Acta Informatica 36, 837–912 (2000)

Verification of clocked and hybrid systems

Yonit Kesten[1], **Zohar Manna**[2], **Amir Pnueli**[3]

[1] Department of Communication Systems Engineering, Ben Gurion University, Beer-Sheva, Israel (e-mail: ykesten@bgumail.bgu.ac.il)
[2] Department of Computer Science, Stanford University, Stanford, CA 94305, USA (e-mail: manna@cs.stanford.edu)
[3] Department of Computer Science, Weizmann Institute, Rehovot, Israel (e-mail: amir@wisdom.weizmann.ac.il)

Received: 23 September 1998 / 7 June 1999

Abstract. This paper presents a new computational model for real-time systems, called the *clocked transition system* (CTS) model. The CTS 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. We present verification rules for proving safety and liveness properties of clocked transition systems. All rules are associated with verification diagrams. The verification of *response* properties requires adjustments of the proof rules developed for untimed systems, reflecting the fact that progress in the real time systems is ensured by the progress of time and not by fairness. The style of the verification rules is very close to the verification style of untimed systems which allows the (re)use of verification methods and tools, developed for untimed reactive systems, for proving all interesting properties of real-time systems.

We conclude with the presentation of a branching-time based approach for verifying that an arbitrary given CTS is *non-zeno*.

Finally, we present an extension of the model and the invariance proof rule for hybrid systems.

This research was supported in part by the National Science Foundation under grant CCR-95-27927, the Defense Advanced Research Projects Agency under NASA grant NAG2-892, ARO under grant DAAH04-95-1-0317, ARO under MURI grant DAAH04-96-1-0341, by Army contract DABT63-96-C-0096 (DARPA), by a basic research grant from the Israeli Academy of Sciences, and by the European Community ESPRIT Basic Research Action Project 6021 (REACT). Preliminary versions of some parts of this paper appeared in [KMP95] and [MP95a].

Contents

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*).

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

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

The computational model proposed for reactive systems is that of a *fair transition system* (FTS) [MP93b].

The approach to real time presented in [MP93a] and [HMP94] is based on the computational model of *timed transition systems* (TTS) in which time itself is not explicitly represented but is reflected in a time stamp affixed to each state in a computation of a TTS.

In this paper we present a new computational model for real-time systems, the *clocked transition system* (CTS) model. 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. The CTS model can be viewed as a natural first-order extension of the timed automata model [AD94].

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.

Consider, for example, the following two important timed properties:

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

Within the CTS computational model, these two yardstick properties can be specified by the following (unextended) temporal formulas:

– Bounded response: $\quad p \wedge (T = t_0) \Rightarrow \Diamond(q \wedge T \leq t_0 + d).$

– Minimal separation: $\quad p \wedge (T = t_0) \Rightarrow \Box(T < t_0 + d \rightarrow \neg q).$

The new computational model has several advantages over previous models such as the model of *timed transition systems* (TTS, see [HMP94]).

The first advantage of the new model, as shown above, is that it leads to a more natural style of specification, explicitly referring to clocks, which are just another kind of system variables, instead of introducing special new constructs, such as the *bounded temporal operators* proposed in *metric temporal logic* (MTL) (see [KVdR83], [KdR83], and [Koy90]) or the *age function* proposed in [MP93a].

A second advantage of the CTS model is that we can reuse many of the methods and tools developed for verifying untimed reactive systems (e.g. [MP95]) for verifying real-time systems under the CTS model. The move from TTS to CTS brings us closer to the approach proposed in [AL91], which also recommends handling real time with a minimal extension of the reactive-systems formalism.

The model of Clocked Transition Systems, as presented in this paper, has been successfully implemented in the *Stanford Temporal Verifier* support system STeP[BBC$^+$95]. We refer the reader to the paper [BMSU97] which uses clocked transition systems to model and verify the generalized railroad crossing benchmark problem.

A model similar to the CTS model presented here was introduced in [AH94], and proof rules for establishing response properties for this model were presented in [HK94]. However, the response verification rules presented there for the general case were based on consideration of the region graph associated with timed automata which, in many cases, becomes very big. Our approach to response verification, while considering the general case, does not refer to the region graph and can be viewed as a natural modification of the response rules for untimed fair transition systems, except that the notion of fairness is replaced by the guaranteed progress of time.

We refer the reader to [AH89], [Ost90], [AL91], 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 Sect. 2, we present the real-time computational model of *clocked transition systems* (CTS). In Sect. 3, we show how programs augmented with timing bounds for the execution of statements can be represented as clocked transition systems. In Sect. 4, we present rules and verification diagrams for verifying safety properties of CTS's. In Sect. 5, we present rules and verification diagrams for establishing response properties of clocked transition systems. In Sect. 6, we present an approach to verifying that a given CTS is non-zeno. Finally, in Sect. 7, we present the extension of the CTS model to deal with general hybrid systems. This yields an extended model to which we refer as a *phase transition system*. A proof rule for verifying safety properties of hybrid systems is introduced and illustrated.

A part of an ongoing research has implemented support for phase transition systems in STeP [MS98], and has used it to successfully (and actually without too much user interaction) verify a few of the HyTech examples [HHWT95].

An earlier workshop version of this paper appeared in [KMP98].

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 : A finite set of *system variables*. The set $V = D \cup C$ is partitioned into $D = \{u_1, \ldots, u_n\}$ the set of *discrete variables* and $C = \{t_1, \ldots, t_k\}$ 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

$$\Theta \quad \rightarrow \quad t_1 = \ldots = t_k = T = 0,$$

i.e., the clocks are reset to zero 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.
We say that a transition $\tau \in \mathcal{T}$ is *enabled* (denoted $En(\tau)$) in some state s, if the following formula is satisfied:

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

Thus, $En(\tau)$ is true in s iff s has some τ-successor.
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:

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

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 $C = \{t_1, \ldots, t_k, T\}$ be the set of its clocks. Then, the expression $C' = C + \Delta$ is an abbreviation for

$$t_1^{'} = t_1 + \Delta \;\wedge\; \cdots \;\wedge\; t_k^{'} = t_k + \Delta \;\wedge\; T' = T + \Delta,$$

and $\Pi(D, C + t)$ is an abbreviation for $\Pi(u_1, \ldots, u_m, t_1 + t, \ldots, t_k + t, T + t)$.

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*: $s_0 \models \Theta$
- *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

$$\Pi: \quad \bigwedge_{i \in N} (p_i \rightarrow t_i < E_i),$$

where N is some finite index set and, for each $i \in N$, 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 time-increment

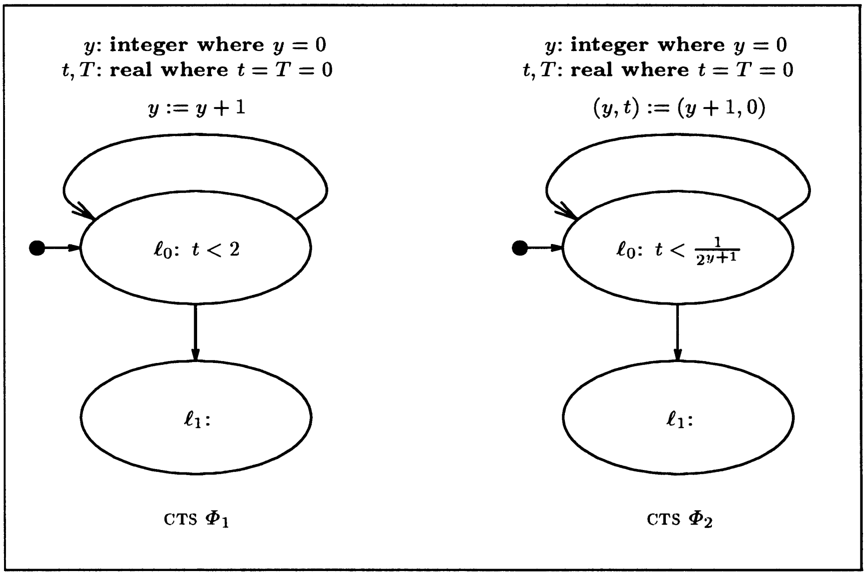

Fig. 1. Two CTS's

limiting formula $\Omega(\Delta)$ can be significantly simplified and assumes the following form:

$$\Omega(\Delta): \quad \Delta > 0 \ \wedge \ \bigwedge_{i \in N} (p_i \rightarrow t_i + \Delta \leq E_i)\,.$$

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 [AL91], [Hen92]). An equivalent formulation is that Φ is non-zeno if it satisfies the following

> 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.

It is not difficult to establish that both Φ_1 and Φ_2 are non-zeno CTS's. This is because, 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.

$V\colon \underbrace{\{\pi\colon \{0,1\};\ y\colon \textbf{integer}\}}_{D} \cup \underbrace{\{t,\ T\colon \textbf{real}\}}_{C}$ $\Theta\colon\ \pi = y = t = T = 0$ $\mathcal{T}\colon\ \{\tau_0, \tau_1\}$ with transition relations $\tau_0\colon\ \pi = \pi' = 0\ \wedge\ y' = y + 1$ $\wedge\ (t, T) = (t', T')$ $\tau_1\colon\ \pi = 0\ \wedge\ \pi' = 1\ \wedge\ (y, t, T) = (y', t', T')$ $\Pi\colon\ \pi = 0 \rightarrow t < 2$ CTS Φ_1	$V\colon \underbrace{\{\pi\colon \{0,1\};\ y\colon \textbf{integer}\}}_{D} \cup \underbrace{\{t,\ T\colon \textbf{real}\}}_{C}$ $\Theta\colon\ \pi = y = t = T = 0$ $\mathcal{T}\colon\ \{\tau_0, \tau_1\}$ with transition relations $\tau_0\colon\ \pi = \pi' = t' = 0\ \wedge\ y' = y + 1$ $\wedge\ T = T'$ $\tau_1\colon\ \pi = 0\ \wedge\ \pi' = 1\ \wedge\ (y, t, T) = (y', t', T')$ $\Pi\colon\ \pi = 0 \rightarrow t < \frac{1}{2^{y+1}}$ CTS Φ_2

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

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

$$\begin{aligned}\rho^1_{tick}\colon \exists \Delta > 0.\ (\pi', y') = (\pi, y)\ \wedge\ (t', T') \\ = (t + \Delta, T + \Delta)\ \wedge\ (\pi = 0 \rightarrow t + \Delta \leq 2) \\ \rho^2_{tick}\colon \exists \Delta > 0.\ (\pi', y') = (\pi, y)\ \wedge\ (t', T') \\ = (t + \Delta, T + \Delta)\ \wedge\ \left(\pi = 0 \rightarrow t + \Delta \leq \frac{1}{2^{y+1}}\right).\end{aligned}$$

Following is a computation of CTS Φ_1:

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

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

2.2 Specification language

To specify properties of reactive systems, we use the language of temporal logic, as presented in [MP93b]. Here, we only use the following:

– *State formulas* (*assertions*) - any first-order formula, possibly including at_ℓ expressions

- $\Box p$ — 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 $s_j \Vdash p$. Satisfaction of (the three considered) temporal formulas over a model σ is defined as follows:

- A model σ satisfies the invariance formula $\Box p$, written $\sigma \models \Box p$, if all positions within σ are p-positions.
- A model σ satisfies the waiting-for formula $p \Rightarrow (q\mathcal{W}r)$, 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,

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

- A model σ satisfies the response formula $p \Rightarrow \Diamond r$, written $\sigma \models p \Rightarrow \Diamond r$, 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.

A temporal formula is specified over a set of variables, partitioned into flexible and rigid variables. *Flexible variables* may assume different values in different states. *Rigid variable* must assume the same value in all states.

The temporal formulas that specify program properties can be arranged in a hierarchy that identifies several classes of formulas, differing in their expressive power. For a full presentation of the hierarchy, we refer the reader to [MP95] (chap. 0), and to [MP93b] for a more extensive discussion. In this paper, we present proof rules for the verification of two subclasses of the general class of safety properties (state-invariance and waiting-for properties), and a subclass of the general class of liveness properties (response). These restricted sets of properties are sufficient to demonstrate the similarity between the verification methods developed for untimed reactive systems and the verification methods that can be used with the CTS and PTS models for real time and hybrid systems.

3 Programs as clocked transition systems

In this section we show how to represent real-time programs as clocked transition systems. First we introduce a simple conccurrent programming language in which example programs will be written. Next we show the representation of such programs as CTS.

3.1 A simple programming language

In [MP93b], we introduced a simple programming language SPL. Here we consider a subset of the language, restricting our attention to the following statements:

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

In this restricted subset, concurrent processes communicate by shared variables, and parallelism is allowed only at the top level of the program.

We start by presenting the syntax of statements and programs in SPL.

