \quad \bigvee_{j=0}^{m} \varphi_j,$$

then we can conclude:

$$P \models p \quad \Rightarrow \quad q\, \mathcal{W}\, r.$$

Justification We observe first that the verification conditions associated with a waiting diagram imply premise U3 of rule WAIT, when we take φ to be $\varphi_1 \vee \cdots \vee \varphi_m$ and r to be φ_0.

If we further take $p = \varphi_0 \vee \cdots \vee \varphi_m$ and $q = \varphi_1 \vee \cdots \vee \varphi_m$, we find that premises U1 and U2 hold trivially. This yields the first part of the claim.

The second part of the claim considers different p and q but explicitly requires that premises U1 and U2 are P-valid. The conclusion follows by rule WAIT. ▟

Example 8. In Fig. 15, we present a waiting diagram that establishes the property

$$at\_\ell_1 \wedge at\_m_1 \wedge x = 0 \wedge T_a = 0 \quad \Rightarrow \quad (T_a \leq 15)\, \mathcal{W}\, at\_\ell_3$$

for program CYCLIC. ▟

5 Verifying Response Properties of Clocked Transition Systems

In this section, we present methods for verifying response properties of clocked transitions systems.

5.1 The Clock-Bounded Chain Rule

The basic rule for proving response properties of clocked transition systems is the *clock-bounded chain rule* (rule CB-CHAIN) presented in Fig. 16. The rule uses auxiliary assertions $\varphi_1, \ldots, \varphi_m$ and refers to assertion q also as φ_0. With each assertion φ_i we associate one of the clocks $c_i \in C$, to which we refer as the *helpful clock*, and an upper bound b_i. The intention is that while remaining in states satisfying φ_i, the clock c_i is bounded by b_i. Since time in a computation grows beyond any bound, this will imply that we cannot continually stay at φ_i-state for too long.

Premise C1 requires that every p-position satisfies one of $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$.

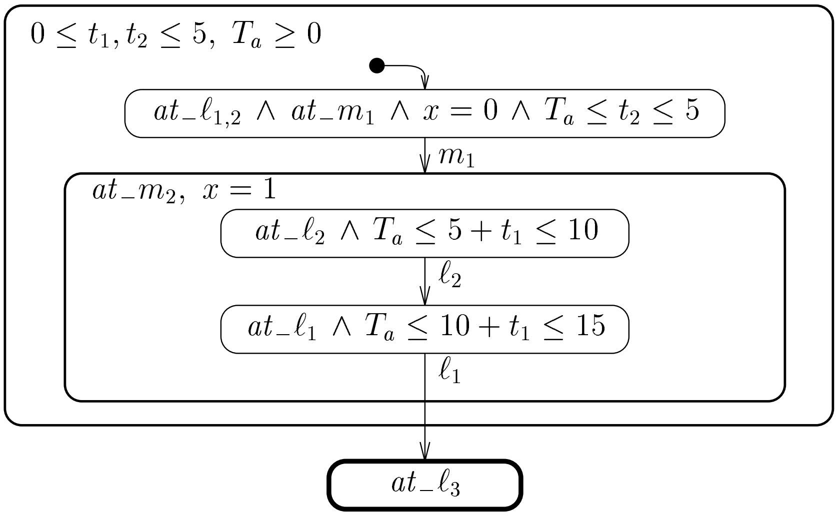

Fig. 15. A waiting diagram, establishing the formula
$at\_\ell_1 \wedge at\_m_1 \wedge x = 0 \wedge T_a = 0 \quad \Rightarrow \quad (T_a \leq 15)\, \mathcal{W}\ at\_\ell_3$

Premise C2 requires that every τ-successor (for any $\tau \in \mathcal{T}_T$) of a φ_i-state s is a φ_j-state for some $j \leq i$. In the case that the successor state satisfies φ_i, it is required that the transition does not decrease the value of c_i.

Premise C3 requires that assertion φ_i implies that c_i is bounded by the constant b_i.

The following claim states the soundness of the rule:

Claim 4 *Rule* CB-CHAIN *is sound for proving that a response formula is Φ-valid.*

Justification: Assume that the premises of the rule are Φ-valid, and let σ be a computation of Φ. We will show that σ satisfies the rule's consequence

$$p \;\Rightarrow\; \diamondsuit\, q,$$

i.e., every p position in σ is followed by a q-position.

Assume that p holds at position k and no later position $i \geq k$ satisfies q. By C1 some φ_j must hold at position k. Let j_k denote the minimal index such that φ_{j_k} holds at k. Obviously $j_k > 0$ by our assumption that q never occurs beyond position k.

By C2, state s_{k+1} must satisfy φ_j for some j, $0 < j \leq j_k$. Let j_{k+1} denote the minimal such index. Continuing in this manner we obtain that every position i beyond k satisfies some φ_{j_i}, where $j_i > 0$ and

$$j_k \geq j_{k+1} \geq j_{k+2} \geq \cdots\ .$$

Since we cannot have an infinite non-increasing sequence of natural numbers

For assertions p, q, and $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$,
clocks $c_1, \ldots, c_m \in C$, and
real constants $b_1, \ldots, b_m \in \mathsf{R}$,

C1. $p \;\rightarrow\; \bigvee_{j=0}^{m} \varphi_j$

The following two premises hold for $i = 1, \ldots, m$

C2. $\rho_\tau \;\wedge\; \varphi_i \;\rightarrow\; (\varpi' \;\wedge\; c_i' \geq c_i) \;\vee\; \bigvee_{j<i} \varphi_j'$

for every $\tau \in \mathcal{T}_T$

C3. $\varphi_i \;\rightarrow\; c_i \leq b_i$

$p \;\Rightarrow\; \diamondsuit\, q$

Fig. 16. Rule CB-CHAIN (clock-bounded chain rule for response).

which decreases infinitely many times, there must exist a position $r \geq k$ such that

$$j_r = j_{r+1} = j_{r+2} = \cdots \ .$$

Denote the value of this eventually-stable assertion index by $u = j_r$.

Consider the value of the clock c_u at states s_i, $i \geq r$. By C2, the value of c_u never decreases. Also, whenever a *tick* transition with increment Δ is taken, c_u increases (as do all clocks) by Δ. It follows that the master clock T cannot increase by more than $b_u - s_r[c_u]$ from its value at state s_r. This contradicts the fact that σ is a computation in which the master clock increases beyond all bounds.

We conclude that our assumption of the existence of a p-position not followed by any q-position is false. Consequently, if the premises of the rule hold then every p-position must be followed by a q-position, establishing the consequence of the rule. ▪

Example 9. We illustrate the use of rule CB-CHAIN to prove the termination of program ANY-Y$_{[3,5]}$, which can be stated by the response formula

$$at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x = t_1 = t_2 = 0 \quad\Rightarrow\quad \diamondsuit(at\_\ell_2 \;\wedge\; at\_m_2).$$

To apply rule CB-CHAIN, we identify p as $at\_\ell_0 \wedge at\_m_0 \wedge x = t_1 = t_2 = 0$ and q as $at\_\ell_2 \wedge at\_m_2$. The auxiliary assertions, helpful clocks, and bounds are presented in the following table:

φ_0: $at\_\ell_2 \wedge at\_m_2$	–	–
φ_1: $at\_\ell_0 \wedge at\_m_1 \wedge x = 1 \wedge t_1 \le 5$	c_1: t_1	b_1: 5
φ_2: $at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \wedge t_1 \le 5$	c_2: t_1	b_1: 5
φ_3: $at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0 \wedge t_1 \le 5 \wedge t_2 \le 5$	c_3: t_2	b_1: 5

We check the premises of rule CB-CHAIN for this selection of auxiliary assertions, helpful clocks, and bounds.

- Premise C1 assumes the form

$$\underbrace{at\_\ell_0 \wedge at\_m_0 \wedge x = y = t_1 = t_2 = T = 0}_{p} \;\rightarrow\; \underbrace{\cdots \vee\; at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0 \wedge t_1 \le 5 \wedge t_2 \le 5}_{\cdots \vee \varphi_3},$$

  which is obviously state valid.
- Premise C2 has to be checked for each $i = 1, \ldots, m$ and each $\tau \in \mathcal{T}_T$. We present only few representative cases.
  Premise C2 for φ_3 and transition m_0 assumes the form

$$\underbrace{\pi' = \pi - \{m_0\} \cup \{m_1\} \wedge x' = 1 \wedge t_1' = t_1 \wedge \cdots}_{\rho_{m_0}} \wedge \underbrace{at\_\ell_{0,1} \wedge t_1 \le 5 \wedge \cdots}_{\varphi_3}$$
$$\rightarrow \left(\begin{array}{ll} & \cdots \\ \vee & \underbrace{at'\_\ell_0 \wedge at'\_m_1 \wedge x' = 1 \wedge t_1' \le 5}_{\varphi_1'} \\ \vee & \underbrace{at'\_\ell_1 \wedge at'\_m_1 \wedge x' = 1 \wedge t_1' \le 5}_{\varphi_2'} \end{array}\right),$$

  which is obviously state valid.
  Premise C2 for φ_2 and the *tick* transition assume the form

$$\underbrace{\Delta > 0 \wedge \pi' = \pi \wedge x' = x \wedge t_1' = t_1 + \Delta \wedge (at\_\ell_{0,1} \rightarrow t_1 + \Delta \le 5) \wedge \cdots}_{\rho_{tick}}$$
$$\wedge \underbrace{at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \wedge t_1 \le 5}_{\varphi_2} \rightarrow$$
$$\underbrace{at'\_\ell_1 \wedge at'\_m_1 \wedge x' = 1 \wedge t_1' \le 5}_{\varphi_2'},$$

  which is obviously state valid.
- Premise C3 is trivially valid, since each φ_i, $i = 1, \ldots, 3$ includes $c_i \le b_i$ as a conjunct.

This analysis indicates that all premises of rule CB-CHAIN are state valid. It follows that the reponse formula

$$at\_\ell_0 \wedge at\_m_0 \wedge x = t_1 = t_2 = 0 \quad\Rightarrow\quad \diamondsuit(at\_\ell_2 \wedge at\_m_2)$$

is valid over program ANY-Y$_{[3,5]}$. ▟

5.2 Clock-Bounded Chain Diagrams

The main ingredients of a proof by rule CB-CHAIN can be conveniently and effectively presented by a special type of verification diagrams that summarizes the auxiliary assertions with their helpful clocks and bounds, and display the possible transitions between the assertions.

We define a *clock bounded chain diagram* (*chain diagram* for short) to be a directed labeled graph constructed as follows:

- *Nodes* in the graph are labeled by assertions. Each node is labeled by a pair of assertions: ϕ_i and β_i, for $i = 0, \ldots, m$. The assertion β_i has the form $c_i \leq b_i$, where $c_i \in C$ is a clock and b_i is a real constant. We refer to the conjunction $\phi_i \wedge \beta_i$ as φ_i, and say that the node is labeled by the (combined) assertion φ_i. We often refer to nodes by the assertion φ_i labeling them.
  The node labeled by φ_0 is called the *terminal node* and is graphically identified by being drawn with bold-face boundaries.
- *Edges* in the graph represent transitions between assertions. Each edge connects one assertion to another and is labeled by one or more transitions. An edge can connect node φ_i to node φ_j only if $i \geq j$. This imposes the restriction that the graph of a chain diagram is weakly acyclic, i.e., the only cycles in the graph consist of a node connected to itself.