Basic statements. First, we consider basic statements. These are statements that can be executed in a single atomic step

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

$$y \ := \ e$$

is an *assignment* statement. We use the **skip** statement as an abbreviation for a trivial assignment $y := y$.

– *Await*: For a boolean expression c,

await c

is an *await* statement. We refer to condition c as the *guard* of the statement. Execution of **await** c changes no variables. Its sole purpose is to wait until c becomes true, at which point it terminates, allowing the execution of subsequent statements.

Schematic statements. The following statements provide schematic representations of segments of code that appear in programs for solving the mutual-exclusion problem. Typically, we are not interested in the internal details of this code but only in its overall behavior concerning termination.

– *Noncritical*:

noncritical

is a *noncritical* statement. This statement represents the noncritical activity in programs for mutual exclusion. It is not required that this statement terminate. The name “noncritical” given to this statement is appropriate for programs that deal with critical sections.

– *Critical*:

critical

is a *critical* statement. This statement represents the critical activity in programs for mutual exclusion, where coordination between the processes is required. It is required that this statement terminate.

Compound statements. Compound statements consist of a controlling frame applied to one or more sub-statements, to which we refer as the *children* of the compound statement.

– *Conditional*: For statements S_1 and S_2 and a boolean expression c,

if c **then** S_1 **else** S_2

is a *conditional* statement. Its intended meaning is that the boolean condition c is evaluated and tested. If the condition evaluates to T (true), statement S_1 is selected for subsequent execution; otherwise, if the condition evaluates to F (false), S_2 is selected. Thus, the first step in an execution of the conditional statement is the evaluation of c and the selection of S_1 or S_2 for further execution. Subsequent steps continue to execute the selected sub-statement.

A special case of the conditional statement is the *one-branch-conditional* statement

if c **then** S_1.

Execution of this statement in the case that c evaluates to F terminates in one step.

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

$S_1; \cdots; S_k$

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

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

– *Selection*: For statements $S_1, \ldots, S_k$,

S_1 **or** $\cdots$ **or** S_k

is a *selection* statement. Its intended meaning is a nondeterministic selection of a statement S_i and its execution. The first step in the execution of the selection statement selects a statement S_i, $i = 1, \ldots, k$, that is currently enabled (ready to be executed) and performs the first step in the execution of S_i. Subsequent steps proceed to execute the rest of the selected sub-statement, ignoring the other S_j's. If more than one of $S_1, \ldots, S_k$ is enabled, the selection is nondeterministic. If none of the branches are enabled, execution of the selection statement is delayed.

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

while c **do** S

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

We introduce the notation

loop forever do S

as a synonym for

while T **do** S.

Another useful abbreviation is the *for* statement

for $i := 1$ **to** m **do** S,

which is an abbreviation for the concatenation

$i := 1;$ **while** $i \leq m$ **do** $[S;\ i := i + 1]$.

Programs. A program P has the form

$$P :: \ \left[\text{declaration; } [P_1 :: \ [\ell_1\!: \ S_1; \ \ \widehat{\ell_1} :] \parallel \cdots \parallel P_m :: \ [\ell_m\!: \ S_m; \ \ \widehat{\ell_m} :]]\right],$$

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

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

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

Each declaration statement lists several variables that share a common type and identifies their type, i.e., the domain over which the variables range. The optional assertion φ imposes constraints on the initial values of the variables declared in this statement.

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

Figure 3 presents a simple program consisting of two processes communicating by the shared variable x, initially set to 0. Process P_1 keeps incrementing variable y as long as $x = 0$. Process P_2 has only one statement, which sets x to 1. Obviously, once x is set to 1, process P_2 terminates and some time later so does P_1, as soon as it observes that $x \neq 0$.

Let P be an SPL program. To obtain a real-time program, we associate with each executable statement S 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

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

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

Fig. 3. Program ANY-Y: A simple concurrent program

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.

3.2 The CTS *corresponding to an* SPL_T *program*

In the following, we show how an SPL_T program can be represented by a CTS, identifying each of the components of a CTS for a given program.

Consider a program P given by

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

Without loss of generality, we assume that all statements in the program are labeled. Let L_i denote the set of locations within process P_i, $i = 1, \ldots, m$, and let $L_P = L_1 \cup \cdots \cup L_m$ denote the set of locations of the entire program P.

State variables and states. The *state variables* V for system Φ_P consist of the *data variables* $Y = y_1, \ldots, y_n$, that are declared at the head of the program, the set of *control variables* $\pi = \{\pi_1, \ldots, \pi_m\}$, one for each process P_i, and the set of *clocks* $C = \{t_1, \ldots, t_m, T\}$, one clock t_i for each process P_i and a master clock T. The data variables Y range over their respectively declared data domains. The control variable π_i ranges over the location set L_i, for $i = 1, \ldots, m$.

As states we take all possible interpretations that assign to the state variables values over their respective domains.

The initial condition. Let φ denote the *data-precondition* of program P. We define the *initial condition* Θ for Φ_P as

$$\Theta\text{:} \quad \pi_1 = \ell_1, \ldots, \pi_m = \ell_m \wedge\ \varphi\ \wedge\ t_1 = \ldots = t_m = T = 0$$

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

Transition relation. Next, we consider each of the statements that may appear in an $\mathrm{SPL_T}$ program and, for each such statement, we identify its contribution to the transition relation ρ. Typically, each statement may contribute an additional disjunct to ρ.

We proceed to define the contributions of each of the previously introduced statements. In these definitions, we use the notation $pres(U)$ as an abbreviation for

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

stating that all the variables in the variable set $U \subseteq V$ are preserved by a considered statement. The statements we consider are displayed in the form:

$$\ell : S \,;\ \widehat{\ell} \quad \in \quad P_i,$$

implying that the statement S with the pre-condition ℓ and post-condition $\widehat{\ell}$ is a statement in process P_i.

For locations ℓ_j and ℓ_k in a process P_i, we denote

$$\begin{aligned} at\_\ell_j \quad &: \pi_i = \ell_j \\ at\_\ell_{j,k} &: \pi_i = \ell_j \vee \pi_i = \ell_k \,. \end{aligned}$$

– *Assignment*: The statement

$$\ell : y := e \,;\ \widehat{\ell} : \quad \in \quad P_i,$$

contributes the disjunct

$$\rho_\ell: \quad \pi_i = \ell \wedge \pi_i' = \widehat{\ell} \wedge y' = e \wedge t_i \geq l_\ell \wedge t_i^{'} = 0 \wedge pres(V - \{\pi_i, t_i, y\})$$

to the transition relation ρ. The conjunct $t_i \geq l_\ell$ asserts that the transition can be taken only when t_i, the clock corresponding to the process P_i, is not below l_ℓ, the lower bound associated with the transition. When taken, the transition resets clock t_i to 0. The last conjunct of ρ_ℓ asserts that all variables, excluding π_i, t_i and y, retain their values over the transition τ_ℓ.

– *Await*: The statement

$$\ell: \textbf{await } c \,;\ \widehat{\ell} : \quad \in \quad P_i,$$

contributes the disjunct

$$\rho_\ell: \quad \pi_i = \ell \wedge \left(\begin{array}{l} \;\; c \;\wedge\; \pi_i' = \widehat{\ell} \;\wedge\; pres(V - \{\pi_i, t_i\}) \\ \quad \vee \\ \neg c \;\wedge\; pres(V - t_i) \end{array} \right) \wedge t_i \geq l_\ell \wedge t_i^{'} = 0.$$

The *await* transition is enabled once control reaches ℓ and $t_i \geq l_\ell$. Thus, within a time lying between l_ℓ and u_ℓ, it will be taken. When taken, control either moves from ℓ to $\widehat{\ell}$ (if c is true) or remains in place. In any case, the clock associated with this statement will be reset.

– *Noncritical* The statement
ℓ: **noncritical**; $\widehat{\ell}$: $\in$ P_i,
contributes the disjunct

$$\rho_\ell: \quad \pi_i = \ell \;\wedge\; \begin{pmatrix} pres(V - t_i) \\ \vee \\ \pi_i' = \widehat{\ell} \;\wedge\; pres(V - \{\pi_i, t_i\}) \end{pmatrix} \wedge\; t_i \geq l_\ell \;\wedge\; t_i' = 0$$

to the transition relation ρ. This statement makes a non-deterministic choice between staying at the same location or terminating. It is acceptable that the statement consistently chooses not to terminate, representing the behavior of a non-critical section that never terminates. Note that a process can remain forever in its non-critical section from a certain point on.

– *Critical* The statement
ℓ: **critical**; $\widehat{\ell}$: $\in$ P_i,
contributes the disjunct

$$\rho_\ell: \quad \pi_i = \ell \;\wedge\; \pi_i' = \widehat{\ell} \;\wedge\; t_i \geq l_\ell \;\wedge\; t_i' = 0 \wedge\; pres(V - \{\pi_i, t_i\})$$

to the transition relation ρ. The observable action of the *critical* statement is to terminate.

– *Conditional* The statement
ℓ: [**if** c **then** ℓ_1: S_1 **else** ℓ_2: S_2]; $\widehat{\ell}$: $\in$ P_i,
contributes the disjunct

$$\rho_\ell: \pi_i = \ell \;\wedge\; \begin{pmatrix} c \;\wedge\; \pi_i' = \ell_1 \\ \vee \\ \neg c \;\wedge\; \pi_i' = \ell_2\} \end{pmatrix} \wedge\; t_i \geq l_\ell \;\wedge\; t_i'$$
$$= 0 \wedge\; pres(V - \{\pi_i, t_i\}).$$

to the transition relation ρ. Thus, the transition for this statement moves from ℓ to ℓ_1 if the condition c evaluates to T, and moves from ℓ to ℓ_2 if the condition c evaluates to F.

For the one-branch conditional
ℓ: [**if** c **then** ℓ_1: S_1]; $\widehat{\ell}$: $\in$ P_i, we take the transition relation

to be

$$\rho_\ell\colon \pi_i = \ell \wedge \begin{pmatrix} c \ \wedge \ \pi_i' = \ell_1 \\ \vee \\ \neg c \ \wedge \ \pi_i' = \widehat{\ell} \end{pmatrix} \wedge \ t_i \geq l_\ell \ \wedge \ t_i' = 0 \wedge \ pres(V - \{\pi_i, t_i\}).$$

– *While*: The statement

ℓ: **while** c **do** $[\widetilde{\ell}\colon \widetilde{S}]$; $\widehat{\ell}$: $\in$ P_i,

contributes the disjunct

$$\rho_\ell\colon \pi_i = \ell \wedge \begin{pmatrix} c \ \wedge \ \pi_i' = \widetilde{\ell} \\ \vee \\ \neg c \ \wedge \ \pi_i' = \widehat{\ell} \end{pmatrix} \wedge \ t_i \geq l_\ell \ \wedge \ t_i' = 0 \wedge \ pres(V - \{\pi_i, t_i\}).$$

to the transition relation. According to ρ_ℓ, when c evaluates to T control moves from ℓ to $\widetilde{\ell}$, and when c evaluates to F control moves from ℓ to $\widehat{\ell}$. Note that the enabling transition of τ_ℓ is $\pi_i = \ell \wedge \ t_i \geq l_\ell$ which does not depend on the value of c.

The *selection* or the *concatenation* statements make no direct contribution to the transition relation.

Time-progress condition. For each executable statement

$$\ell\colon S$$

in process P_i, Π includes the conjunct

$$\pi_i = \ell \quad \rightarrow \quad t_i < u_S,$$

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.

This concludes the definition of the transition system Φ_P.

Examples of computations. Consider the program ANY-Y presented in Figure 3. 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 $\text{ANY-Y}_{[3,5]}$ is defined as follows:

– *System Variables:* $V = \{\pi_1, \pi_2, x, y, t_1, t_2, T\}$. In addition to the control variables π_1 and π_2, 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:*

$$\Theta: \quad \pi_1 = \ell_0 \ \wedge\ \pi_2 = m_0 \ \wedge\ x = y = 0 \ \wedge\ t_1 = t_2 = T = 0.$$

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

$$\rho_{\ell_0}: \ \pi_1 = \ell_0 \ \wedge \begin{pmatrix} x = 0 \ \wedge\ \pi_1' = \ell_1 \\ \vee \\ x \neq 0 \ \wedge\ \pi_1' = \ell_2 \end{pmatrix} \wedge\ t_1 \geq 3 \ \wedge\ t_1' = 0$$
$$\wedge\, pres(\{\pi_2, x, y, t_2, T\})$$

$$\rho_{\ell_1}: \ \pi_1 = \ell_1 \ \wedge\ \pi_1' = \ell_0 \ \wedge\ y' = y + 1 \ \wedge\ t_1 \geq 3 \ \wedge\ t_1' = 0$$
$$\wedge\ pres(\{\pi_2, x, t_2, T\})$$

$$\rho_{m_0}: \pi_2 = m_0 \ \wedge\ \pi_2' = m_1 \ \wedge\ x' = 1 \ \wedge\ t_2 \geq 3 \ \wedge\ t_2' = 0$$
$$\wedge\ pres(\{\pi_1, y, t_1, T\}).$$

– *Time-progress condition:*

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

The *tick* transition relation for this system is given by

$$\begin{aligned} \rho_{tick}: \quad & \exists \Delta > 0 . pres(\pi_1, \pi_2, x, y) \wedge (t_1', t_2', T') \\ & \quad = (t_1 + \Delta, t_2 + \Delta, T + \Delta) \wedge \\ & \qquad (at\_\ell_{0,1} \ \rightarrow\ t_1 + \Delta \leq 5) \wedge (at\_m_0 \ \rightarrow\ t_2 + \Delta \leq 5)\,. \end{aligned}$$

Having defined the CTS Φ_P 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.

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)

4 Verifying safety properties of clocked transition systems

In this section, we present methods for verifying safety properties of clocked transitions systems. Safety properties are those that can be expressed by a formula generated from state formulas, the boolean operators $\vee$ and $\wedge$, and the temporal operators $\Box$ and $\mathcal{W}$. we refer to a formula of this form as a *cannonical safety formula*.

In this paper, we consider only invariance and *wait-for* properties specified over state-formulas.

4.1 The invariance rule

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

The accessibility rule. As a preliminary step, we introduce a rule that establishes the Φ-state 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 $\tau \in \mathcal{T}_T$ 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 $\Box p$.

For assertions φ and p,

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

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

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 2. 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$_{[3,5]}$ (Fig. 3). This assertion claims that, at every state in the execution of program ANY-Y$_{[3,5]}$, 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

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

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

$$\underbrace{at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x = 0 \;\wedge\; \cdots}_{\Theta} \quad \rightarrow \quad \underbrace{(x = 0 \wedge at\_\ell_{0,1}) \vee \cdots}_{\varphi}$$

which is obviously valid.

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

$$\left(\begin{array}{l} \underbrace{\cdots \wedge \begin{pmatrix} x = 0 \ \wedge \ at'\_\ell_1 \\ \vee \\ x \neq 0 \ \wedge \ at'\_\ell_2 \end{pmatrix} \wedge \ x' = x \ \wedge \ \cdots}_{\rho_{\ell_0}} \\ \\ \wedge \ \underbrace{(x = 0 \wedge at\_\ell_{0,1}) \ \vee \ at\_m_1}_{\varphi} \end{array}\right) \quad \rightarrow$$

$$\underbrace{(x' = 0 \wedge at'\_\ell_{0,1}) \ \vee \ at'\_m_1}_{\varphi'}.$$

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

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

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

4.2 Verification diagrams

In proofs of properties of clocked and hybrid systems, it is typically necessary to deal with several assertions at the same time and trace which transitions lead from one assertion to another. These proofs can be effectively presented by the graphical formalism of *verification diagrams*. Verification diagrams have been introduced as a visualization tool in the deductive proofs of untimed, reactive systems (see [MP94], [BMS95]). In this paper we use the formalism with minor changes, adapting it to the proof rules of clocked and hybrid systems.

In the following we define the *basic verification diagram*, a diagram whose properties are common to all verification diagrams presented in this paper.

A basic verification diagram is a directed labeled graph constructed as follows:

– *Nodes* in the graph are labeled by assertions $(\varphi_0), \varphi_1, \ldots, \varphi_m$. 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 may be designated as *initial nodes*. They are annotated by an entry arrow .
- Optionally, one of the nodes is designated as a *terminal node*, ("goal" node). In the graphical representation, this node is distinguished by having a boldface boundary, and is labeled by the assertion φ_0. No edges depart from a terminal node.

Verification conditions. With each verification diagram, we associate a set of verification conditions, each corresponding to some premise of the proof rule represented by the diagram. To facilitate the expression of verification conditions, we introduce the abbreviation

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

for assertions p and q and transition τ.

We say that a verification diagram is *valid over a* CTS Φ (Φ-valid) if all the verification conditions associated with the diagram are Φ-state valid.

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.

4.3 Invariance verification diagrams

An invariance diagram is a basic verification diagram with no terminal node. For example, in Fig. 6 we present a verification diagram for program $\text{ANY-Y}_{[3,5]}$.

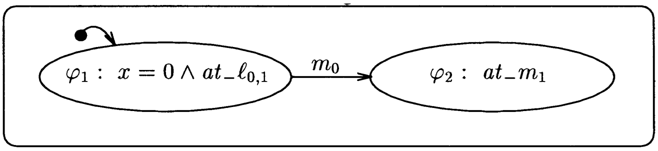

Fig. 6. Verification Diagram D_1

Verification conditions for invariance diagrams. With each invariance diagram, we associate the following *verification conditions*:

– 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 verification condition associated with φ and τ (corresponding to premise I3) is given by
$$\{\varphi\}\, \tau \,\{\varphi \vee \varphi_1 \vee \ldots \vee \varphi_k\}\,.$$
In particular, if $k = 0$ (i.e., φ has no τ-successors), the associated verification condition is
$$\{\varphi\}\ \tau\ \{\varphi\}.$$

For example, the verification conditions associated with diagram D_1 (Fig. 6), are:

$$\begin{array}{ll}
\text{I3 for } \varphi_1 \text{ and } \ell_0 : & \{x = 0 \wedge at_\ell_{0,1}\}\, \ell_0 \,\{x = 0 \wedge at_\ell_{0,1}\} \\
\text{I3 for } \varphi_1 \text{ and } \ell_1 : & \{x = 0 \wedge at_\ell_{0,1}\}\, \ell_1 \,\{x = 0 \wedge at_\ell_{0,1}\} \\
\text{I3 for } \varphi_1 \text{ and } m_0 : & \{x = 0 \wedge at_\ell_{0,1}\}\, m_0 \,\{(x = 0 \wedge at_\ell_{0,1}) \vee at_m_1\} \\
\text{I3 for } \varphi_1 \text{ and } \textit{tick} : & \{x = 0 \wedge at_\ell_{0,1}\}\, \textit{tick} \,\{x = 0 \wedge at_\ell_{0,1}\}
\end{array}$$

$$\begin{array}{ll}
\text{I3 for } \varphi_2 \text{ and } \ell_0 : & \{at_m_1\}\, \ell_0 \,\{at_m_1\} \\
\text{I3 for } \varphi_2 \text{ and } \ell_1 : & \{at_m_1\}\, \ell_1 \,\{at_m_1\} \\
\text{I3 for } \varphi_2 \text{ and } m_0 : & \{at_m_1\}\, m_0 \,\{at_m_1\} \\
\text{I3 for } \varphi_2 \text{ and } \textit{tick} : & \{at_m_1\}\, \textit{tick} \,\{at_m_1\}\,.
\end{array}$$

Let D_P be an invariance diagram associated with (the CTS of) a program P. Let $\varphi_1, \ldots, \varphi_m$ be the assertions labeling the nodes of D_P.

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

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

If, in addition, $\Theta \to \bigvee_{i=1}^{m} \varphi_i$ *and* $\varphi_i \to p$ *for every* $i = 1, \ldots, m$*, then*

$$P \models \Box p.$$

In case there is a subset $N \subseteq \{1, \ldots, m\}$ *such that* $\Theta \to \bigvee_{i \in N} \varphi_i$*, we identify* φ_i*,* $i \in N$ *as initial nodes.*

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

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

Using the encapsulation conventions, we can draw a more detailed invariance diagram establishing (1), as shown in Fig. 7.

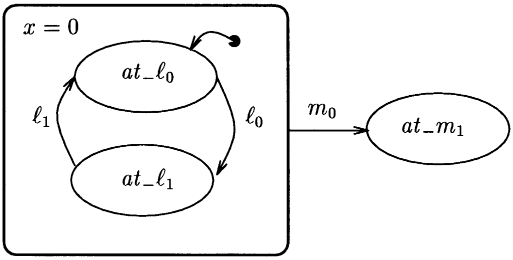

Fig. 7. A more detailed diagram, using encapsulation conventions

In the diagram of Fig. 7, the single assertion $x = 0 \land at\_\ell_{0,1}$, has been broken into the two sub-cases: $x = 0 \land at\_\ell_0$ and $x = 0 \land at\_\ell_1$, explicitly displaying the fact that transitions ℓ_0 and ℓ_1 cause the system to move between these two sub-cases.

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

$$\Box(T \leq 15 \lor (at\_\ell_2 \land at\_m_1))\,.$$

This formula claims that every accessible state in which the program has not terminated yet (i.e., $at\_\ell_2 \land at\_m_1$ 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 Sect. 5.

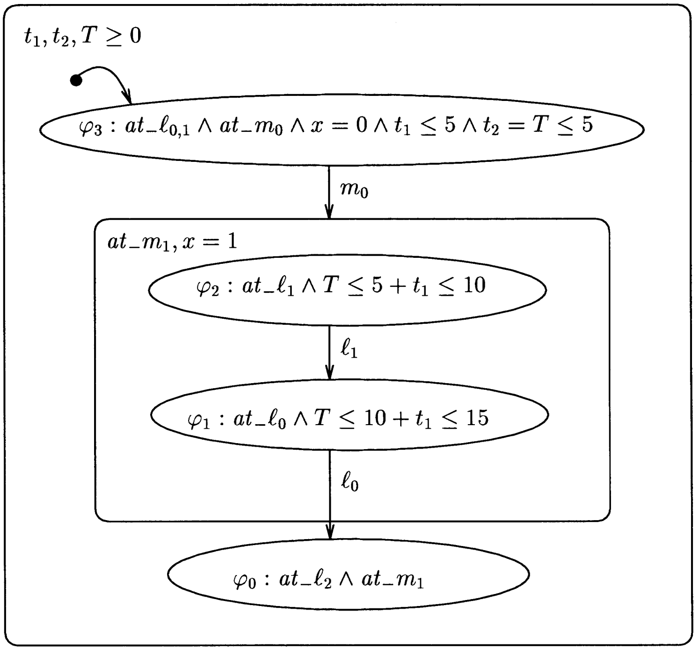

Fig. 8. Termination of ANY-Y within 15 time units

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 φ_2: $at\_\ell_1 \wedge T \leq 5 + t_1 \leq 10$ is preserved by the *tick* transition.

$$\underbrace{at\_\ell_1 \wedge T \leq 5 + t_1 \leq 10}_{\varphi_2} \quad \wedge$$
$$\underbrace{\Delta > 0 \wedge \pi' = \pi \wedge T' = T + \Delta \wedge t_1' = t_1 + \Delta \wedge (at\_\ell_1 \rightarrow t_1 + \Delta \leq 5) \wedge \cdots}_{\rho_{tick}}$$
$$\rightarrow \quad \underbrace{at'\_\ell_1 \wedge T' \leq 5 + t_1' \leq 10}_{\varphi_2'} .$$

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

Example 4. 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], [AL91], 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

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

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

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

$$\left[\begin{array}{l}
\ell_0 : \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1 : \textbf{noncritical} \\
\ell_2 : \textbf{skip} \\
\ell_3 : \textbf{while } x \neq 1 \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_4 : \textbf{await } x = 0 \\
\ell_5 : x := 1 \\
\ell_6 : \textbf{skip} \\
\ell_7 : \textbf{if } x = 1 \textbf{ then} \\
\qquad \ell_8 : \textbf{critical}
\end{array}\right] \\
\ell_9 : \textbf{skip} \\
\ell_{10} : x := 0
\end{array}\right]
\end{array}\right]
\;\Bigg\|\;
\left[\begin{array}{l}
m_0 : \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
m_1 : \textbf{noncritical} \\
m_2 : \textbf{skip} \\
m_3 : \textbf{while } x \neq 2 \textbf{ do} \\
\quad \left[\begin{array}{l}
m_4 : \textbf{await } x = 0 \\
m_5 : x := 2 \\
m_6 : \textbf{skip} \\
m_7 : \textbf{if } x = 2 \textbf{ then} \\
\qquad m_8 : \textbf{critical}
\end{array}\right] \\
m_9 : \textbf{skip} \\
m_{10} : x := 0
\end{array}\right]
\end{array}\right]$$

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

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 entered 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 keeps 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 intentions to enter the critical sections.

A formal proof that the assertion $\neg(at\_\ell_8 \ \wedge \ at\_m_8)$ is an invariant of program MUTEX is presented by the invariance diagram of Fig. 10.