Verification Conditions Implied by a Chain Diagram

Consider a nonterminal node labeled by assertion φ: $\phi \wedge \beta$ where the clock-bound assertion is β: $c \leq b$. Let $\tau \in \mathcal{T}_T$ be a transition and let $\varphi_1, \ldots, \varphi_k$, $k \geq 0$, be the successors of φ by edges labeled with τ (possibly including φ itself). With each such node and transition, we associate the following verification condition:

$$\rho_\tau \;\wedge\; \varphi \quad\rightarrow\quad (\varphi' \wedge c' \geq c) \;\vee\; \varphi_1' \;\vee\; \cdots \;\vee\; \varphi_k'.$$

In particular, if $k = 0$ (i.e., φ has no τ-successors), the associated verification condition is

$$\rho_\tau \;\wedge\; \varphi \quad\rightarrow\quad \varphi' \wedge c' \geq c.$$

Valid Chain Diagrams

A chain diagram is said to be *valid over* CTS Φ (Φ*-valid* for short) if all the verification conditions associated with the diagram are Φ-state valid.

The consequences of having a valid chain diagram are stated in the following claim:

Claim 5 *If D is a Φ-valid chain diagram with nodes $\varphi_0, \ldots, \varphi_m$, then*

$$\Phi \models \bigvee_{j=0}^{m} \varphi_j \quad\Rightarrow\quad \Diamond\,\varphi_0$$

If, in addition, $\varphi_0 = q$ *and*

$$\text{C1:} \quad p \quad \rightarrow \quad \bigvee_{j=0}^{m} \varphi_j,$$

then we can conclude:

$$\Phi \models p \quad \Rightarrow \quad \diamondsuit\, q.$$

The claim follows from the observation that the verification conditions associated with a chain diagram precisely correspond to premise C2 of rule CB-CHAIN. Premise C1 is trivially satisfied for the first part of the claim, where we take p to be $\bigvee_{j=0}^{m} \varphi_j$. It is explicitly provided in the second part of the claim. Premise C3 is trivially satisfied by having β_i: $c_i \leq b_i$ as an explicit conjunct of $varp_i = \phi_i \wedge \beta_i$.

Example 10. In Fig. 17, we present a chain diagram for proving that

$$at\_\ell_0 \wedge at\_m_0 \wedge x = t_1 = t_2 = 0 \quad \Rightarrow \quad \diamondsuit(at\_\ell_2 \wedge at\_m_2)$$

is valid over program ANY-Y$_{[3,5]}$.

Note the use of encapsulation in labeling the compound node by the common clock bound B: $t_1 \leq 5$, which is factored out of nodes φ_1 and φ_2. We also remove the index from the β assertion labeling a node and write β instead of β_i. ▟

5.3 Winning a Race

We introduce an additional graph-structuring convention which leads to a more economic and comprehensible verification diagrams. Similar to the previously introduced encapsulation conventions, this one is also inspired by the Statechart language [Har87].

A *conjunctive compound node* is a compound node which contains two sets of encapsulated nodes: $\{\phi_1, \ldots, \phi_m\}$ and $\{\psi_1, \ldots, \psi_n\}$. The two sets are separated by a dashed line. Edges may connect nodes within each of the sets, and external nodes to nodes in each of the sets. No edge may connect a ϕ-node to a ψ-node. We also allow a multi-source edge such as the edge connecting nodes ϕ_2 and ψ_2 to the external node χ.

In Fig. 18, we present a graph with a conjunctive compound node.

Any diagram containing conjunctive nodes can be expanded into an equivalent flat diagram, to which we refer as the *expanded diagram*, as follows:

- For each $i \in \{1, \ldots, m\}$ and $j \in \{\psi_1, \ldots, n\}$, the expanded diagram contains a node, labeled by the conjunction $\phi_i \wedge \psi_j$.
- For each τ-labeled edge connecting ϕ_a to ϕ_b, there are τ-labeled edges connecting expanded node $\phi_a \wedge \psi_j$ to $\phi_b \wedge \psi_j$ for all $j \in \{1, \ldots, n\}$.
- For each τ-labeled edge connecting ψ_c to ψ_d, there are τ-labeled edges connecting expanded node $\phi_i \wedge \psi_c$ to $\phi_i \wedge \psi_d$ for all $i \in \{1, \ldots, m\}$.
- For each τ-labeled edge connecting ϕ_a to external node χ, there are τ-labeled edges connecting expanded node $\phi_a \wedge \psi_j$ to χ for all $j \in \{1, \ldots, n\}$.

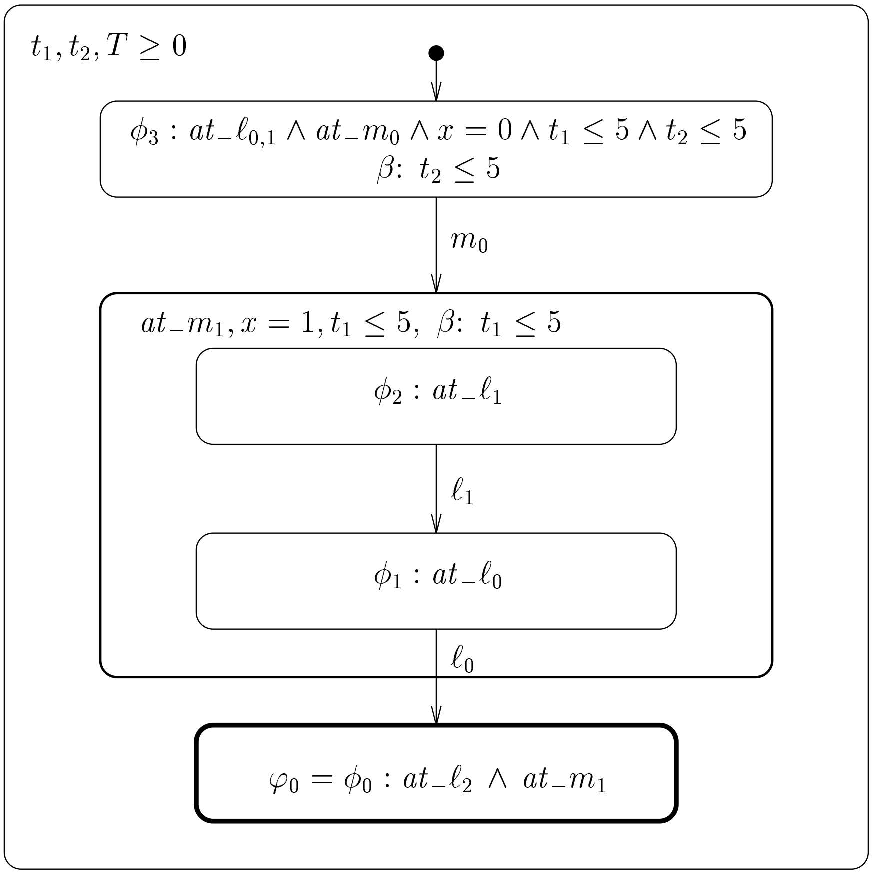

Fig. 17. Verifying termination of program ANY-Y$_{[3,5]}$.

- For each τ-labeled edge connecting ψ_c to external node χ, there are τ-labeled edges connecting expanded node $\phi_i \wedge \psi_c$ to χ for all $i \in \{1, \ldots, m\}$.
- For each τ-labeled multi-source edge connecting nodes ϕ_a and ψ_c to external node χ, there exists a τ-labeled edge connecting expanded node $\phi_a \wedge \psi_c$ to node χ.

In Fig. 19, we present the flat diagram equivalent to the diagram of Fig. 18.

Analyzing Races between Processes

Conjunctive nodes are particularly helpful for proving that one process always wins in a race against a competing process. Consider the trivial program RACE presented in Fig. 20.

As in the case of program MUTEX$_{[L,U]}$, we assign to all statements of program RACE time bounds $[L, U]$, stipulating that $2L > U$. It is clear that when this

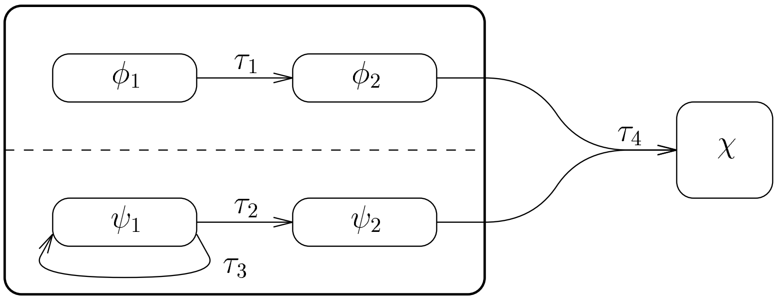

Fig. 18. A conjunctive compound node.

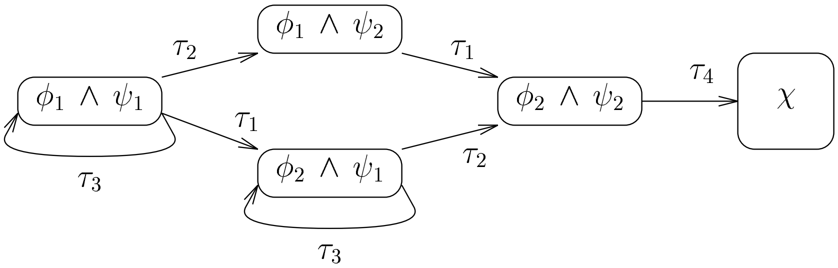

Fig. 19. An expanded equivalent to the conjunctive diagram.

program is run, process P_1 will terminate before P_2 does. This is because P_1 mst terminate within $2U$ time units, while P_2 must take at least $4L > 2U$ time units to terminate. How do we formally prove this property which can be stated by the response formula

$$at\_\ell_0 \wedge at\_m_0 \wedge t_1 = t_2 = T = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_{0..3}) \ ?$$

In Fig. 21, we present a chain diagram which proves this property.

$$\begin{bmatrix} \ell_0 : \textbf{skip} \\ \ell_1 : \textbf{skip} \\ \ell_2 : \end{bmatrix} \quad \Big\| \quad \begin{bmatrix} m_0 : \textbf{skip} \\ m_1 : \textbf{skip} \\ m_2 : \textbf{skip} \\ m_3 : \textbf{skip} \\ m_4 : \end{bmatrix}$$

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

Fig. 20. Program RACE.

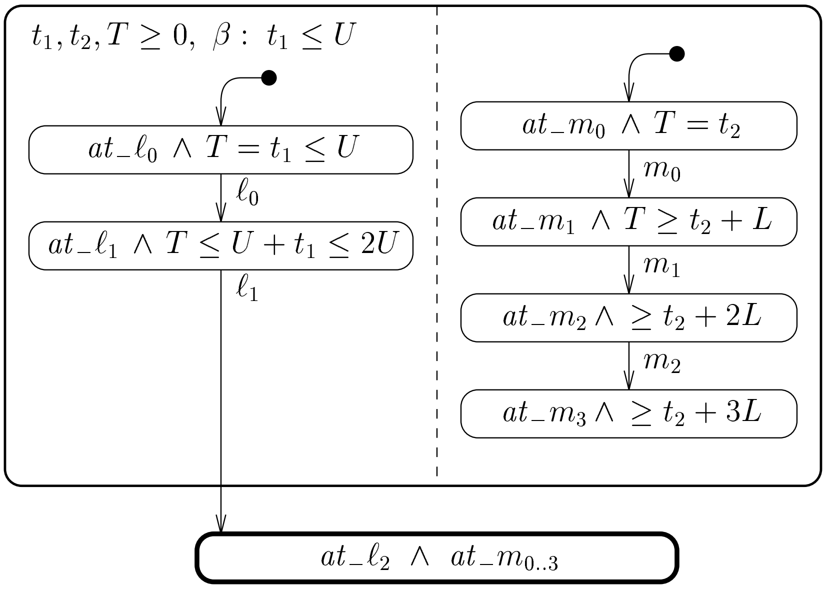

Fig. 21. Chain diagram proving that P_1 wins the race.