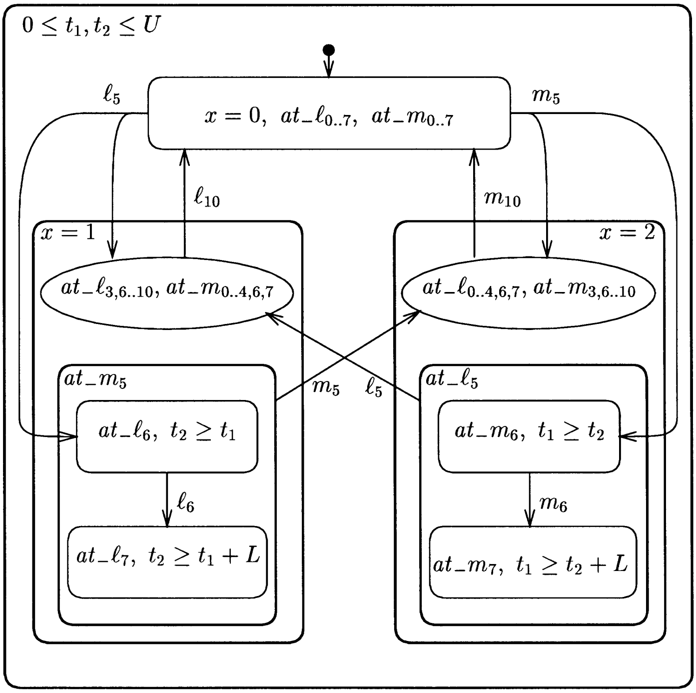

Fig. 10. Mutual exclusion for program MUTEX

All verification conditions associated with this diagram, have been verified automatically (with no user intervention) by the STeP verifier [BBC+95], using the axiom $2L > U$.

This concludes the proof of mutual exclusion for program MUTEX.

4.4 Completeness of rule INV

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

Claim 2 *If formula $\Box p$ is valid over non-zeno* CTS *Φ, 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_Φ that holds in a state s iff s is Φ-accessible, i.e., appears in some run of Φ. We then show that if $\Box p$ is Φ-valid, then the premises of rule INV are state-valid (implying that they are also Φ-state valid) when taking acc_Φ 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_Φ, we refer the reader to [MP91] or Chapter 2 of [MP95]. Assume that we have constructed an assertion acc_Φ such that

$$s \models acc_\Phi \qquad \text{iff} \qquad s \text{ is a } \Phi\text{-accessible state.}$$

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

I1. $acc_\Phi \to p$
By our assumption that $\Box p$ is Φ-valid, it follows that each state appearing in some computation satisfies p. As Φ is non-zeno, every accessible state appears in some computation and, hence, every accessible state satisfies p. Since acc_Φ characterizes precisely the accessible states, the premise follows.

I2. $\Theta \to acc_\Phi$
It is obvious that every state satisfying Θ is initial and is, therefore, Φ-accessible. Consequently, such a state must satisfy acc_Φ, which characterizes all Φ-accessible states.

I3. $\rho_\tau(U, \widetilde{U}) \wedge acc_\Phi(U) \quad \to \quad acc_\Phi(\widetilde{U})$, 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 $acc_\Phi(U)$ are true. We will show that $acc_\Phi(\widetilde{U})$ is also true.
Let s and $\widetilde{s}$ be two arbitrary states such that $s[V] = U$ and $\widetilde{s}[V] = \widetilde{U}$. Since $acc_\Phi(U) = \mathrm{T}$, we have that $s \models acc_\Phi$. By the meaning of acc_Φ, it follows that s is Φ-accessible. From $\rho_\tau(U, \widetilde{U}) = \mathrm{T}$, 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 $\widetilde{s} \models acc_\Phi$, leading to $acc_\Phi(\widetilde{U}) = \mathrm{T}$.

This concludes the proof of completeness of rule INV.

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

Example 5. Consider the CTS Φ_3 which is presented graphically in Fig. 11 and textually in Fig. 12.

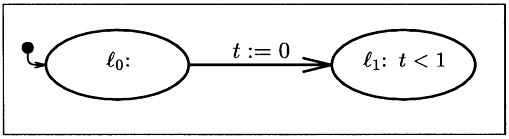

Fig. 11. CTS Φ_3

V: $\{\pi: \{0,1\}; t, T: \textbf{real}\}$

Θ: $\pi = t = T = 0$

$\mathcal{T}$: $\{\tau_{01}\}$ with transition relation

τ_{01}: $\pi = 0 \wedge \pi' = 1 \wedge t' = 0 \wedge T' = T$

Π: $\pi = 1 \rightarrow t < 1$

Fig. 12. CTS Φ_3 in textual form

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 $\Box at\_\ell_0$ is Φ_3-valid. However, examination of the arguments for the soundness of rule INV shows that any assertion p, whose invariance is proven by rule INV, must hold on all accessible states. The state $\langle \pi: 1 \rangle$ is accessible for CTS Φ_3 but does not satisfy at_ℓ_0. Therefore, the Φ_3-validity of $\Box at\_\ell_0$ cannot be proven by rule INV.

4.5 Verifying waiting-for formulas

The invariance formula $\Box q$ 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 6. Consider program CYCLIC, presented in Fig. 13. This program 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.

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

$$\left[\begin{array}{l} \ell_0 : \ \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1 : \textbf{while } x = 0 \textbf{ do} \\ \qquad \ell_2 : \ y := y + 1 \\ \ell_3 : x := 0 \end{array}\right] \end{array}\right] \quad \| \quad \left[\begin{array}{l} m_0 : \ \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1 : x := 1 \\ m_2 : \textbf{await } x = 0 \end{array}\right] \end{array}\right]$$

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

Fig. 13. Program CYCLIC

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

$$T_a \quad = \quad T - a.$$

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.

Waiting diagrams. Proofs by rule WAIT can be succinctly represented by the basic verification diagrams. We refer to basic verification diagrams which are used to represent waiting-for proofs as *waiting diagrams*.

For assertions p, q, φ, and r,

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

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

Verification conditions implied by a waiting diagram. Consider a non-terminal 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:

$$\{\varphi\}\;\tau\;\{\varphi \;\vee\; \varphi_1 \;\vee\; \cdots \;\vee\; \varphi_k\}.$$

Similar to invariance verification diagrams, if $k = 0$ (i.e., φ has no τ-successors), the associated verification condition is

$$\{\varphi\}\;\tau\;\{\varphi\}.$$

Valid waiting diagrams. The consequences of having a valid waiting 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*

$$P \models \bigvee_{j=0}^{m} \varphi_j \;\;\Rightarrow\;\; \left(\bigvee_{j=1}^{m} \varphi_j\right) \mathcal{W} \varphi_0 \,.$$

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

$$\text{U1:} \quad \bigvee_{j=1}^{m} \varphi_j \;\;\rightarrow\;\; q, \qquad \textit{and} \qquad \text{U2:} \quad p \;\;\rightarrow\;\; \bigvee_{j=0}^{m} \varphi_j,$$

then we can conclude:

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

In case there is a subset $N \subseteq \{1, \ldots, m\}$ such that $p \rightarrow \bigvee_{i \in N} \varphi_i$, we identify φ_i, $i \in N$ as initial nodes.

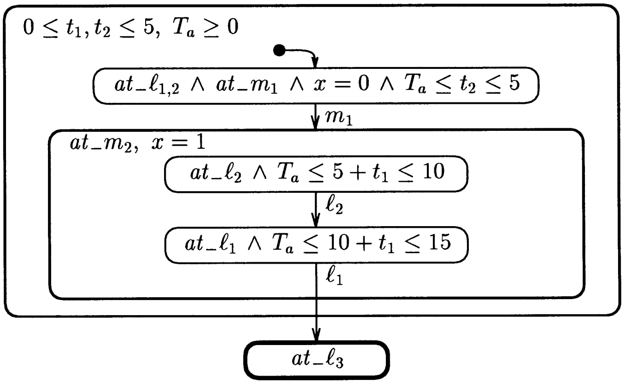

Fig. 15. A waiting diagram, establishing the formula
$at\_\ell_1 \wedge at\_m_1 \wedge x = 0 \wedge T_a = 0 \quad \Rightarrow \quad (T_a \le 15)\mathcal{W} at\_\ell_3$

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

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

for program $\text{CYCLIC}_{[3,5]}$.

5 Verifying response properties of clocked transition systems

In this section, we present methods for verifying response properties of clocked transition systems. We begin with the clock-bounded chain rule, which is an adaptation of the chain rule developed for reactive systems, to real-time systems. This rule is used to prove response properties which require a bounded number of (non-tick) steps for their achievement. Next, we present the clock-bounded well-founded rule, which is the real time extension to the WELL rule used for reactive systems. Similar to the WELL rule, the clock-bounded well-founded rule is used to prove response properties which require an unbounded number of steps. To deal with these cases, we

For assertions p, q, and $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$,
clocks $t_1, \ldots, t_m \in C$, and
real constants $b_1, \ldots, b_m \in \mathsf{R}$,

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

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

$$\text{C2.} \quad \rho_\tau \wedge \varphi_i \;\rightarrow\; (\varphi_i' \wedge t_i' \geq t_i) \vee \bigvee_{j<i} \varphi_j' \qquad \text{for every } \tau \in \mathcal{T}_T$$

$$\text{C3.} \quad \varphi_i \;\rightarrow\; t_i \leq b_i$$

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

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

must generalize the induction over a fixed finite subrange of the integers into an explicit induction over an arbitrary well-founded relation.

The changes made to both the CHAIN and the WELL rules developed for untimed systems, reflect the fact that progress in the real time systems is ensured by the progress of time and not by fairness.

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 $t_i \in C$, to which we refer as the *helpful clock*, and a real-valued upper bound b_i. The intention is that while remaining in states satisfying φ_i, the clock t_i is bounded by b_i and never reset. Since time in a computation grows beyond any bound, this will imply that we cannot continually stay at a φ_i-state for too long.

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

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 t_i.

Premise C3 requires that assertion φ_i implies that t_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

$$p \Rightarrow \Diamond q,$$

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 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 t_u at states s_i, $i \geq r$. By C2, the value of t_u never decreases. Also, whenever a *tick* transition with increment Δ is taken, t_u increases (as do all clocks) by Δ. It follows that the master clock T cannot increase by more than $b_u - s_r[t_u]$ 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.

Note that the premises of rule CB-CHAIN ensure the stronger time-bounded response property

$$p \wedge T_a = 0 \Rightarrow \Diamond \left(q \wedge T_a \leq \sum_{i=1}^{m} b_i \right).$$

Example 8. 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

$$at\_\ell_0 \wedge at\_m_0 \wedge x = t_1 = t_2 = T = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1).$$

To apply rule CB-CHAIN, we identify $at\_\ell_0 \wedge at\_m_0 \wedge x = t_1 = t_2 = T = 0$ as p and $at\_\ell_2 \wedge at\_m_1$ as q. The auxiliary assertions, helpful clocks, and bounds are presented in the following table:

φ_0: $at\_\ell_2 \wedge at\_m_1$	–	–
φ_1: $at\_\ell_0 \wedge at\_m_1 \wedge x = 1 \wedge t_1 \leq 5$	t_1: t_1	b_1: 5
φ_2: $at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \wedge t_1 \leq 5$	t_2: t_1	b_2: 5
φ_3: $at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0 \wedge t_1 \leq 5 \wedge t_2 \leq 5$	t_3: t_2	b_3: 5.

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

– Premise C1 assumes the form

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

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 a few representative cases.
Premise C2 for φ_3 and transition m_0 assumes the form

$$\underbrace{\pi_2' = m_1 \wedge x' = 1 \wedge t_1' = t_1 \wedge \cdots}_{\rho_{m_0}} \wedge \underbrace{at\_\ell_{0,1} \wedge t_1 \leq 5 \wedge \cdots}_{\varphi_3}$$
$$\rightarrow \left(\begin{array}{ll} & \cdots \\ \vee & \underbrace{at'\_\ell_0 \wedge at'\_m_1 \wedge x' = 1 \wedge t_1' \leq 5}_{\varphi_1'} \\ \vee & \underbrace{at'\_\ell_1 \wedge at'\_m_1 \wedge x' = 1 \wedge t_1' \leq 5}_{\varphi_2'} \end{array}\right),$$

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

$$\underbrace{\Delta > 0 \wedge pres(\pi_1, \pi_2, x) \wedge t_1' = t_1 + \Delta \wedge (at\_\ell_{0,1} \rightarrow t_1 + \Delta \leq 5) \wedge \cdots}_{\rho_{tick}}$$
$$\wedge \underbrace{at\_\ell_1 \wedge at\_m_1 \wedge x = 1 \wedge t_1 \leq 5}_{\varphi_2} \rightarrow$$
$$\underbrace{at'\_\ell_1 \wedge at'\_m_1 \wedge x' = 1 \wedge t_1' \leq 5}_{\varphi_2'} \wedge \underbrace{t_1 \leq t_1'}_{t_2 \leq t_2'},$$

which is obviously state valid.

– Premise C3 is trivially valid, since each φ_i, $i = 1, \ldots, 3$ includes $t_i \leq b_i$ as a conjunct.

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

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

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 summarize 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 basic verification diagram satisfying:

– The terminal node is labeled by an assertion φ_0. All other nodes are labeled by a pair of assertions: ϕ_i and β_i, for $i = 1, \ldots, m$. The assertion β_i has the form $t_i \leq b_i$, where $t_i \in C$ 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. For uniformity, we define $\phi_0 = \varphi_0$.
– 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: $t \leq b$. 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:

$$\rho_\tau \ \wedge \ \varphi \ \rightarrow \ (\varphi' \ \wedge \ t' \geq t) \ \vee \ \varphi_1' \ \vee \ \cdots \ \vee \ \varphi_k'.$$

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

$$\rho_\tau \ \wedge \ \varphi \ \rightarrow \ \varphi' \ \wedge \ t' \geq t.$$

Valid chain diagrams. The consequences of having a valid clock-bounded chain diagram are stated in the following claim:

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

$$P \quad \models \quad \left(\bigvee_{j=0}^{m} \varphi_j\right) \Rightarrow \Diamond\varphi_0\,.$$

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

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

then we can conclude:

$$P \quad \models \quad p \Rightarrow \Diamond q.$$

In case there is a subset $N \subseteq \{1, \ldots, m\}$ such that $p \rightarrow \bigvee_{i \in N} \varphi_i$, we identify φ_i, $i \in N$ as initial nodes.

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 $\beta_i\colon\ t_i \leq b_i$ as an explicit conjunct of $\varphi_i = \phi_i \wedge \beta_i$.

Example 9. In Fig. 17, we present a chain diagram for proving that

$$at\_\ell_0 \wedge at\_m_0 \wedge x = t_1 = t_2 = T = 0 \quad \Rightarrow \quad \Diamond(at\_\ell_2 \wedge at\_m_1)$$

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

Note the use of encapsulation in labeling the compound node by the common clock bound $\beta\colon\ t_1 \leq 5$, 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 more economic and comprehensible verification diagrams. Similar to the previously introduced encapsulation conventions, this one is also inspired by the Statechart language [Har87].

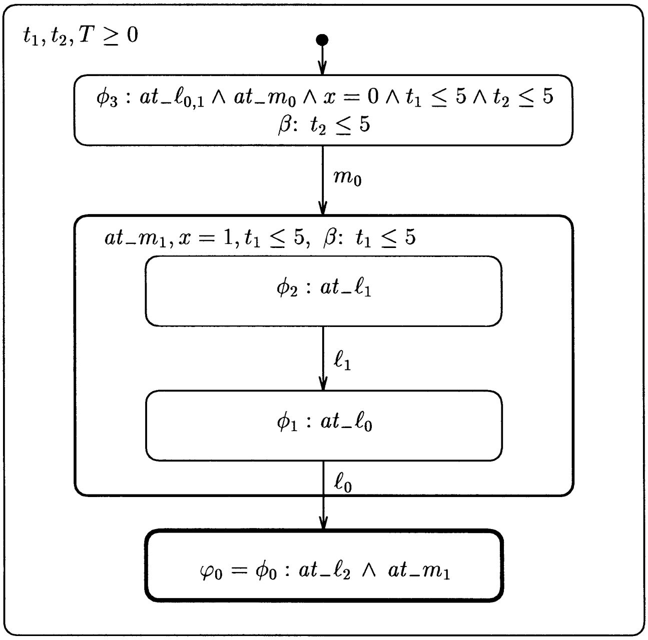

Fig. 17. Verifying termination of program ANY-Y$_{[3,5]}$

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.

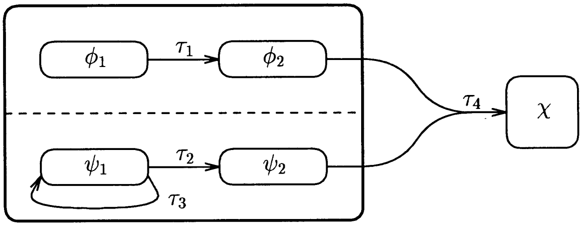

Fig. 18. A conjunctive compound node

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 $j \in \{1, \ldots, n\}$, 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 $j \in \{1, \ldots, n\}$.
- 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.

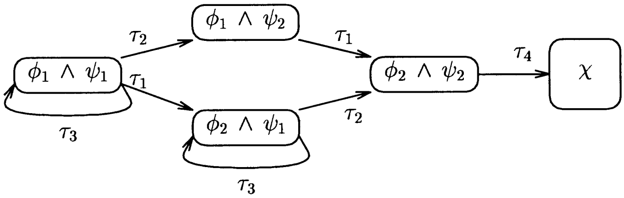

Fig. 19. An expanded equivalent to the conjunctive diagram

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 program is run, process P_1 will terminate before P_2 does. This is because P_1 must 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.

A central argument in the validation of this diagram is that transition m_3 is disabled on all nodes within the conjunctive compound node. Transition m_3 can be enabled only on an at_m_3-state and only when $t_2 \geq L$. Combining

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

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

Fig. 20. Program RACE

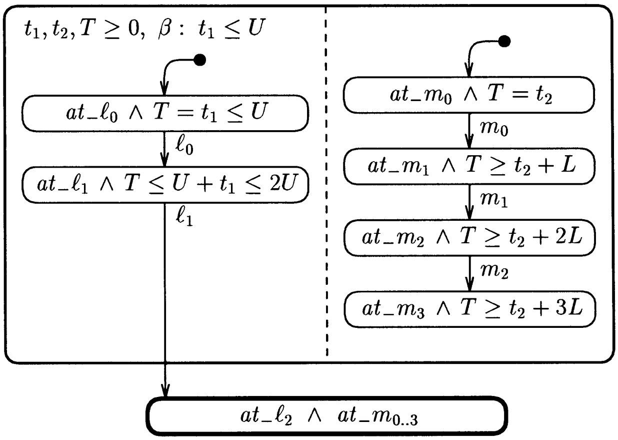

Fig. 21. Chain diagram proving that P_1 wins the race

the assertion attached to the at_m_3-node with $t_2 \geq L$, we obtain $T \geq 4L > 2U$. However, all assertions on the left-hand side of the conjunctive node imply $T \leq 2U$. This shows that m_3 is disabled on all states covered by the conjunctive node, and the only exit is via ℓ_1.

This establishes the property

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

5.4 Proving accessibility for program MUTEX$_{[L,U]}$

As a more ambitious example, we prove for program MUTEX$_{[L,U]}$ (Fig. 9) the property of accessibility which can be stated (for process P_1) by the response formula

$$at\_\ell_2 \quad \Rightarrow \quad \Diamond at\_\ell_8.$$

A similar formula states accessibility for process P_2.

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

$$
\begin{array}{ll}
\chi_0: & \Box(x \in \{0,1,2\}) \\
\chi_1: & at\_\ell_{0..2,4,5} \quad \Rightarrow \quad x \neq 1 \\
\chi_2: & at\_m_{0..2,4,5} \quad \Rightarrow \quad x \neq 2 \\
\chi_3: & at\_\ell_4 \quad \Rightarrow \quad x = 0 \ \lor \ (at\_m_{6..10} \ \land \ x = 2) \\
\chi_4: & at\_\ell_{6,7} \ \land \ x = 1 \quad \Rightarrow \quad at\_m_{0..7} \, .
\end{array}
$$

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

$$
\begin{array}{ll}
\psi_1: & at\_\ell_2 \quad \Rightarrow \quad \Diamond at\_\ell_4 \\
\psi_2: & at\_\ell_4 \ \land \ x = 0 \quad \Rightarrow \\
 & \qquad \Diamond(at\_\ell_8 \ \lor \ (at\_\ell_{4,6,7} \ \land \ at\_m_6 \ \land \ x = 2 \ \land \ t_2 = 0)) \\
\psi_3: & at\_\ell_{4,6,7} \ \land \ at\_m_6 \ \land \ x = 2 \ \land \ t_2 = 0 \quad \Rightarrow \\
 & \qquad \Diamond(at\_\ell_4 \ \land \ at\_m_0 \ \land \ x = 0 \ \land \ t_2 = 0) \\
\psi_4: & at\_\ell_4 \ \land \ at\_m_{6..10} \ \land \ x = 2 \quad \Rightarrow \\
 & \qquad \Diamond(at\_\ell_4 \ \land \ at\_m_0 \ \land \ x = 0 \ \land \ t_2 = 0) \\
\psi_5: & at\_\ell_4 \ \land \ at\_m_0 \ \land \ x = 0 \ \land \ t_2 = 0 \quad \Rightarrow \\
 & \qquad \Diamond(at\_\ell_6 \ \land \ at\_m_{0..4} \ \land \ x = 1) \\
\psi_6: & at\_\ell_6 \ \land \ at\_m_{0..4} \ \land \ x = 1 \quad \Rightarrow \quad \Diamond at\_\ell_8 \, .
\end{array}
$$

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 ψ_1: $\quad at\_\ell_2 \Rightarrow \Diamond at\_\ell_4$. 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.

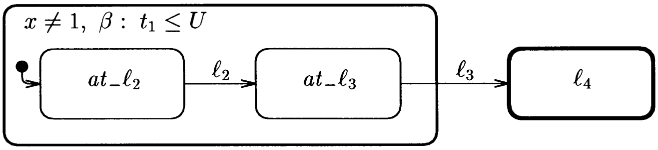

Fig. 22. Chain diagram for ψ_1: $at\_\ell_2 \Rightarrow \Diamond at\_\ell_4$

Proving ψ_2. Response formula ψ_2 is given by

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

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.

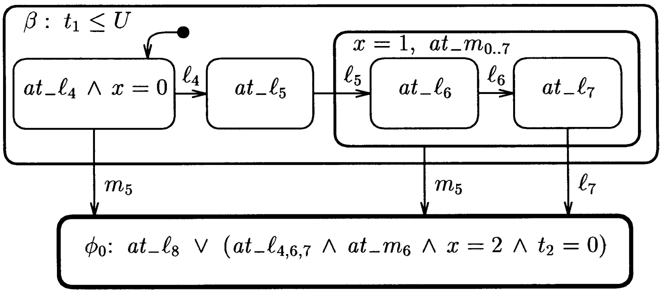

Fig. 23. Chain diagram for the formula
ψ_2: $at\_\ell_4 \wedge x = 0 \ \Rightarrow\ \Diamond\Big(at\_\ell_8 \vee (at\_\ell_{4,6,7} \wedge at\_m_6 \wedge x = 2 \wedge t_2 = 0)\Big)$

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. Here, we use invariant χ_4 to infer that when P_1 moves from ℓ_5 to ℓ_6, setting x to 1, process P_2 can only be at locations $m_0, \ldots, m_7$. From this point on, either P_2 will perform m_5, leading again to ϕ_0, or P_1 will perform ℓ_7 moving to ℓ_8, which is also a goal state.

Proving ψ_3. Response formula ψ_3 is given by

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

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 ℓ_4. This results from the number of statements that each process has to execute until it reaches 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

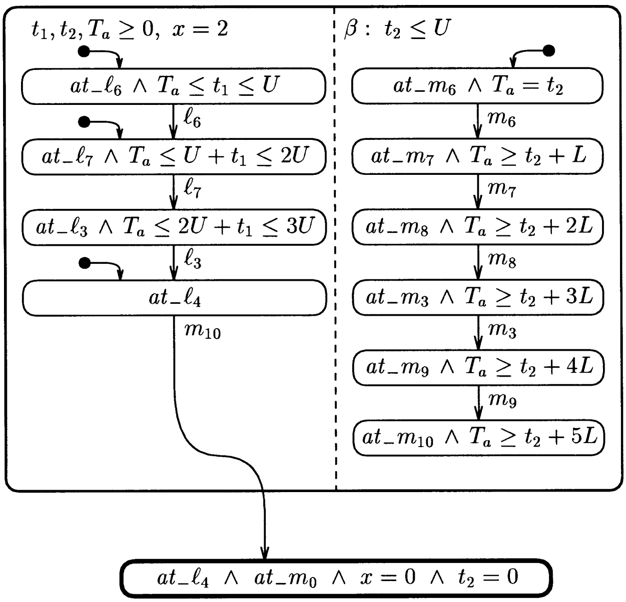

Fig. 24. Chain diagram for the formula

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

Note that m_{10} is enabled only at states in which P_1 is at ℓ_4. This is because 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, being 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 conjunctive compound node.