A central argument in the validation of this diagram is that transition m_2 is disabled on all nodes within the conjunctive compound node. Transition m_2 can be enabled only on an at_m_2-state and only when $t_2 \geq L$. Combining the assertion attached to the m_2 node with $t_2 \geq L$, we obtain $T \geq 2L > 4U$. However, all assertions on the left-hand side of the conjunctive node imply $T \leq 4U$. This shows that m_2 is disabled on all states covered by the conjunctive node, and the only exit is via ℓ_1.

This establishes the property

$$at\_\ell_0 \wedge at\_m_0 \wedge t_1 = t_2 = T = 0 \quad \Rightarrow \quad \diamondsuit(at\_\ell_2 \wedge at\_m_{0..3}).$$

5.4 Proving Accessibility for Program $\text{MUTEX}_{[L,U]}$

As a more ambitious example, we prove for program $\text{MUTEX}_{[L,U]}$ the property of accessibility which can be stated (for process P_1) by the response formula

$$at\_\ell_2 \quad \Rightarrow \quad \diamondsuit\, at\_\ell_9.$$

A similar formula states accessibility for process P_2.

From the invariance diagram of Fig. 10, we can infer the following four invariants, which we will use in the response proof:

$$\begin{array}{ll}
\chi_0: & \Box(x \in \{0,1,2\}) \\
\chi_1: & \Box(at_\ell_{0..2,4,5} \quad \rightarrow \quad x \neq 1) \\
\chi_2: & \Box(at_m_{0..2,4,5} \quad \rightarrow \quad x \neq 2) \\
\chi_3: & at_\ell_4 \quad \Rightarrow \quad x = 0 \ \vee\ (at_m_{6..10} \wedge x = 2)
\end{array}$$

The accessibility formula is proved in several steps, verifying separately the following response formulas:

$$\begin{array}{ll}
\psi_1: & at_\ell_2 \quad \Rightarrow \quad \Diamond\, at_\ell_4 \\
\psi_2: & at_\ell_4 \wedge x = 0 \quad \Rightarrow \\
\multicolumn{2}{l}{\Diamond\big(at_\ell_8 \ \vee\ (at_\ell_{4,6,7} \wedge at_m_6 \wedge x = 2 \wedge t_2 = 0)\big)} \\
\psi_3: & at_\ell_{4,6,7} \wedge at_m_6 \wedge x = 2 \wedge t_2 = 0 \quad \Rightarrow \\
 & \qquad\qquad \Diamond(at_\ell_4 \wedge at_m_0 \wedge x = 0 \wedge t_2 = 0) \\
\psi_4: & at_\ell_4 \wedge at_m_{6..10} \wedge x = 2 \quad \Rightarrow \\
 & \qquad\qquad \Diamond(at_\ell_4 \wedge at_m_0 \wedge x = 0 \wedge t_2 = 0) \\
\psi_5: & at_\ell_4 \wedge at_m_0 \wedge x = 0 \wedge t_2 = 0 \quad \Rightarrow \\
 & \qquad\qquad\qquad \Diamond(at_\ell_6 \wedge at_m_{0..4} \wedge x = 1) \\
\psi_6: & at_\ell_6 \wedge at_m_{0..4} \wedge x = 1 \quad \Rightarrow \quad \Diamond\, at_\ell_8
\end{array}$$

It is not difficult to see that response formulas ψ_1–ψ_6 lead to the accessibility property.

We proceed to prove each of the response formulas.

Proving $\boldsymbol{\psi_1}$: $\quad at\_\boldsymbol{\ell}_2 \quad \Rightarrow \quad \Diamond\, at\_\boldsymbol{\ell}_4$

Formula ψ_1 states that, starting at ℓ_2, process P_2 is guaranteed to reach ℓ_4. Statement ℓ_2 is unconditional and is guaranteed to terminate within U time units. By χ_1, when we enter ℓ_3 from ℓ_2, x is different from 1, and will remain so as long as we stay at ℓ_3 (ℓ_5 is the only statement that can set x to 1). Consequently, within U time units, P_1 will proceed to ℓ_4.

The formal proof of ψ_1 is provided by the chain diagram of Fig. 22

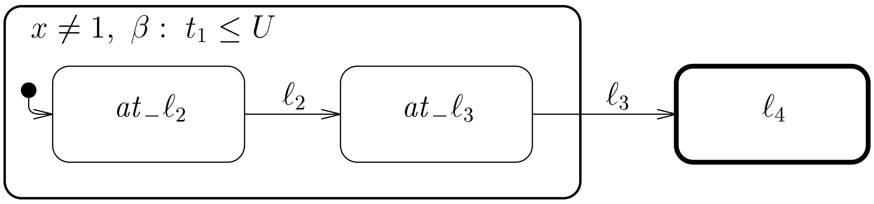

Fig. 22. Chain diagram for ψ_1: $at\_\ell_2 \ \Rightarrow\ \Diamond\, at\_\ell_4$.

Proving ψ_2

Response formula ψ_2 is given by

$$at\_\ell_4 \wedge x = 0 \quad \Rightarrow \quad \Diamond\big(at\_\ell_8 \ \vee \ (at\_\ell_{4,6,7} \wedge at\_m_6 \wedge x = 2 \wedge t_2 = 0)\big).$$

It states that, starting at location ℓ_4 with $x = 0$, process P_1 will either reach the critical section (at ℓ_8), or be overtaken by process P_2 just entering location m_6 (and hence $t_2 = 0$), while setting x to 2. In the latter case, P_1 will be overtaken at one of the locations ℓ_4, ℓ_6, or ℓ_7.

The formal proof of ψ_2 is presented by the chain diagram of Fig. 23.

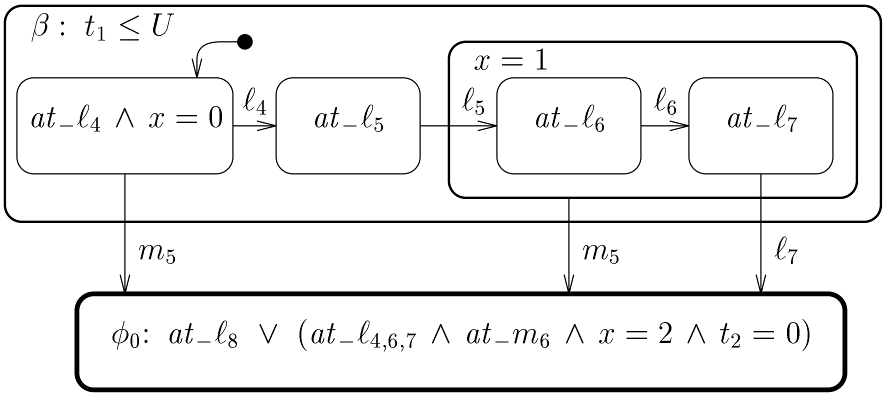

Fig. 23. Chain diagram for the formula
ψ_2: $at\_\ell_4 \wedge x = 0 \quad \Rightarrow \quad \Diamond\big(at\_\ell_8 \ \vee \ (at\_\ell_{4,6,7} \wedge at\_m_6 \wedge x = 2 \wedge t_2 = 0)\big)$.

The diagram follows the progress of process P_1 from ℓ_4 with $x = 0$. While P_1 is at ℓ_4, P_2 may execute statement m_5 and set x to 2, which reaches the goal ϕ_0. If this does not happen, P_1 proceeds to ℓ_5 and then to ℓ_6, setting x to 1. From this point on, either P_2 will perform m_5, leading again to ϕ_0, or P_2 will perform ℓ_7 moving to ℓ_8, which is also a goal state.

Proving ψ_3

Response formula ψ_3 is given by

$$at\_\ell_{4,6,7} \wedge at\_m_6 \wedge x = 2 \wedge t_2 = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_4 \wedge at\_m_0 \wedge x = 0 \wedge t_2 = 0).$$

The formula states that, once P_1 has been overtaken by P_2, it will return to location ℓ_4 *before* P_2 exits its critical section and returns to m_0, setting x back to 0. The fact that, being denied entry to ℓ_8, process P_1 must eventually return to ℓ_3 and proceed to ℓ_4 is obvious. Less obvious is the fact that when P_2 performs

m_{10} on its exit from the critical section, P_1 is already at $\ell 4$. This result from the number of statements that each process has to execute until they reach ℓ_4 and m_0, respectively, and from the assumption $U < 2L$ which guarantees that P_1 completes the execution of a single statement before P_2 can complete the execution of two statements.

The worst case for P_1 is if it is overtaken at ℓ_6. To reach ℓ_4 it must execute 3 statements: ℓ_6, ℓ_7, and ℓ_3. It will take P_1 at most $3U$ to do so. To reach location m_0 from its initial location at m_6, P_2 has to execute at least 6 statements: m_6, m_7, m_8, m_3, m_9, and m_{10}. It will take P_2 at least $6L$ to reach m_0. Since $3U < 6L$, P_1 will get to ℓ_3 first.

The precise analysis of this race between P_1 and P_2 is presented by the chain diagram of Fig. 24

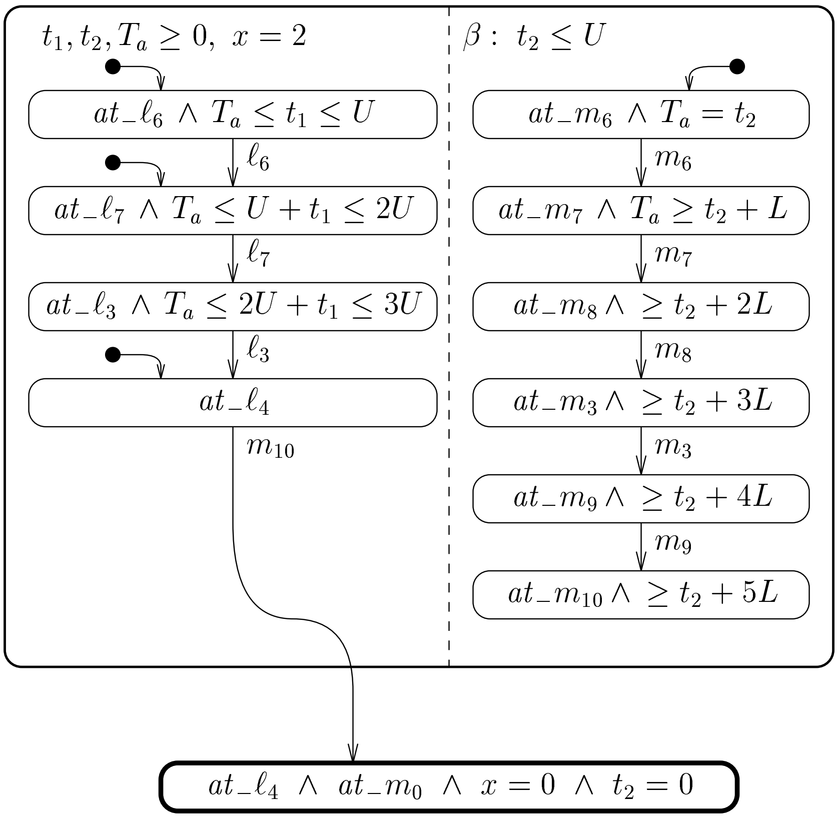

Fig. 24. Chain diagram for the formula

$$\psi_3: \quad at\_\ell_{4,6,7} \land at\_m_6 \land x = 2 \land t_2 = 0 \land T_a = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_4 \land at\_m_0 \land x = 0 \land t_2 = 0).$$

Note that m_{10} is enabled only at states in which P_1 is at ℓ_4. This is becaus at all other P_1-locations, $T_a \leq 3U < 6L$. For m_10 to be enabled, T_a must be at least $6L$.