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 interfere 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

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

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

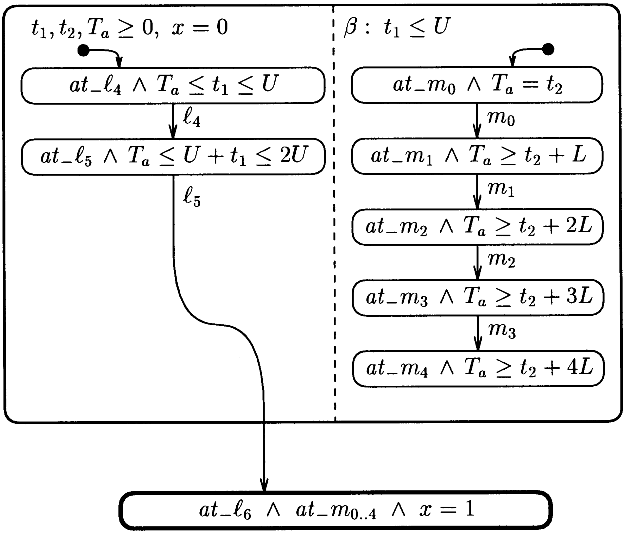

Fig. 25. Chain diagram for the formula
ψ_5: $at\_\ell_4 \wedge at\_m_0 \wedge x = 0 \wedge t_2 = 0 \wedge T_a = 0 \;\Rightarrow\; \diamondsuit(at\_\ell_6 \wedge at\_m_{0..4} \wedge x = 1)$

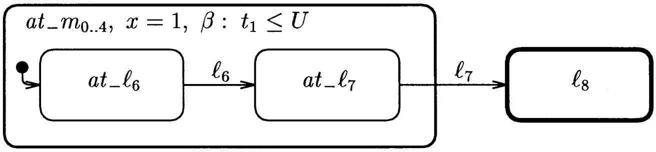

Fig. 26. Chain diagram for the formula
ψ_6: $at\_\ell_6 \wedge at\_m_{0..4} \wedge x = 1 \quad\Rightarrow\quad \diamondsuit at\_\ell_8$

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 steps 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 $a \in \mathcal{A}$.

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 clocked 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 $t_i \in C$, to which we refer as the *helpful clock*, and an upper bound B_i, which is a

For assertions p, q, and $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$,
clocks $t_1, \ldots, t_m \in C$,
real expressions $B_1, \ldots, B_m \in \mathsf{R}$,
a well-founded domain $(\mathcal{A}, \succ)$, and
ranking functions $\delta_1, \ldots, \delta_m$: $\Sigma \mapsto \mathcal{A}$,

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

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

$$\text{W2.} \quad \rho_\tau \;\wedge\; \varphi_i \;\;\rightarrow\;\; \bigvee_{j=0}^{m} (\varphi'_j \;\wedge\; \delta_i \succ \delta'_j) \;\vee$$

$$(\varphi'_i \;\wedge\; \delta'_i = \delta_i \;\wedge\; t_i \leq t'_i \;\wedge\; B'_i \leq B_i)$$

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

$$\text{W3.} \quad \varphi_i \;\;\rightarrow\;\; t_i \leq B_i$$

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

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

real-valued expression. Note that allowing B_i to be an expression, is a generalization over the rule CB-CHAIN where the upper time bounds are constants. This generalization is necessary in order to guarantee the completeness of the rule (claim 7).

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.

Premise W1 requires that every p-position satisfies one of $\varphi_0 = q, \varphi_1, \ldots,$ φ_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, 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 t_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 t_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 p 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 j_k be the smallest index such that φ_{j_k} holds at s_k and $u_k = \delta_{j_k}(s_k)$.

By W2, state s_{k+1} must satisfy φ_j for some $j > 0$, implying that s_{k+1} has a defined rank u_{k+1}. Premise W2 requires that $u_{k+1} \preceq u_k$.

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 t_{j_r} at states s_i, $i \geq r$. Since the rank never decreases beyond r, the value of t_{j_r} never decreases and the value of B_{j_r} never increases beyond that position. Also, whenever a *tick* transition with increment Δ is taken, t_{j_r} increases (as do all clocks) by Δ. It follows that the master clock T cannot increase by more than $B_{j_r}(s_r) - s_r[t_{j_r}]$ 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.

Claim 7 *Rule* CB-WELL *is complete for proving that a response formula is valid over a non-zeno system Φ.*

Justification. (A sketch). The meaning of this claim is that if the response formula $p \Rightarrow \Diamond q$ is valid over the non-zeno system Φ, then there exist constructs as required by rule CB-WELL, such that all premises of the rule are Φ-state valid.

An execution segment σ is called *q-free* if no state in σ satisfies q. A state s' is said to be a *$\neg q$-follower* of state s if there is a q-free Φ-execution segment leading from s to s'. We follow the techniques of [MP91] and take for (a single) φ the assertion $pending_q$, constructed in such a way that

$$s \Vdash pending_q \qquad \text{iff} \qquad s \text{ is a } \neg q\text{-follower of a } \Phi\text{-accessible } p\text{-state.}$$

We define a binary relation $\sqsupset$ such that $s \sqsupset s'$ if s satisfies $pending_q$, s' is a $\neg q$-follower of s, and $s'[T] \geq s[T] + 1$. Obviously, $\sqsupset$ is well-founded, because an infinite sequence $s_0 \sqsupset s_1 \sqsupset s_2 \sqsupset \cdots$ would lead to a computation violating $p \Rightarrow \diamond q$.

Based on a transcendentally inductive construction, we can define a ranking function $\delta\colon \Sigma \mapsto \mathcal{O}rd$, mapping states into the ordinals, such that

O1. If s' is a $\neg q$-follower of the $pending_q$-state s, then $\delta(s) \geq \delta(s')$.
O2. If $s \sqsupset s'$, where s is a $pending_q$-state, then $\delta(s) > \delta(s')$.

Given a pending state s, let $B(s)$ denote the supremum of all values $s'[T]$ where s' is a $\neg q$-follower of s and $\delta(s') = \delta(s)$. Due to property O2, this supremum exists and is bounded by $s[T] + 1$. It can now be shown that all premises of rule CB-WELL hold for the choice of $m = 1$, $\varphi_1 = pending_q$, $t_1 = T$, $B_1 = B(s)$, $(\mathcal{A}, \succ) = (\mathcal{O}rd, >)$, and $\delta_1 = \delta$ as defined above.

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

Example 10. Consider program UP-DOWN presented in Fig. 28.

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

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

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

Fig. 28. Program UP-DOWN

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

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

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: $at\_\ell_3 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ y > 0 \ \wedge \ t_1 \leq U$	t_1: t_1	δ_1: $(1, 2y)$
φ_2: $at\_\ell_2 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ y \geq 0 \ \wedge \ t_1 \leq U$	t_2: t_1	δ_2: $(1, 2y+1)$
φ_3: $at\_\ell_0 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ y \geq 0 \ \wedge \ t_1 \leq U$	t_3: t_1	δ_3: 2
φ_4: $at\_\ell_1 \ \wedge \ at\_m_1 \ \wedge \ x = 1 \ \wedge \ y \geq 0 \ \wedge \ t_1 \leq U$	t_4: t_1	δ_4: 3
φ_5: $at\_\ell_{0,1} \ \wedge \ at\_m_0 \ \wedge \ x = 0 \ \wedge \ y \geq 0 \ \wedge \ t_1 \leq U \ \wedge \ t_2 \leq U$	t_5: t_2	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

$$\underbrace{\begin{pmatrix} y > 0 \ \wedge \ move(\ell_2, \ell_3) \\ \vee \\ y \leq 0 \ \wedge \ move(\ell_2, \ell_4) \end{pmatrix} \wedge \ pres(V - \pi_1)}_{\rho_{\ell_2}} \ \wedge \ \underbrace{at\_m_1 \ \wedge \ x = 1 \ \wedge \ y \geq 0 \ \wedge \ t_1 \leq U}_{\varphi_2}$$

$$\rightarrow$$

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

The implication uses the abbreviation

$$move(\ell_i, \ell_j): \quad \pi_1 = \ell_i \ \wedge \ \pi'_1 = \ell_j.$$

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 W2 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 response property

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

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 summarize the auxiliary assertions, their helpful clocks and bounds and their ranking functions, and display the possible transitions between the assertions.

We define a *ranked diagram* to be a basic verification diagram satisfying:

– The terminal node is labeled by an assertion φ_0. All other nodes are labeled by a pair of assertions: ϕ_i and β_i, for $i = 1, \ldots, m$, and a ranking function δ_i. The assertion β_i has the form $t_i \leq B_i$, where $t_i \in C$ 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. For uniformity, we define $\phi_0 = \varphi_0$.

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: $t \leq B$ 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:

$$\rho_\tau \wedge \varphi \rightarrow (\varphi' \wedge \delta' = \delta \wedge t \leq t' \leq B' \leq B) \vee (\varphi' \wedge \delta \succ \delta') \vee$$
$$(\varphi'_1 \wedge \delta \succ \delta'_1) \vee \cdots \vee (\varphi'_k \wedge \delta \succ \delta'_k).$$

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

$$\rho_\tau \ \wedge \ \varphi \ \ \rightarrow \ \ (\varphi' \ \wedge \ \delta' = \delta \ \wedge \ t \leq t' \leq B' \leq B) \ \vee \ (\varphi' \ \wedge \ \delta \succ \delta').$$

Valid ranked diagrams. The consequences of having a valid ranked diagram are stated in the following claim:

Claim 8 *If D is a P-valid ranked diagram with nodes $\varphi_0, \ldots, \varphi_m$, then*

$$P \models \bigvee_{j=0}^{m} \varphi_j \ \ \Rightarrow \ \ \diamondsuit \ varp_0 \, .$$

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

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

then we can conclude:

$$P \models p \ \ \Rightarrow \ \ \diamondsuit q.$$

In case there is a subset $N \subseteq \{1, \ldots, m\}$ such that $p \rightarrow \bigvee_{i \in N} \varphi_i$, we identify φ_i, $i \in N$ as initial nodes.

Example 11. 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, supports the inference of a response formula from a waiting-for formula of a special form. The rule refers to a *rigid expression* B, which is an expression that does not change its value from one state to the next.

Justification. Assume that the waiting-for premise is P-valid. Consider a P-computation σ, and a p-position $j \geq 0$ in σ. By the waiting-for formula,

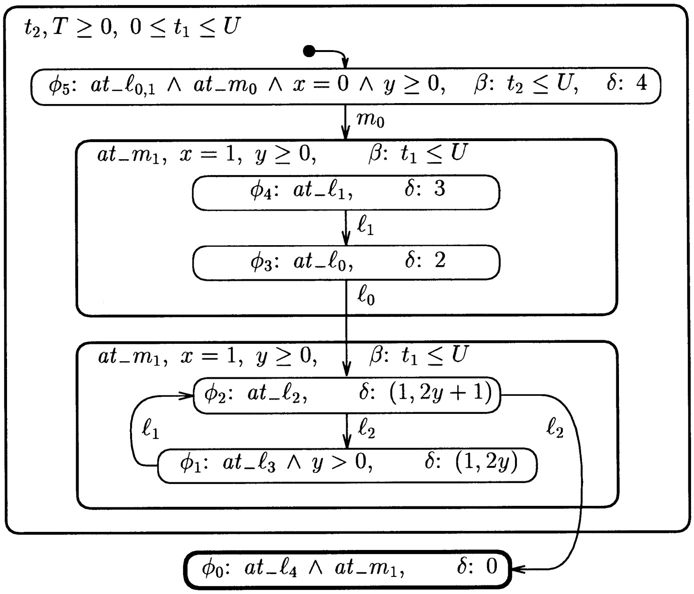

Fig. 29. A ranked diagram, establishing the formula
$$at\_\ell_0 \wedge at\_m_0 \wedge x = y = t_1 = t_2 = T = 0 \quad \Rightarrow \quad \diamond(at\_\ell_4 \wedge at\_m_1)$$

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)

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 the constant value of B at all positions. It follows that j must be followed by an r-position.