Proving ψ_4

Response formula ψ_4 is given by

$$at\_\ell_4 \ \wedge \ at\_m_{6..10} \ \wedge \ x = 2 \quad \Rightarrow \quad \diamond(at\_\ell_4 \ \wedge \ at\_m_0 \ \wedge \ x = 0 \ \wedge \ t_2 = 0).$$

The formula states that, starting with P_1 at ℓ_4, $x = 2$, and P_2 somewhere within $\{m_6, \ldots, m_{10}\}$, we are guaranteed to reach a state in which P_1 is still at ℓ_4 but P_2 has just moved to m_0 (hence $t_2 = x = 0$). This property relies on the progress of P_2 through $\{m_6, \ldots, m_{10}\}$ to m_0 while $x = 2$ and P_1 cannot change the value of x, beeing stuck at ℓ_4. For a formal proof of this property, we can use again the diagram of Fig. 24, where the initial nodes are within the compound node labeled by at_ℓ_4.

Proving ψ_5

Response formula ψ_5 is given by

$$at\_\ell_4 \ \wedge \ at\_m_0 \ \wedge \ x = 0 \ \wedge \ t_2 = 0 \quad \Rightarrow \quad \diamond(at\_\ell_6 \ \wedge \ at\_m_{0..4} \ \wedge \ x = 1).$$

The formula considers another possible race between P_1 and P_2, starting with P_1 at ℓ_4 and P_2 just arriving to m_0. Formula ψ_5 states that P_1 will reach ℓ_6 (with $x = 1$ before P_2 reaches m_5. Note that from a state satisfying $at\_\ell_6 \ \wedge \ at\_m_{0..4} \ \wedge \ x = 1$ the entry of P_1 to its critical section is guaranteed, since P_2 cannot pass the test at m_4 and interefere with P_1's progress.

Formula ψ_5 is verified by the chain diagram of Fig. 25.

A simple informal argument explains why P_1 is sure to win this race. To move from ℓ_4 to ℓ_6, P_1 has to execute 2 statements: ℓ_4 and ℓ_5, which takes at most $2U$. In that time, P_2 cannot complete the execution of the 5 statements m_0–m_4 necessary to reach m_5, since $5L > 2U$.

Proving ψ_6

Response formula ψ_6 is given by

$$\psi_6: \quad at\_\ell_6 \ \wedge \ at\_m_{0..4} \ \wedge \ x = 1 \quad \Rightarrow \quad \diamond \ at\_\ell_8.$$

This formula states that, once P_1 reached ℓ_6 while P_2 is still confined within the range $\{m_0, \ldots, m_4\}$, entry of P_1 to ℓ_8 is guaranteed. The proof presented in the diagram of Fig. 26 simply traces the progress of P_1 from ℓ_6 to ℓ_8, while x keeps its value of 1.

This concludes the proof of accessibility for program $\text{MUTEX}_{[L,U]}$.

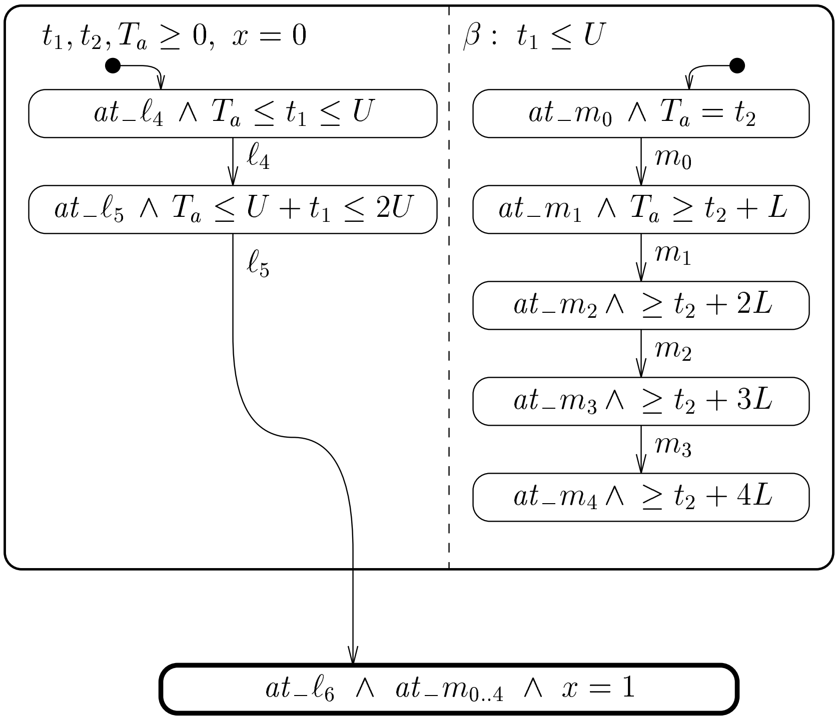

Fig. 25. Chain diagram for the formula
ψ_5: $\quad at\_\ell_4 \wedge at\_m_0 \wedge x = 0 \wedge t_2 = 0 \wedge T_a = 0 \quad \Rightarrow \quad \diamondsuit(at\_\ell_6 \wedge at\_m_{0..4} \wedge x = 1)$.

5.5 Clock-Bounded Well-Founded Rule

Rule CB-CHAIN is adequate for proving response properties in which a q state is achieved in a number of significant steps which is a priori bounded. For example, in verifying termination of program ANY-Y$_{[3,5]}$, there were 3 helpful step leading to termination. These are represented in the chain diagram of Fig. 17 by the edges entering nodes ϕ_2–ϕ_0.

In many cases, the number of helpful steps needed to reach the goal q cannot be bounded a priori. For these cases we need a stronger rule, based on well-founded ordering.

Well-founded Domains

We define a *well-founded domain* $(\mathcal{A}, \succ)$ to consist of a set $\mathcal{A}$ and a *well-founded order* relation $\succ$ on $\mathcal{A}$. A binary relation $\succ$ is called an *order* if it is

- transitive: $a \succ b$ and $b \succ c$ imply $a \succ c$, and
- irreflexive: $a \succ a$ for no $a \in \mathcal{A}$.

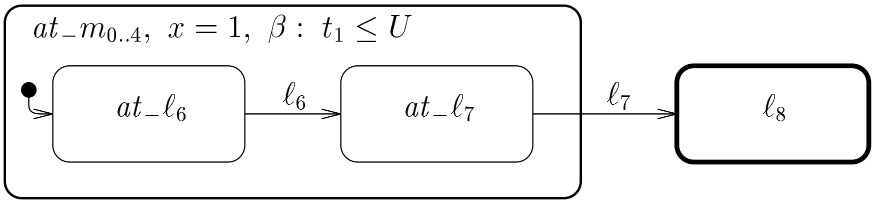

Fig. 26. Chain diagram for the formula
ψ_6: $\quad at\_\ell_6 \wedge at\_m_{0..4} \wedge x = 1 \quad \Rightarrow \quad \diamond\, at\_\ell_8$.

The relation $\succ$ is called *well-founded* if there does not exist an infinitely descending sequence $a_0, a_1, \ldots$ of elements of $\mathcal{A}$ such that

$$a_0 \ \succ \ a_1 \ \succ \ \cdots .$$

A typical example of a well-founded domain is $(\mathbb{N}, >)$, where $\mathbb{N}$ are the natural numbers (including 0) and $>$ is the greater-than relation. Clearly, $>$ is well-founded over the natural numbers, because there cannot exist an infinitely descending sequence of natural numbers

$$n_0 \ > \ n_1 \ > \ n_2 \ > \ \ldots .$$

For $\succ$, an arbitrary order relation on $\mathcal{A}$, we define its *reflexive extension* $\succeq$ to hold between a, $a' \in \mathcal{A}$ if either $a \succ a'$ or $a = a'$.

Lexicographic Tuples

Another frequently used well-founded domain is $(\mathbb{N}^k, \succ)$, where $\mathbb{N}^k$ is the set of k-tuples of natural numbers. The order $\succ$ is defined by

$$(n_1, \ldots, n_k) \ \succ \ (m_1, \ldots, m_k) \quad \text{iff} \quad \begin{array}{l} n_1 = m_1,\ \ldots,\ n_{i-1} = m_{i-1},\ n_i > m_i \\ \text{for some } i,\ 1 \leq i \leq k. \end{array}$$

For example, for $k = 3$

$$(7,\ 2,\ 1) \ \succ \ (7,\ 0,\ 45).$$

It is easy to show that the domain $(\mathbb{N}^k, \succ)$ is well-founded.

It is possible to make lexicographic comparisons between tuples of integers of different lengths. The convention is that the relation holding between $(a_1, \ldots, a_i)$ and $(b_1, \ldots, b_k)$ for $i < k$ is determined by lexicographically comparing $(a_1, \ldots, a_i, 0, \ldots, 0)$ to $(b_1, \ldots, b_i, b_{i+1}, \ldots, b_k)$. That is, we pad the shorter tuple by zeros on the right until it assumes the length of the longer tuple.

According to this definition, $(0,\ 2) \succ 0$, since $(0,\ 2) \succ (0,\ 0)$. In a similar way, $1 \succ (0,\ 2)$.

5.6 Rule CB-WELL

In Fig. 27, we present the *clock-bounded well-founded response rule* (rule CB-WELL) for proving response properties of fair transition systems. The rule uses auxiliary assertions $\varphi_1, \ldots, \varphi_m$ and refers to assertion q also as φ_0. With each assertion φ_i, $i > 0$, we associate one of the clocks $c_i \in C$, to which we refer as the *helpful clock*, and an upper bound B_i, which is a real-valued expression. Also required are a well-founded domain $(\mathcal{A}, \succ)$, and ranking functions δ_i: $\Sigma \mapsto \mathcal{A}$, $i = 1, \ldots, m$, mapping states of the system to elements of $\mathcal{A}$. The ranking functions measure progress of the computation towards the goal q.

For assertions p, q, and $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$,
clocks $c_1, \ldots, c_m \in C$,
real expressions $B_1, \ldots, B_m \in \mathsf{R}$,
a well-founded domain $(\mathcal{A}, \succ)$, and
ranking functions $\delta_1, \ldots, \delta_m$: $\Sigma \mapsto \mathcal{A}$,

W1. $p \;\rightarrow\; \bigvee_{j=0}^{m} \varphi_j$

The following two premises hold for $i = 1, \ldots, m$

W2. $\rho_\tau \;\wedge\; \varphi_i \;\rightarrow\; \bigvee_{j=0}^{m} (\varphi'_j \;\wedge\; \delta_i \succ \delta'_j) \;\vee$
$(\varphi'_i \;\wedge\; \delta'_i = \delta_i \;\wedge\; c_i \leq c'_i \;\wedge\; B'_i \leq B_i)$
for every $\tau \in \mathcal{T}_T$

W3. $\varphi_i \;\rightarrow\; c_i \leq B_i$

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

Fig. 27. Rule CB-WELL (clock-bounded well-founded rule for response).

Premise W1 requires that every p-position satisfies one of $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$.

Premise W2 requires that every τ-successor (for any $\tau \in \mathcal{T}_T$) of a φ_i-state s is a φ_j-state for some $j \leq i$, with a rank δ'_j not exceeding δ_i. In the case that the successor state satisfies φ_i, it is allowed that $\delta'_i = \delta_i$ but is required that the transition does not decrease the value of c_i or increase the value of B_i. In all other cases it is required that $\delta'_j \prec \delta_i$, i.e., that the rank strictly decreases.

Premise W3 requires that assertion φ_i implies that c_i is bounded by the constant B_i.

The following claim states the soundness of the rule:

Claim 6 *Rule* CB-WELL *is sound for proving that a response formula is Φ-valid.*

Justification: Assume that the premises of the rule are Φ-valid, and let σ be a computation of Φ. We will show that σ satisfies the rule's consequence

$$p \Rightarrow \Diamond q.$$

Assume that Φ holds at position k and no later position $i \geq k$ satisfies q. By W1 some φ_j must hold at position k. Let $u_k \in \mathcal{A}$ be the minimal rank of state s_k, i.e. the minimal value of $\delta_j(s_k)$ among all φ_j which hold at s_k. Let $u_k = \delta_{j_k}(s_k)$, which implies that φ_{j_k} holds at s_k.

By C2, state s_{k+1} must satisfy φ_j for some $j > 0$, implying that s_{k+1} has a define rank u_{k+1}. Premise C2 requires that $u_{k+1} \preceq uk$.

proceeding in this manner we obtain that every position i beyond k has a rank u_i, such that

$$u_k \succeq u_{k+1} \succeq u_{k+2} \succeq \cdots .$$

Since $\mathcal{A}$ is well-founded, there must exist a position $r \geq k$ such that

$$u_r = u_{r+1} = u_{r+2} = \cdots .$$

Denote the value of this eventually-stable rank by $u = u_r$, and let $j_r > 0$ be the index of the assertion such that $\delta_{j_r}(s_r) = u$.

Consider the value of the clock c_{j_r} at states s_i, $i \geq r$. Since the rank never decreases beyond r, the value of c_{j_1} never decreases. Also, whenever a *tick* transition with increment Δ is taken, c_{j_1} increases (as do all clocks) by Δ. It follows that the master clock T cannot increase by more than $B_{j_r}(s_r) - s_r[c_{j_r}]$ from its value at state s_r. This contradicts the fact that σ is a computation in which the master clock increases beyond all bounds.

We conclude that our assumption of the existence of a p-position not followed by any q-position is false. Consequently, if the premises of the rule hold then every p-position must be followed by a q-position, establishing the consequence of the rule. ▪

The following example illustrates an application of rule CB-WELL.

Example 11. Consider program UP-DOWN presented in Fig. 28.

This program can be viewed as a generalization of program ANY-Y in which, after terminating the while loop at ℓ_0, ℓ_1, process P_1 proceeds to perform a second while loop at ℓ_2, ℓ_3, decrementing y until it reaches 0.

Assume, we assign the uniform time bounds $[L, U]$ to all executable statements of program UP-DOWN, where our only information about L and U is given by

$$0 \leq L < U < \infty.$$

We use rule CB-WELL to verify that program UP-DOWN terminates. This property can be expressed by the response formula

$$\underbrace{at\_\ell_0 \land at\_m_0 \land x = y = t_1 = t_2 = T = 0}_{p} \quad \Rightarrow \quad \Diamond(\underbrace{at\_\ell_4 \land at\_m_1}_{q}).$$

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

$$\left[\begin{array}{l} \ell_0 : \mathbf{while}\ x = 0\ \mathbf{do} \\ \quad \ell_1 :\ y := y + 1 \\ \ell_2 : \mathbf{while}\ y > 0\ \mathbf{do} \\ \quad \ell_3 :\ y := y - 1 \\ \ell_4 : \end{array}\right] \quad \Big\| \quad \left[\begin{array}{l} m_0 : x := 1 \\ m_1 : \end{array}\right]$$

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

Fig. 28. Program UP-DOWN.

As the well-founded domain, we take $(\mathbb{N}^2, \succ)$, i.e., the domain of lexicographic pairs. As time bounds, we use B_i: U for all $i = 1, \ldots, 5$. The auxiliary assertions, helpful clocks, and ranking functions are given by the following table:

φ_0: $at\_\ell_4 \wedge at\_m_1$		δ_0: 0
φ_1: $at\_\ell_3 \wedge at\_m_1 \wedge x = 1 \wedge y > 0 \wedge t_1 \leq U$	c_1: t_1	δ_1: $(1, 2y)$
φ_2: $at\_\ell_2 \wedge at\_m_1 \wedge x = 1 \wedge y \geq 0 \wedge t_1 \leq U$	c_2: t_1	δ_2: $(1, 2y + 1)$
φ_3: $at\_\ell_0 \wedge at\_m_1 \wedge x = 1 \wedge y \geq 0 \wedge t_1 \leq U$	c_3: t_1	δ_3: 2
φ_4: $at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \wedge y \geq 0 \wedge t_1 \leq U$	c_4: t_1	δ_4: 3
φ_5: $at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0 \wedge y \geq 0 \wedge t_1 \leq U \wedge t_2 \leq U$	c_5: t_1	δ_5: 4

We consider two instances of premise W2: transition ℓ_2 taken from φ_2 and transition ℓ_3 taken from φ_1.

For the case of φ_2, premise W2 assumes the form

$$\underbrace{\begin{pmatrix} y > 0 \wedge move(\ell_2, \ell_3) \\ \vee \\ y \leq 0 \wedge move(\ell_2, \ell_4) \end{pmatrix} \wedge pres(V - \{\pi\})}_{\rho_{\ell_2}} \wedge \underbrace{at\_m_1 \wedge x = 1 \wedge y \geq 0 \wedge t_1 \leq U}_{\varphi_2}$$

$$\to$$

$$\cdots \vee \begin{pmatrix} \underbrace{at'\_\ell_3 \wedge at'\_m_1 \wedge x' = 1 \wedge y' > 0 \wedge t'_1 \leq U}_{\varphi'_1} \wedge \underbrace{(1, 2y + 1) \succ (1, 2y)}_{\delta_2 \succ \delta'_1} \\ \vee \\ \underbrace{at'\_\ell_4 \wedge at'\_m_1}_{\varphi'_0} \wedge \underbrace{(1, 2y + 1) \succ 0}_{\delta_2 \succ \delta'_0} \end{pmatrix}$$

The implication uses the abbreviation

$$move \ell_i, \ell_j\text{:} \quad \ell_i \in \pi \ \wedge\ \pi' = (\pi - \{\ell_i\}) \cup \{\ell_j\}.$$

Note that $move(\ell_i, \ell_j)$ implies at'_ℓ_j and $at'\_m_1 = at\_m_1$. Since ρ_{ℓ_2} implies $y' = y$, the implication is obviously valid. In particular, due to

$$(1, 2y + 1) \succ (1, 2y) \qquad \text{and} \qquad (1, 2y + 1) \succ 0.$$

Premise R2 for φ_1 and transition ℓ_3 assumes the form

$$\underbrace{move(\ell_3,\ell_2)\ \wedge\ y' = y-1\ \wedge\ pres(x,t_1,t_2,T)}_{\rho_{\ell_3}}\ \wedge$$

$$\underbrace{at\_\ell_3\ \wedge\ at\_m_1\ \wedge\ x=1\ \wedge\ y>0\ \wedge\ t_1 \leq U}_{\varphi_1}\ \rightarrow$$

$$\cdots \vee \left(\underbrace{at'\_\ell_2\ \wedge\ at'\_m_1\ \wedge\ x'=1\ \wedge\ y' \geq 0\ \wedge\ t_1' \leq U}_{\varphi_2'}\ \wedge\ \underbrace{(1,2y) \succ (1,2y'+1)}_{\delta_1 \succ \delta_2'}\right)$$

Since ρ_{ℓ_3} implies $y' = y-1$, it is easy to verify that $y>0$ implies $y' \geq 0$ and

$$(1,2y'+1) = (1,2(y-1)+1) = (1,2y-1) \prec (1,2y).$$

This establishes that the reponse property

$$at\_\ell_0\ \wedge\ at\_m_0\ \wedge\ x=y=t_1=t_2=T=0 \quad\Rightarrow\quad \diamondsuit(at\_\ell_4\ \wedge\ at\_m_1).$$

is valid over program UP-DOWN. ▟

5.7 Ranked Diagrams

The main ingredients of a proof by rule CB-WELL can be conveniently and effectively presented by a special type of verification diagrams that summarizes the auxiliary assertions, their helpful clocks and bounds, and their ranking functions, and display the possible transitions between the assertions.

We define a *rabked diagram* to be a directed labeled graph constructed as follows:

- *Nodes* in the graph are labeled by assertions. Each node is labeled by a pair of assertions: ϕ_i and β_i, for $i = 0,\ldots,m$, and a ranking function δ_i. The assertion β_i has the form $c_i \leq B_i$, where $c_i \in C$ is a clock and B_i is a real-valued expression. We refer to the conjunction $\phi_i \wedge \beta_i$ as φ_i, and say that the node is labeled by the (combined) assertion φ_i. We often refer to nodes by the assertion φ_i labeling them.
  The node labeled by φ_0 is called the *terminal node* and is graphically identified by being drawn with bold-face boundaries.
- *Edges* in the graph represent transitions between assertions. Each edge connects one assertion to another and is labeled by one or more transitions.

Verification Conditions Implied by a Ranked Diagram

Consider a nonterminal node labeled by assertion φ: $\phi \wedge \beta$ where the clock-bound assertion is β: $c \leq B$ and the ranking function is δ. Let $\tau \in \mathcal{T}_T$ be a transition and let $\varphi_1,\ldots,\varphi_k$, $k \geq 0$, be the successors of φ by edges labeled with

τ (possibly including φ itself). With each such node and transition, we associate the following verification condition:

$$\begin{aligned}\rho_\tau \ \wedge \ \varphi \quad \rightarrow \quad & (\varphi' \wedge \delta' = \delta \wedge c \leq c' \leq B' \leq B) \ \vee \ (\varphi' \wedge \delta \succ \delta') \ \vee \\ & (\varphi'_1 \ \wedge \delta \succ \delta'_1) \ \vee \ \cdots \ \vee \ (\varphi'_k \ \wedge \delta \succ \delta'_k).\end{aligned}$$

In particular, if $k = 0$ (i.e., φ has no τ-successors), the associated verification condition is

$$\rho_\tau \ \wedge \ \varphi \quad \rightarrow \quad (\varphi' \wedge \delta' = \delta \wedge c \leq c' \leq B' \leq B) \ \vee \ (\varphi' \wedge \delta \succ \delta').$$

Valid Ranked Diagrams

A Ranked diagram is said to be *valid over* CTS Φ (*Φ-valid* for short) if all the verification conditions associated with the diagram are Φ-state valid.

The consequences of having a valid chain diagram are stated in the following claim:

Claim 7 *If D is a Φ-valid ranked diagram with nodes $\varphi_0, \ldots, \varphi_m$, then*

$$\Phi \models \bigvee_{j=0}^{m} \varphi_j \quad \Rightarrow \quad \diamondsuit \varphi_0$$

If, in addition, $\varphi_0 = q$ and

$$\text{W1:} \quad p \quad \rightarrow \quad \bigvee_{j=0}^{m} \varphi_j,$$

then we can conclude:

$$\Phi \models p \quad \Rightarrow \quad \diamondsuit q.$$

Example 12. In Fig. 29, we present a ranked diagram which establishes that the response property

$$at\_\ell_0 \wedge at\_m_0 \wedge x = y = t_1 = t_2 = T = 0 \quad \Rightarrow \quad \diamondsuit(at\_\ell_4 \wedge at\_m_1).$$

is valid over program UP-DOWN.

Observe that the β assertions for nodes ϕ_1-ϕ_4 appear at the head of the compound nodes containing these nodes, as part of the encapsulation conventions. ▟

5.8 From Waiting-for to Response Properties

In many useful cases, we can infer response formulas from a waiting-for formula of a particular form.

Rule W→R, presented in Fig. 30, infers a response formula from a waiting-for formula of a special form.