Example 12. 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

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

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

In Fig. 31, we present a waiting diagram which establishes the UP-DOWN$_{[1,5]}$-validity of the waiting-for formula

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

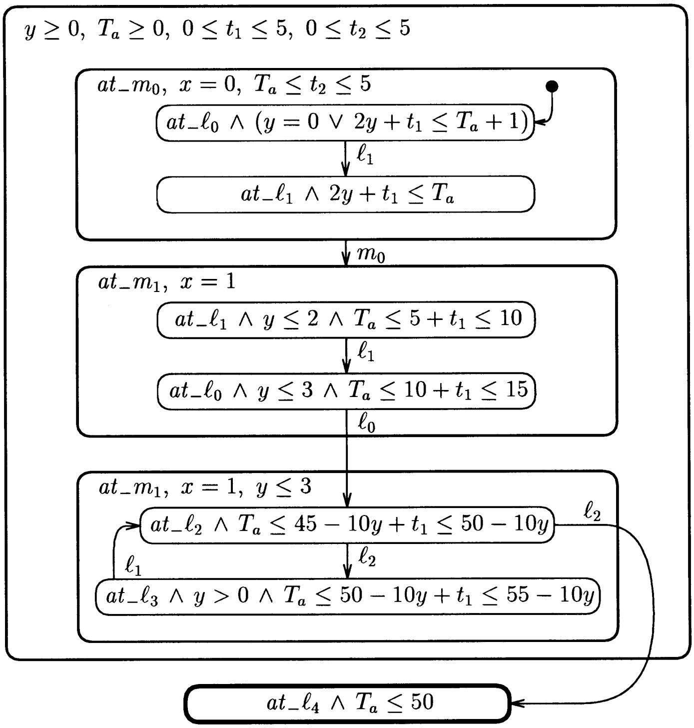

Fig. 31. A waiting diagram, establishing the formula
$at\_\ell_0 \wedge at\_m_0 \wedge x = y = T_a = 0 \quad \Rightarrow \quad (T_a \leq 50)\mathcal{W}(at\_\ell_4 \wedge T_a \leq 50)$

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:

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

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

5.9 Are rules CB-CHAIN *and* CB-WELL *really necessary?*

Rule W$\rightarrow$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$\rightarrow$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 12, and establish the waiting-for formula

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

where

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

Applying rule W$\rightarrow$R to this formula, one can infer the response formula

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

guaranteeing 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$\rightarrow$R. On the other hand, in Example 10, 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$\rightarrow$R, but is provable by rule CB-WELL.

As the second answer justifying the introduction 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 requires 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 little 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 analysis, and are only interested in the qualitative fact that *eventually* q will occur (which is the essence of the $\diamond$ temporal 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.

A more careful analysis of the conditions under which the investment in rules CB-CHAIN and CB-WELL is justified, requires further study and experimentation.

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 statements 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 cases is the following:

Claim 9 *Let P be an* SPL$_T$ *program in which the upper bound assigned to each executable statement is a positive constant. Then Φ_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 $(= s_k)$ 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 $s \Vdash \Omega(\Delta)$.
- 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 system 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 *await* statement.

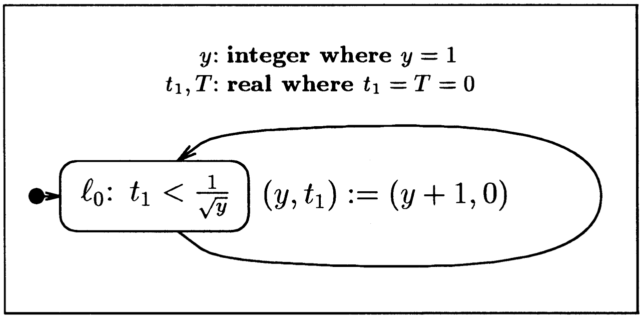

Fig. 32. A non-zeno CTS Φ_4 with upper bounds tending to 0

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 times by the amount $U_m > 0$ and, therefore, the run is time-divergent.

6.1 A rule for establishing non-zenoness

While Claim 9 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.

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

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

For assertion φ,

N1. $\Phi \Vdash \varphi$

N2. $\Phi \models AG(\varphi \wedge T_a = 0 \quad \rightarrow \quad EF(T_a \geq 1))$

Φ is non-zeno

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

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 [FG96].

Premise N2 states that, from every φ-state s, it is possible to trace a computation segment in which time increases by at least 1 from its value at s. We use the constant a to represent the global time at s.

Justification. By premise N1, all Φ-reachable states satisfy the assertion φ. Let s be an arbitrary reachable Φ-state. By N1, it satisfies φ. By N2, we can construct a computation segment from s to another state s_1, such that $s_1[T] \geq s[T] + 1$. Applying premise N2 to s_1, we are guaranteed of a computations segment leading from s_1 to some state s_2, such that $s_2[T] \geq s_1[T] + 1$.

Proceeding in this manner, we can construct a time-divergent run, starting at s. It follows that any finite run, such as the one leading to s, can be extended to a computation. We conclude that Φ is non-zeno.

6.2 Verifying possibility formulas

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

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

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

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

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

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

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.

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

The rule requires finding auxiliary assertions φ_i, functions $Next_i\colon\ \Sigma \mapsto \Sigma$, and transitions τ_i, $i = 1, \ldots, m$, a well-founded domain $(\mathcal{A}, \succ)$, and ranking functions $\delta_i\colon\ \mathcal{S} \mapsto \mathcal{A}$. 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 $j_1 = 0$, we are done, since s_1 satisfies $\varphi_0 = q$.

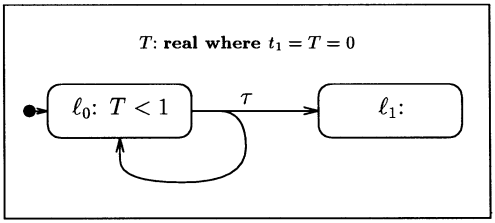

Fig. 35. A non-zeno CTS Φ_5

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 not 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 13. We illustrate the application of rule G-POSS for proving that the possibility formula

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

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

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

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

is Φ_5-valid.

Systems with Deterministic Transitions

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

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 $\delta_0, \ldots, \delta_m \colon \Sigma \mapsto \mathcal{A}$,

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)

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 implies $\Omega(\Delta_i)$, the enabling condition 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 14. We illustrate the use of rule POSS for proving that the possibility formula

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

is valid over CTS Φ_4, where u is an auxiliary rigid variable recording the value of y at the state described by $at\_\ell_0 \wedge 1 \leq y \wedge 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \wedge T_a = 0$.

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:

$$\begin{array}{llllllll}
\varphi_0: & T_a \geq 1 & \tau_0: & \tau_{00} & \Delta_0: & 0 & \delta_0: & 0 \\
\varphi_1: & \multicolumn{7}{l}{at_\ell_0 \ \wedge\ 1 \leq u \leq y \leq 2u+1 \ \wedge\ t_1 = 0 \ \wedge\ \frac{y-u-1}{\sqrt{2u+1}} \leq T_a < 1} \\
 & & \tau_1: & \textit{tick} & \Delta_1: & \frac{1}{\sqrt{y}} & \delta_1: & (|2u+1-y|, 2) \\
\varphi_2: & \multicolumn{7}{l}{at_\ell_0 \ \wedge\ 1 \leq u \leq y \leq 2u+1 \ \wedge\ t_1 = \frac{1}{\sqrt{y}} \ \wedge\ \frac{y-u}{\sqrt{2u+1}} \leq T_a < 1} \\
 & & \tau_2: & \tau_{00} & \Delta_2: & 0 & \delta_2: & (|2u+1-y|, 1) \\
\varphi_3: & \multicolumn{7}{l}{at_\ell_0 \ \wedge\ 1 \leq u \leq y \leq 2u+1 \ \wedge\ 0 < t_1 < \frac{1}{\sqrt{y}} \ \wedge\ 0 \leq T_a < 1} \\
 & & \tau_3: & \textit{tick} & \Delta_3: & \frac{1}{\sqrt{y}} - t_1 & \delta_3: & 2u+2\,.
\end{array}$$

The idea behind this selection is the following. Starting in a state at which $y = u \geq 1$ 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 $u+1$ times, letting y increase from u to $2u+1$. Since the time step in each round is decreasing, the total time increase is not less than $u+1$ times the last time increment which is $\frac{1}{\sqrt{2u+1}}$. Thus the total time increase is not less than

$$\frac{u+1}{\sqrt{2u+1}} \ \geq\ 1,$$

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

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

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

is valid over CTS Φ_4.

6.3 *Possibility diagrams*

Proofs according to rule POSS can be succinctly represented by special type of verification diagram. A possibility diagram is a basic verification diagram, satisfying the following constraints:

- The terminal node is labeled by an assertion denoted φ_0. All other nodes are labeled by an assertion and a ranking function δ.
- Every non-terminal node must have an edge departing from them.
- Edges are labeled by the name of either an untimed transition in the program, as in basic verification diagrams, or by a label of the form $tick(\Delta)$.
- All edges departing from the same node must have the same label.

Verification conditions implied by a possibility diagram. Consider a non-terminal node labeled by assertion φ and ranking function δ. 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:

$$\begin{array}{ll} \text{P2.} & \rho_{tick} \ \wedge\ T' = T + \Delta \ \wedge\ \varphi \\ & \qquad \rightarrow (\varphi_1' \ \wedge\ \delta \succ \delta_1') \ \vee\ \cdots \ \vee\ (\varphi_k' \ \wedge\ \delta \succ \delta_k') \\ \text{P3t.} & \varphi \ \rightarrow\ \Omega(\Delta)\,. \end{array}$$

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

$$\begin{array}{ll} \text{P2.} & \rho_{\tau} \ \wedge\ T' = T \ \wedge\ \varphi \\ & \qquad \rightarrow (\varphi_1' \ \wedge\ \delta \succ \delta_1') \ \wedge\ \cdots \ \wedge\ (\varphi_k' \ \wedge\ \delta \succ \delta_k') \\ \text{P3n.} & \varphi \ \rightarrow\ En(\tau)\,. \end{array}$$

Valid possibility diagrams. The consequences of having a valid possibility diagram are stated in the following claim:

Claim 10 *If D is a P-valid possibility diagram with nodes $\varphi_0, \ldots, \varphi_m$, then*

$$P \models AG\left(\bigvee_{j=0}^{m} \varphi_j \ \rightarrow\ EF\varphi_0\right).$$

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

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

then we can conclude:

$$P \models AG(p \ \rightarrow\ EFq).$$

In case there is a subset $N \subseteq \{1, \ldots, m\}$ such that $p \rightarrow \bigvee_{i \in N} \varphi_i$, we identify φ_i, $i \in N$ as initial nodes.

Example 15. In Fig. 37, we present a possibility diagram that establishes the possibility property

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

for system Φ_4.

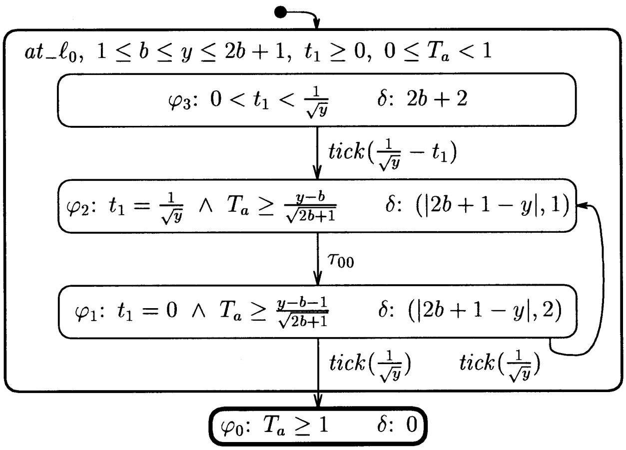

Fig. 37. A possibility diagram, establishing the formula
$AG\left(at\_\ell_0 \wedge 1 \leq y = u \wedge 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \wedge T_a = 0 \quad\rightarrow\quad EF(T_a \geq 1)\right)$

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

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

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

$$AG\ \left(at\_\ell_0 \wedge y \geq 1 \wedge 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \wedge T_a = 0 \right. \\ \left. \rightarrow \quad EF(T_a \geq 1)\right). \tag{2}$$

Example 14 establishes the possibility formula

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

to which we may apply existential quantification over the rigid variable u to obtain

$$AG\ \left(\exists u\colon \left(at\_\ell_0 \wedge y \geq 1 \wedge 0 \leq t_1 \leq \frac{1}{\sqrt{y}} \wedge y = u \wedge T_a = 0\right) \right. \\ \left. \rightarrow EF(T_a \geq 1)\right), \tag{3}$$

using well-established simplification rules for rigid quantification (see, for example [MP93b]).

Since the left-hand sides of the implications in (2) and (3) can be shown to be equivalent, it follows that the formula (2) is Φ_4-valid.

We conclude that CTS Φ_4 is non-zeno.

7 Hybrid systems

In this section we consider the case of hybrid systems. Similar to our treatment of real-time systems, we present first a computational model for hybrid systems that can be viewed as an extension of the CTS model. Then we present rule INV-H for proving invariance properties of hybrid systems, and illustrate its use.

7.1 Computation model: phase transition system

Hybrid systems are modeled as phase transition systems (PTS). The PTS model was originally presented in [MMP92] and [MP93c]. The PTS model presented here is an extension of the CTS model. A closely related model for hybrid systems is presented in [ACH$^+$95].