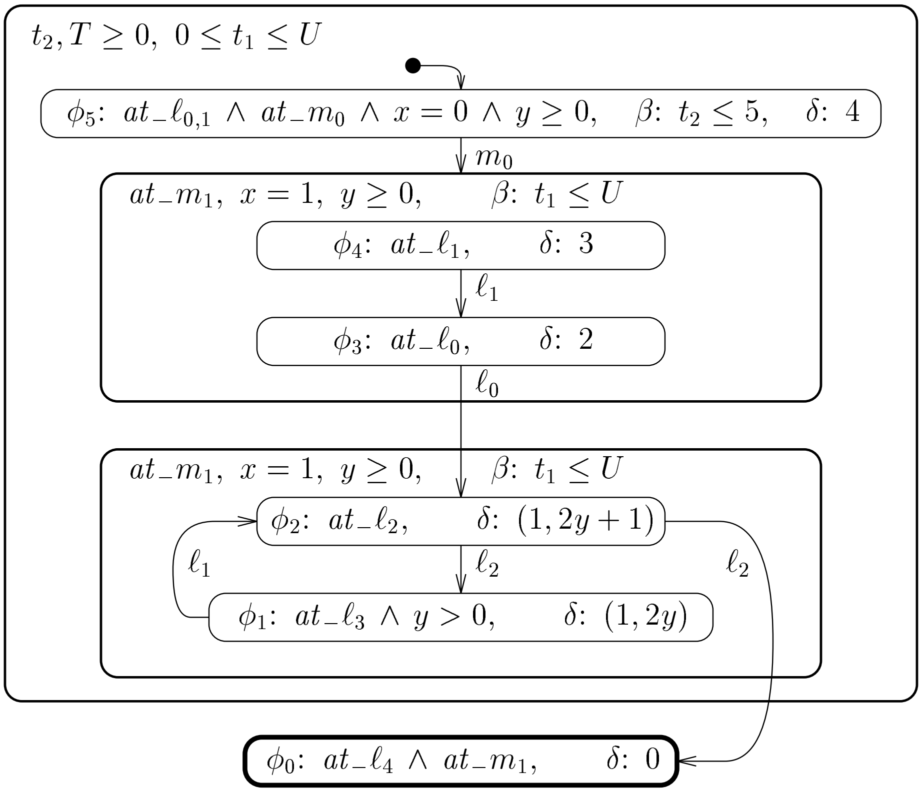

Fig. 29. A ranked diagram, establishing the formula
$at\_\ell_0 \wedge at\_m_0 \wedge x = y = t_1 = t_2 = T = 0 \quad \Rightarrow \quad \diamondsuit(at\_\ell_4 \wedge at\_m_1)$

Justification: Assume that the waiting-for premise is Φ-valid. Consider a Φ-computation σ, and a p-position $j \geq 0$ in σ. By the waiting-for formula, j initiates an interval, all of whose positions satisfy $q \wedge T_a \leq B$, which either extends to infinity or is terminated by an r-position. Since σ is a computation, T must grow beyond all values and cannot remain bounded by $a + b$ ($T_a \leq b$ is equivalent to $T \leq a + b$) at all positions. It follows that j must be followed by an r-position. ▟

Example 13. Consider the SPL$_T$ program UP-DOWN$_{[1,5]}$, which is program UP-DOWN with time bounds [1,5] uniformly assigned to all executable statements.

We use rule W→R to verify that the response formula

$$at\_\ell_0 \wedge at\_m_0 \wedge x = y = T_a = 0 \quad \Rightarrow \quad \diamondsuit(at\_\ell_4 \wedge at\_m_1 \wedge T_a \leq 50)$$

is valid over program UP-DOWN$_{[1,5]}$.

In Fig. 31, we present a waiting diagram which establisehs the UP-DOWN$_{[1,5]}$-

For assertions p, q, and r, and rigid expression B,

$$\frac{p \quad \Rightarrow \quad (q \wedge T_a \leq B)\,\mathcal{W}\,r}{p \quad \Rightarrow \quad \Diamond\, r}$$

Fig. 30. Rule W→R (from waiting-for to response formulas).

validity of the waiting-for formula

$$at\_\ell_0 \wedge at\_m_0 \wedge x = y = T_a = 0 \quad \Rightarrow \quad (T_a \leq 50)\,\mathcal{W}\,(at\_\ell_4 \wedge T_a \leq 50)$$

Note that the assertion describing the initial state does not specify initial values for either t_1 or t_2. To show that the waiting diagram is valid, we rely on the following invariant:

$$\Box\big((at\_\ell_0 \;\rightarrow\; 0 \leq t_1 \leq 5) \;\wedge\; (at\_m_0 \;\rightarrow\; 0 \leq t_2 \leq 5)\big).$$

This invariant can be separately established, using the methods of Section 4. ▟

5.9 Are Rules CB-CHAIN and CB-WELL Really Necessary?

Rule W→R enables the derivation of a response property from a timed waiting-for property, which can be established using rule WAIT. Rule WAIT (and its equivalent formulation in terms of waiting diagrams) is, in principle, simpler than either rule CB-CHAIN or rule CB-WELL, because it does not require the identification of explicit time bounds or ranking functions as auxiliary constructs.

In view of this, a naturally rising question is why do we need the response-specific rules CB-CHAIN and CB-WELL. Isn't rule W→R adequate for establishing all response properties of interest?

We provide two answers to this question. The first answer is that there are some response properties that cannot be established through timed waiting-for properties.

To support this point, consider again program UP-DOWN but with general (uniform) time bounds, $[L, U]$, such that $0 \leq L \leq U < \infty$. For all cases that $L > 0$, we can essentially repeat the analysis done in Example 13, and establish the waiting-for formula

$$at\_\ell_0 \wedge at\_m_0 \wedge x = y = T_a = 0 \quad \Rightarrow \quad (T_a \leq B)\,\mathcal{W}\,(at\_\ell_4 \wedge T_a \leq B),$$

where

$$B = (6 + 2\left\lfloor \frac{U}{2L} \right\rfloor)U.$$

Applying rule W→R to this formula, one can infer the response formula

$$at\_\ell_0 \wedge at\_m_0 \wedge x = y = T_a = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_4 \wedge T_a \leq B),$$

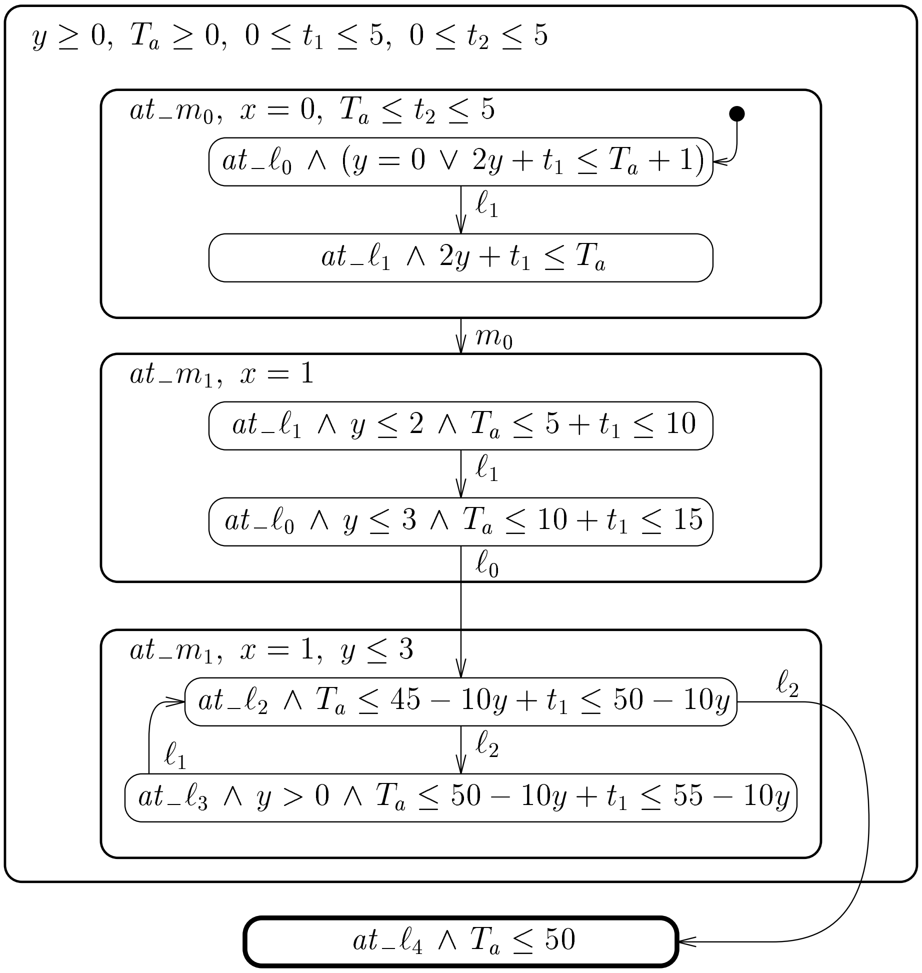

Fig. 31. A waiting diagram, establishing the formula
$at\_\ell_0 \wedge at\_m_0 \wedge x = y = T_a = 0 \quad \Rightarrow \quad (T_a \leq 50)\, \mathcal{W}\, (at\_\ell_4 \wedge T_a \leq 50)$

guranteeing termination within B time units.

One can see that as L gets closer to 0, the bound on termination time gets larger. It is therefore not surprising that when $L = 0$, there is no bound on the time it takes the program to terminate. Yet, all computations of this program eventually lead to the termination state at_ℓ_4. Thus, termination of program UP-DOWN in the case of $L = 0$ is a response property that cannot be verified using rule W→R. On the other hand, in Example 11, we established termination of UP-DOWN, using rule CB-WELL, in a proof that is valid for all $L \geq 0$. This illustrates the case of a response property that cannot be proven by rule W→R, but is provable by rule CB-WELL.

As the second answer justifying the introduciton of rule CB-WELL, we propose to compare the verification diagram of Fig. 29 with that of Fig. 31, both establishing termination of UP-DOWN for the time bounds $[1, 5]$ (Fig. 29 actually established it for general $[L, U]$). It is obvious that diagram 31 require a much more detailed analysis of the precise time interval which we can spend at each of the diagram nodes. In comparison, the diagram of Fig. 29 said very litle about these time intervals. The only timing information included in this diagram was that the time spent at each of the nodes is bounded by U. Thus, when we need or are ready to conduct a very precise analysis of the time intervals spent at each node, it makes sense to use waiting diagrams and rule W$\rightarrow$R. If, on the other hand, we are content with less quantitative anlysis, and are only interested in the qualitative fact that *eventually* q will occur (which is the essence of the $\diamondsuit$ termporal operator), we may use rule CB-WELL or rule CB-CHAIN. These rules may be conceptually more complicated than rule WAIT, but their application calls for a simpler analysis of the program.

6 Proving that a CTS is Non-Zeno

It is by now a widely accepted notion that the only interesting real-time systems are those which obey the non-zeno restriction. One of the reasons is that, since we only consider time-divergent runs as computations, a possibly-zeno system may contain components that will never be accessed in a computation. In some sense, these components are redundant to the description of the system and their inclusion is superfluous and often confusing and misleading. Non-zeno systems, on the other hand, contain no such redundancy, since every accessible state also appears in some computation.

In view of the significance of the non-zeno restriction, it is important to be able to verify that an arbitrary given CTS is non-zeno.

In many cases, there are simple sufficient conditions which guarantee that the system is non-zeno. One of the most important case is the following:

Claim 8 *Let P be an SPL$_T$ program in which the upper bound assigned to each executable statement is positive. Then F_P, the CTS corresponding to P, is a non-zeno system.*

Justification Let $U_m > 0$ be the minimal upper bound. Consider a finite run r: $s_0, \ldots, s_k$. We wish to show that r can be extended into a computation.

The recipe for extending r considers the last reached state s (starts at $s = s_k$ and decides to apply the next transition as follows:

- If the *tick* transition is enabled on s, take the *tick* transition with increment $\Delta > 0$, which is the maximal $\Delta \leq 1$ satisfying $s \models \Omega(\Delta)$.
- If the *tick* transition is disabled on s, it must be blocked by one of the processes, say P_i, whose clock t_i has reached the upper bound of some transition τ of P_i which is currently enabled. In this case, take this ripe transition τ.

It is not difficult to check that an accessible state in a clocked transition systems derived from a program with positive upper bounds always has at least one extended transition enabled on it. Thus, the described recipe produces an infinite run. This observation hinges on the revised transition relation we associated with the *wait* statement.

It only remains to check that the infinite run produced by this recipe is time-divergent and, hence, is a computation. By considering the different possibilities, we observe that two cases are possible.

In one case, we eventually reach a state s such that $s \Vdash \Omega(\Delta)$, for every $\Delta > 0$. In this case, once we reach this s, we continue to take *tick* steps with $\Delta = 1$.

In the remaining case, for every reached state s_i, there exists a limit Δ_i such that $s_i \Vdash \Omega(\Delta)$ for no $\Delta > \Delta_i$. In this case, our construction must take infinitely many untimed transitions, i.e., transitions $\tau \neq$ *tick*. Note that each such transition resets one of the clocks to 0. It follows that the construction of the infinite run causes at least one of the clocks, say t_i, to be reset to 0 infinitely many times. It is not difficult to see that between two consecutive resets of clock t_i, time must progress by at least U_m. It follows that time has progressed infinitely many time by the amount $U_m >$ and, therefore, the run is time-divergent. ■

6.1 A Rule for Establishing Non-Zenoness

While Claim 8 settles the question of non-zenoness for many useful cases, there are additional cases which require different methods. The claim was established for the simpler case that the upper bounds assigned to transitions were positive constants. It can easily be generalized to state-dependent upper bounds which are bounded from below by a positive constant. However, this still does not cover all possible cases. For example, CTS Φ_4 presented in Fig. 32, is a non-zeno system, even though, the upper bounds of the transition connecting ℓ_0 to itself have no lower bound.

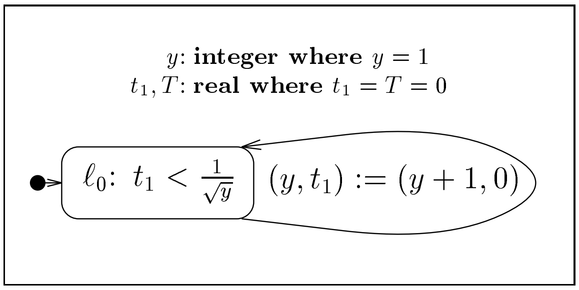

Fig. 32. A nonzeno CTS Φ_4 with upper bounds tending to 0.

The general strategy we propose for proving that a given CTS Φ is non-zeno is summarized in rule NONZ, presented in Fig. 33.

For assertion φ,

N1. $\Phi \Vdash \varphi$

N2. $\Phi \models AG(\varphi \wedge V = V_0 \wedge T_a = 0 \quad \rightarrow \quad EF(T_a > 0))$

Φ is non-zeno

Fig. 33. Rule NONZ (Φ is a non-zeno CTS).

The rule uses an auxiliary assertion φ. Premise N1 requires that φ is Φ-state valid, and can be proven using rule ACC.

Premise N2 belongs to the realm of branching-time temporal logic, which is different from the linear-time temporal framework we have been consistently using in this paper. It has long been observed that the property of being non-zeno cannot be formulated in linear-time TL and needs branching-time TL for its precise formulation. In a recent paper ([Lam95]), Lamport makes this observation but suggests a method by which properties such as non-zenoness can still be verified in a linear framework. We prefer to use the branching-time logic CTL([EC82]) for formulating the required property, as in premise N2, and present a single proof rule which is adequate to establish CTL formulas such as the one presented in N2. A deductive system for verifying the main CTL properties appeared in [BAMP83]. A more comprehensive deductive system for CTL was recently proposed in [FG95].

6.2 Verifying Possibility Formulas

A *possibility formula* is a CTLformula of the form

$$AG(p \rightarrow EFq),$$

for assertions p and q. Without entering into the individual meaning of the CTL temporal operators AG and EF, we say that the possibility formula $AG(p \rightarrow EFq)$ is *valid over* CTS Φ (*Φ-valid*) if

> For every accessible p-state s, there exists a run segment $s{=}s_1, \ldots, s_k$ leading from s to a q-state s_k.

We write

$$\Phi \models AG(p \rightarrow EFq)$$

to indicate that the possibility formula $AG(p \rightarrow EFq)$ is Φ-valid.

For assertions p, q, and $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$,
transitions $\tau_1, \ldots, \tau_m \in \mathcal{T}_T$,
functions $Next_1, \ldots, Next_m$: $\Sigma \mapsto \Sigma$,
a well-founded domain $(\mathcal{A}, \succ)$, and
ranking functions $\delta_0, \ldots, \delta_m$: $\Sigma \mapsto \mathcal{A}$,

$$\text{G1.} \quad p \quad \rightarrow \quad \bigvee_{j=0}^{m} \varphi_j$$

The following premise holds for $i = 1, \ldots, m$

$$\text{G2.} \quad \varphi_i \ \wedge \ V' = Next_i \quad \rightarrow \quad \rho_{\tau_i} \ \wedge \ \bigvee_{j=0}^{m} (\varphi'_j \ \wedge \ \delta_i \succ \delta'_j)$$

$$\Phi \models AG(p \quad \Rightarrow \quad EFq)$$

Fig. 34. Rule G-POSS (Φ validity of a possibility formula).

In Fig. 34, we present rule POSS which is sound and complete for proving the Φ-validity of a possibility formula.

The rule requires finding auxiliary assertions φ_i, functions $Next_i$: $\Sigma \mapsto \Sigma$, and transitions τ_i, $i = 1, \ldots, m$, a well-founded domain $(\mathcal{A}, \succ)$, and ranking functions δ_i: $\mathcal{S} \mapsto \mathcal{A}$. Each assertion φ_i is associated with the transition τ_i that is helpful at positions satisfying φ_i, with a function $Next_i$ that selects a successor state, and with its own ranking function δ_i. We have presented the successor-selection functions $Next_i$ as mapping states to states but, in fact, they map $s[V]$, i.e., the values of the system variables in state s, to $s'[V]$, the values of the system variables in a successor state s'.

Premise G1 requires that every p-position satisfies one of $\varphi_0, \ldots, \varphi_m$.

Premise G2 considers a φ_i-state s and a state $\widetilde{s}$ such that $\widetilde{s}[V] = s[V'] = Next_i$, for some $i = 1, \ldots, m$. The premise requires that $\widetilde{s}$ is a τ_i-successor of s which satisfies φ_j, for some $j = 0, \ldots, m$ and has a rank lower than that of s.

Justification Let s be a p-state. We will show that there exists a run segment $s{=}s_1, \ldots, s_k$ which leads from s to a q-state. By premise G1, $s = s_1$ must satisfy φ_j, for some $j = 0, \ldots, m$. Let j_1 be the minimal j such that $s \Vdash \varphi_j$. If $j = 0$, we are done, since s_1 satisfies $\varphi_0 = q$.

Otherwise, $j_1 > 0$ and let $u_1 = \delta_{j_1}(s_1)$. Let s_2 be any state such that $s_2[V] = Next_{j_1}(s_1[V])$. By premise G2, s_2 is a τ_{j_1}-successor of s_1, satisfies φ_{j_2} for some $j_2 \in \{0, \ldots, m\}$, and a rank $u_2 = \delta_{j_2}(s_2) \prec u_1$. If $j_2 = 0$, we are done. Otherwise, we take s_3 to be some $Next_{j_2}$-selected successor of s_2, and so on.

This construction can terminate when we reach some k such that $j_k = 0$. If it does no terminate, we generate an infinite descending sequence

$$u_1 \succ u_2 \succ \cdots .$$

This is impossible due to the well-foundedness of $\mathcal{A}$. We conclude that the construction must terminate in a state s_k satisfying $\varphi_0 = q$. ▟

Example 14. We illustrate the application of rule G-POSS for proving that the possibility formula

$$AG\big(\underbrace{at\_\ell_{0,1} \wedge T_a = 0}_{p} \quad \rightarrow \quad EF(\underbrace{T_a \geq 1}_{q})\big)$$

is valid over CTS Φ_5, presented in Fig. 35.

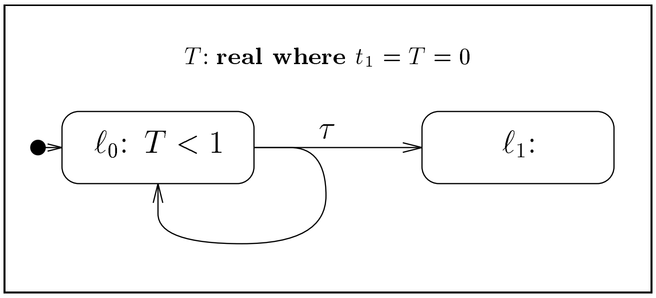

Fig. 35. A nonzeno CTS Φ_5.

Observe that transition τ, the only transition in Φ_5, has two successors. One satisfying at_ℓ_0 and the other satisfying at_ℓ_1.

To apply rule G-POSS, we take $(\mathbb{N}, >)$ (the domain of the natural numbers) as the well-founded domain. The auxiliary assertions, helpful transitions, successor-selection functions, and ranking functions are given by the following table:

φ_0: $T_a \geq 1$	τ_0: *tick*	$Next_0$: $(\pi' : 1,\ T' : T+1)$	δ_0: 0
φ_1: $at\_\ell_1 \wedge T_a \geq 0$	τ_1: *tick*	$Next_1$: $(\pi' : 1,\ T' : T+1)$	δ_1: 1
φ_2: $at\_\ell_0 \wedge T_a \geq 0$	τ_2: τ	$Next_2$: $(\pi' : 1,\ T' : T)$	δ_2: 2

This choice of constructs, corresponds to a strategy of constructing a run segment in which time will progress by at least 1. According to this strategy, if we are at ℓ_0, we choose to take transition τ and choose a τ-successor state satisfying at_ℓ_1. If we are at ℓ_1, we choose to take the *tick* transition with time increment 1.

It is not difficult to check that all premises of rule G-POSS are satisfied by this choice of constructs.

It follows that the possibility formula

$$AG\big(at\_\ell_{0,1} \wedge T_a = 0 \quad \rightarrow \quad EF(T_a \geq 1)\big)$$

is Φ_5-valid. ▟

Systems with Deterministic Transitions

Rule G-POSS is very general and can be shown to be complete for proving possibilities properties of all clocked transition systems. However, there is a big family of systems which can be handled by a rule which call for identification of simpler constructs.

A transition τ of a CTS is called *deterministic* if all the τ-successors of an accessible state assign the same values to the system variables. That is, if we restrict our attention to the values of the system variables, each accessible state has at most one τ-successor. We say that a CTS Φ is *transition-deterministic* (Φ is a TD-CTSS for short) if all of its (untimed) transitions are deterministic. All the systems we have presented in this paper, excluding Φ_5, are transition-deterministic.

Consider a TD-CTSS Φ. Assume that we wish to construct the successor-selection function $Next_i$ corresponding to assertion φ_i, where $\tau_i \neq tick$. Since τ_i is deterministic, $Next_i$ is uniquely determined, and its explicit specification is redundant. The situation is different with the *tick* transition, which may have (uncountably) many successors, each corresponding to a different value of the time increment Δ. However, once we specify the value of Δ, the successor of a *tick* transition is also uniquely determined up to differences in non-system variables.

Thus, instead of specifying a complete successor function, specifying the values of all system variables in the successor state, it is sufficient to specify the time increment Δ_i associated with the successor. For uniformity, we specify values of Δ_i also for untimed transitions, but then we ensure that $\Delta_i = 0$.

This leads to rule POSS (Fig. 36) which is adequate for proving possibility properties of every TD-CTSS. Rule POSS has more premises than rule G-POSS but it requires the identification of simpler constructs, and the premises are easier to verify.

Note that rule POSS splits premise G2 of rule G-POSS into two premises. Premise P2 guarantees that the (unique) successor of a φ_i-state corresponding to the identification of the helpful transition τ_i and the time increment Δ_i (if it exists), satisfies some φ_j with a lower rank. Premise P3 guarantees that each φ_i state does have a successor corresponding to τ_i and Δ_i. The premise is split into the case of the *tick* transition (sub-premise P3t) and the case of all other untimed transitions (sub-premise P3n). Sub-premise P3t requires that φ_i imply Ω, the enabling condition of of transition *tick*, with the specified Δ_i. Sub-premise P3n requires that $\Delta_i = 0$ and that φ_i implies the enableness of τ_i.

Example 15. We illustrate the use of rule POSS for proving that the possibility formula

$$AG(\underbrace{at\_\ell_0 \,\wedge\, 1 \leq b = y \,\wedge\, 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \,\wedge\, 0 = T_a}_{p} \quad \rightarrow \quad EF(\underbrace{T_a \geq 1}_{q}))$$

is valid over CTS Φ_4.

For assertions p, q, and $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$,
transitions $\tau_1, \ldots, \tau_m \in \mathcal{T}_T$,
time increments $\Delta_1, \ldots, \Delta_m \geq 0$,
a well-founded domain $(\mathcal{A}, \succ)$, and
ranking functions $\delta_0, \ldots, \delta_m$: $\Sigma \mapsto \mathcal{A}$,

P1. $p \rightarrow \bigvee_{j=0}^{m} \varphi_j$

The following premises hold for $i = 1, \ldots, m$

P2. $\rho_{\tau_i} \wedge T' = T + \Delta_i \wedge \varphi_i \rightarrow \bigvee_{j=0}^{m} (\varphi'_j \wedge \delta_i \succ \delta'_j)$

P3. If $\tau_i = tick$ then

P3t. $\varphi_i \rightarrow \Omega(\Delta_i)$

Otherwise

P3n. $\varphi_i \rightarrow \Delta_i = 0 \wedge En(\tau_i)$

$\Phi \models AG(p \Rightarrow EFq)$

Fig. 36. Rule POSS (Φ validity of a possibility formula).

Note that CTS Φ_4 has a single untimed transition, to which we refer as τ_{00}.

As the well-founded domain, we take $(\mathbb{N}^2, \succ)$, the domain of lexicographic pairs. The auxiliary assertions, helpful transitions, time increments, and ranking functions are given by the following table:

$$
\begin{array}{llll}
\varphi_0: & T_a \geq 1 \quad \tau_0: \ \tau_{00} \quad \Delta_0: \ 0 & & \delta_0: \ 0 \\
\varphi_1: & \multicolumn{3}{l}{at\_\ell_0 \wedge 1 \leq b \leq y \leq 2b+1 \wedge t_1 = 0 \wedge \frac{y-b-1}{\sqrt{2b+1}} \leq T_a < 1} \\
 & \qquad \tau_1: \ tick \quad \Delta_1: \ \frac{1}{\sqrt{y}} & & \delta_1: \ (|2b+1-y|, 2) \\
\varphi_2: & \multicolumn{3}{l}{at\_\ell_0 \wedge 1 \leq b \leq y \leq 2b+1 \wedge t_1 = \frac{1}{\sqrt{y}} \wedge \frac{y-b}{\sqrt{2b+1}} \leq T_a < 1} \\
 & \qquad \tau_2: \ \tau_{00} \quad \Delta_2: \ 0 & & \delta_2: \ (|2b+1-y|, 1) \\
\varphi_3: & \multicolumn{3}{l}{at\_\ell_0 \wedge 1 \leq b \leq y \leq 2b+1 \wedge 0 < t_1 < \frac{1}{\sqrt{y}} \wedge 0 \leq T_a < 1} \\
 & \qquad \tau_3: \ tick \quad \Delta_3: \ \frac{1}{\sqrt{y}} - t_1 & & \delta_3: \ 2b+2
\end{array}
$$

The idea behind this selection is the following. Starting in a state at which $y = b \geq 1$ and $0 < t_1 < \frac{1}{\sqrt{y}}$ (described by assertion φ_3), we first step time with an increment $\frac{1}{\sqrt{y}} - t_1$, this will get us to a state in which $t_1 = \frac{1}{\sqrt{y}}$ (described by φ_2). From this point on, we alternate between taking untimed transition τ_{00} which increments y by 1 but preserves time, and taking transition *tick*, which increments time by $\frac{1}{\sqrt{y}}$ but preserves the value of y. We repeat this couple of steps at most $b+1$ times, letting y increase from b to $2b+1$. Since the time step in each round is decreasing, the total time increase is not less than $b+1$ times the last time increment which is $\frac{1}{\sqrt{2b+1}}$. Thus the total time increase is not less than

$$\frac{b+1}{\sqrt{2b+1}} \;\geq\; 1,$$

where the inequality holds for every $b \geq 1$.

This informal argument can be formalized by checking that all premises of rule POSS are state valid for the specified selection of the auxiliary assertions, helpful transitions, time increments, and well-founded ranking.

This establishes that the possibility formula

$$AG(at\_\ell_0 \wedge 1 \leq b = y \wedge 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \wedge 0 = T_a \quad \rightarrow \quad EF(T_a \geq 1))$$

is valid over CTS Φ_4. ▟

6.3 Possibility Diagrams

Proofs according to rule POSS can be succinctly represented by *possibility diagrams*.

A possibility diagram is a directed labeled graph constructed as follows:

- *Nodes* in the graph are labeled by assertions $\varphi_0, \ldots, \varphi_m$. We will often refer to a node by the assertion labeling it. In addition to the assertion, each node is also labeled by a ranking function δ.
  The node labeled by φ_0 is called the *terminal node* and is graphically identified by being drawn with bold-face boundaries.
- *Edges* in the graph represent transitions between assertions. Each edge is labeled by the name of an untimed transition in the program or by a label of the form $tick(\Delta)$. We refer to an edge labeled by τ (including the case that $\tau = tick$) as a *τ-edge*. All edges departing from the same node must have the same label. Every non-terminal node must have an edge departing from it. No edges depart form the terminal node.
- Some of the nodes are designated as *initial nodes*. They are annotated by an entry arrow.

Verification Conditions Implied by a Chain Diagram

Consider a nonterminal node labeled by assertion φ and ranking function *del*. Let $\varphi_1, \ldots, \varphi_k$, $k > 0$, be the successors of φ by edges departing from φ (possibly including φ itself). With each such node, we associate the following verification condition:

- If the label of all edges departing from φ is $tick(\Delta)$, then we require the following verification conditions to hold:

P2. $\quad \rho_{tick} \wedge T' = T + \Delta \wedge \varphi \quad \rightarrow \quad (\varphi_1' \wedge \delta \succ \delta_1') \wedge \cdots \wedge (\varphi_k' \wedge \delta \succ \delta_k')$

P3t. $\quad \varphi \quad \rightarrow \quad \Omega(\Delta)$

- If the label of all edges departing from φ is $\tau \neq tick$, then we require the following verification conditions to hold:

  P2. $\quad \rho_\tau \;\wedge\; T' = T \;\wedge\; \varphi \;\;\rightarrow\;\; (\varphi'_1 \wedge \delta \succ \delta'_1) \;\wedge\; \cdots \;\wedge\; (\varphi'_k \wedge \delta \succ \delta'_k)$

  P3n. $\quad \varphi \;\;\rightarrow\;\; En(\tau)$

Valid Possibility Diagrams

A possibility diagram is said to be *valid over* CTS Φ (*Φ-valid* for short) if all the verification conditions associated with the diagram are P-state valid.

The consequences of having a valid chain diagram are stated in the following claim:

Claim 9 *If D is a Φ-valid possibility diagram with nodes $\varphi_0, \ldots, \varphi_m$, then*

$$\Phi \models AG\Big(\bigvee_{j=0}^{m} \varphi_j \quad \rightarrow \quad EF\varphi_0\Big)$$

If, in addition, $\varphi_0 = q$, and

$$p \quad \rightarrow \quad \bigvee_{j=0}^{m} \varphi_j,$$

then we can conclude:

$$\Phi \models AG(p \quad \rightarrow \quad EFq).$$

Example 16. In Fig. 37, we present a waiting diagram that establishes the possibility property

$$AG(at\_\ell_0 \;\wedge\; 1 \leq b = y \;\wedge\; 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \;\wedge\; 0 = T_a \quad \rightarrow \quad EF(T_a \geq 1))$$

for system Φ_4. ▟

6.4 Proving that Φ_4 is Non-Zeno

We conclude this discussion by applying rule NONZ to show that CTS Φ_4 is non-zeno.

As the assertion φ required by rule NONZ, we take

$$\varphi: \quad at\_\ell_0 \;\wedge\; y \geq 1 \;\wedge\; 0 \leq t_1 \leq \frac{1}{\sqrt{y}}.$$

It is not difficult to show by rule ACC that φ is Φ_4-state valid. This establishes premise N1 of rule NONZ.

For premise N2 of NONZ, we have to verify the possibility formula

$$AG(at\_\ell_0 \wedge y \geq 1 \wedge 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \wedge y = b \wedge T_a = 0 \quad \rightarrow \quad EF(T_a \geq 1)).$$

This is essentially done in Example 15.

We conclude that CTS Φ_4 is non-zeno.

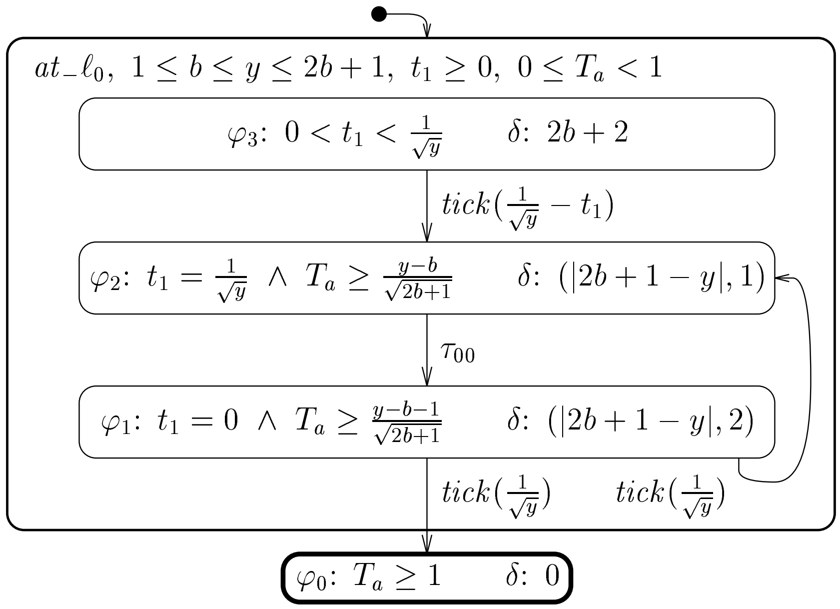

Fig. 37. A possibility diagram, establishing the formula
$AG(at\_\ell_0\ \wedge\ 1 \le b = y\ \wedge\ 0 \le t_1 \le \frac{1}{\sqrt{y}}\ \wedge\ 0 = T_a \quad \rightarrow \quad EF(T_a \ge 1))$

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 careful reading and critical comments of Oded Maler, Henny Sipma, and Tomás Uribe.

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, 1994.

[BAMP83] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[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.

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

[FG95] L. Fix and O. Grumberg. Verificaiton of temporal properties. to appear in *Logic and Computation*, 1995.

[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.

[Lam95] L. Lamport. Proving possibiity properties. Technical Report 137, Digital Equipment Corporation, Systems Research Center, Palo Alto, July 1995.

[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 STAN-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.

[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.

[MP93] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 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.

[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.