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

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

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

– $\mathcal{T}$: A finite set of *transitions*, defined as in the CTS model.
– $\mathcal{A}$: A finite set of *activities*. Each activity $\alpha \in \mathcal{A}$ is represented by an *activity relation*:

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

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

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

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

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

That is, $F_i^\alpha(V^0, 0)$ agrees with the initial value of x_i, and the effect of evolution of length t on the master clock (integrator x_m) is to add t to T. It is required that the activation conditions associated with the different activities be exhaustive and exclusive, i.e., exactly one of them holds on any state.
– Π : The *time-progress condition*, defined as in the CTS model.

The enabling condition of a transition τ can always be written as $\delta \wedge \kappa$, where δ is the largest sub-formula that does not depend on integrators. We call κ the *integrator component* of the enabling condition, and denote it by $En_I(\tau)$.

In descriptions of concrete hybrid systems, the evolution functions F^α (V^0, t) are often presented by sets of ordinary differential equations of the form $\dot{x}_j = g_j^\alpha(V)$ for $j = 1, \ldots, m$. In such cases, the evolution functions $F^\alpha(V^0, t)$ can be obtained as solutions of the differential equations. It is straightforward to extend the model to also cover non-deterministic evolutions. In such cases, we may represent the evolution functions as solutions of differential inclusions.

Example 16. Consider the hybrid system Φ_1 presented in Fig. 38.

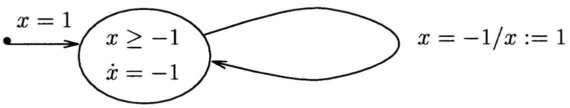

Fig. 38. A hybrid system Φ_1

This system can be modeled by the following PTS:

$$
\begin{array}{ll}
V = I : \{x, T\} & \\
\Theta : & x = 1 \ \wedge \ T = 0 \\
\mathcal{T} : & \{\tau\} \text{ where } \rho_\tau : x = -1 \ \wedge \ x' = 1 \ \wedge \ T' = T \\
\mathcal{A} : & \{\alpha\} \text{ with activity relation (omitting the } \alpha \text{ subscript and superscript)} \\
& \underbrace{\mathit{true}}_{p} \rightarrow \underbrace{x = x^0 - t}_{F(x^0, t)} \\
\Pi : & x \geq -1
\end{array}
$$

The behavior of this system is (informally) presented in Fig. 39.

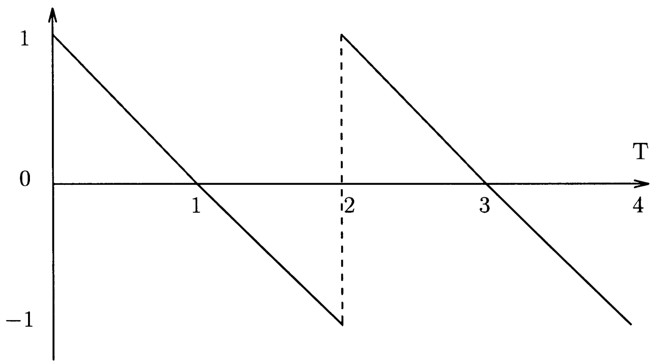

Fig. 39. Behavior of PTS Φ_1

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

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

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

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

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

Runs and computations. The *runs* and *computations* of a phase transition system $\Phi : \langle V, \Theta, \mathcal{T}, \mathcal{A}, \Pi \rangle$ are defined as in the CTS model.

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

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

– States may be labeled by (unconditional) differential equations. The implication is that the activity associated with the differential equation is active precisely when the state it labels is active.

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

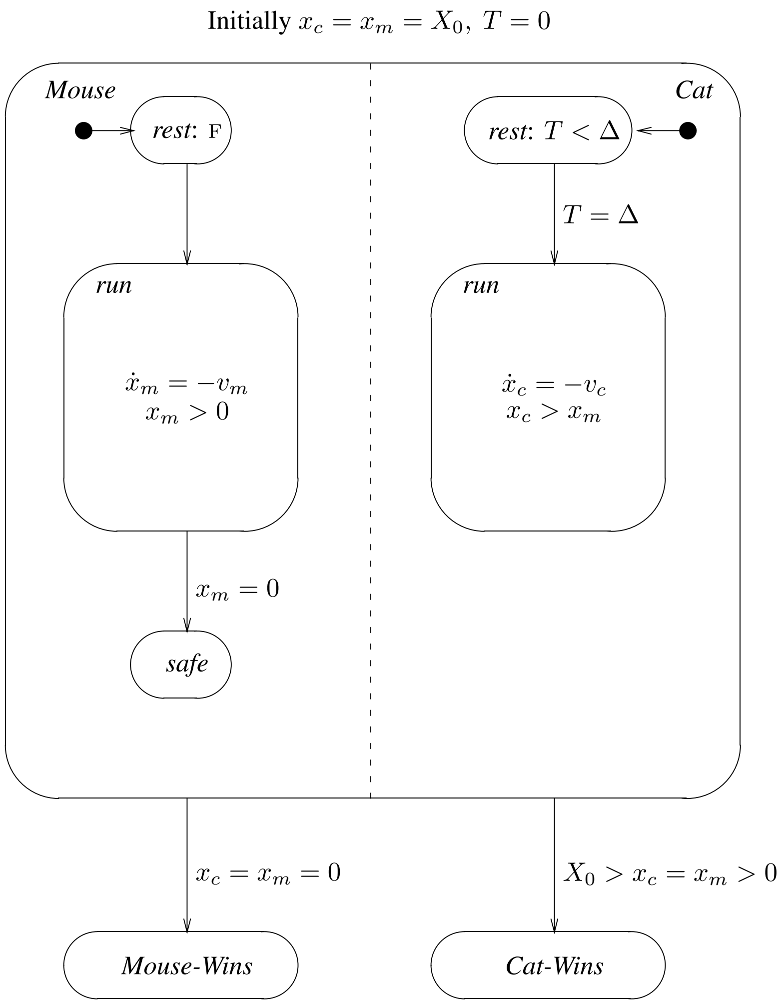

Fig. 40. Specification of Cat and Mouse

the mouse, or will the mouse find sanctuary while the cat crashes against the wall?

The statechart in Fig. 40 describes the possible scenarios. This statechart (and the underlying phase transition system) uses the integrators x_m and x_c, measuring the distance of the mouse and the cat, respectively, from the wall. The waiting time of the cat before it starts running is measured by the master clock T. The statechart refers to the constants X_0, v_m, v_c, and Δ.

A behavior of the system starts with states *M.rest* and *C.rest* active, variables x_m and x_c set to the initial value X_0, and the master clock T set to 0. The mouse proceeds immediately to the state of running, in which its variable x_m changes continuously according to the equation $\dot{x}_m = -v_m$. The cat waits for a delay of Δ before entering its running state, using the master clock T to measure this delay. There are two possible termination scenarios. If the event $x_m = 0$ happens first, the mouse reaches sanctuary and moves to state *safe*, where it waits for the cat to reach the wall. As soon as this happens, detectable by the condition $x_c = x_m = 0$ becoming true, the system moves to state *Mouse-Wins*. The other possibility is that the event $X_0 > x_c = x_m > 0$ occurs first, which means that the cat overtook the mouse before the mouse reached sanctuary. In this case they both stop running and the system moves to state *Cat-Wins*. The compound conditions $x_c = x_m = 0$ and $X_0 > x_c = x_m > 0$ stand for the conjunctions $x_c = x_m \wedge x_m = 0$ and $X_0 > x_c \wedge x_c = x_m \wedge x_m > 0$, respectively.

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

The underlying phase transition system. Following the graphical representation, we identify the phase transition system underlying the picture of Fig. 40. We refer to states in the diagram that do not enclose other states as *basic states*.

- *System Variables:* $V = D \cup I$, where D: $\{\pi_m, \pi_c\}$ and I: $\{x_c, x_m, T\}$. Variables π_m and π_c are control variables whose values are the basic states of the mouse and cat subsyst which are currently active.
- *Initial Condition:* Given by

$$\Theta : \quad \pi_m = M.rest \ \wedge \ \pi_c = C.rest \ \wedge \ x_c = x_m = X_0 \ \wedge \ T = 0.$$

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

$$\begin{aligned}
M.\textit{rest-run} &: \pi_m = M.rest \ \wedge \ \pi_m' = M.run \\
C.\textit{rest-run} &: \pi_c = C.rest \ \wedge \ T = \Delta \ \wedge \ \pi_c' = C.run \\
M.\textit{run-safe} &: \pi_m = M.run \ \wedge \ x_m = 0 \ \wedge \ \pi_m' = M.safe \\
M.win &: \pi_m \in Active \ \wedge \ x_c = x_m = 0 \ \wedge \ \pi_m' = \pi_c' = \textit{Mouse-Wins} \\
C.win &: \pi_c \in Active \ \wedge \ X_0 > x_c = x_m > 0 \ \wedge \ \pi_c' = \pi_m' = \textit{Cat-Wins},
\end{aligned}$$

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

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

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

$$\begin{aligned}
\alpha: \quad x_m = x_m^0 - (at_M.run) \cdot v_m t \ \wedge \ x_c \\
= x_c^0 - (at_C.run) \cdot v_c(t - \Delta) \ \wedge \ T = T^0 + t.
\end{aligned}$$

For assertions φ and p,

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

Fig. 41. Rule INV-H (invariance) applied to PTS Φ

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

– *Time Progress Condition:* Given by

$$\Pi : \begin{pmatrix} \pi_m \neq M.rest \ \wedge \ (\pi_c = C.rest \ \rightarrow \ T < \Delta) \ \wedge \\ (\pi_m = M.run \ \rightarrow \ x_m > 0) \ \wedge \ (\pi_c = C.run \rightarrow \ x_c > x_m) \end{pmatrix}.$$

7.2 Verifying invariance properties over PTS

Invariance properties of hybrid systems can be verified by rule INV-H, presented in Fig. 41.

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

Note that rule INV-H is identical to rule INV, except that we use $\mathcal{T}_H$ as the set of extended transitions. Consequently, we can adopt the notations of invariance diagrams for the concise representation of invariance proofs over PTS's.

Verifying a property of the cat and mouse system. Consider the property that, under the assumptions

$$X_0,\ v_c,\ v_m,\ \Delta > 0, \qquad \frac{X_0}{v_m} < \Delta + \frac{X_0}{v_c} \tag{4}$$

all computations of the Cat and Mouse system satisfy

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

This invariant guarantees that the cat can never win.

In Fig. 42, we present a verification diagram for this invariance property. In this diagram we use control assertions indicating that certain basic states

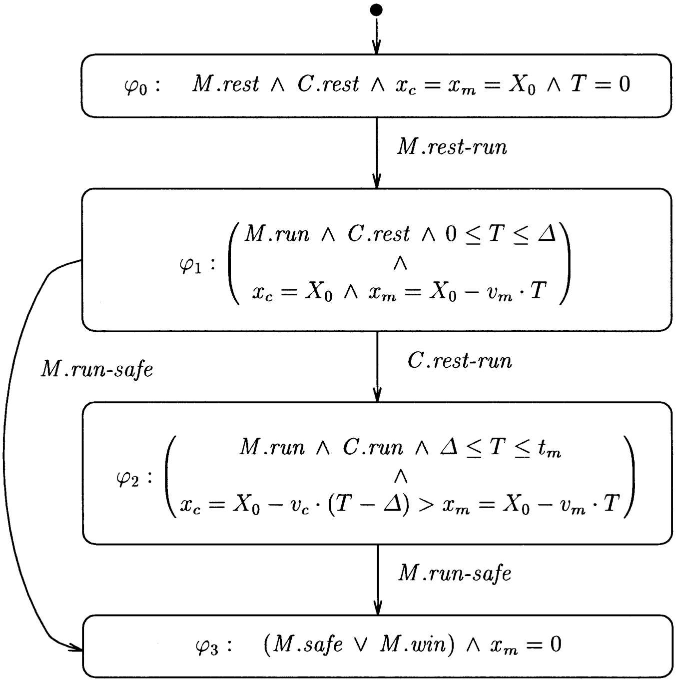

Fig. 42. A hybrid invariance proof diagram

are contained in π_m and π_c. For example, *C.run* stands for $\pi_c = Cat.run$. We also use t_m for $\frac{X_0}{v_m}$, the time it takes the mouse to run the distance X_0.

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

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

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

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

$$\frac{v_m}{v_c} > 1 - \frac{\Delta}{T}\,. \tag{5}$$

From inequality (4), we can obtain

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

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

$$\frac{v_m}{v_c} > 1 - \frac{\Delta}{t_m}. \tag{6}$$

Since $T \leq t_m$, we have $1 - \dfrac{\Delta}{t_m} \geq 1 - \dfrac{\Delta}{T}$ establishing (5).

It remains to show that

$$\underbrace{M.rest \ \wedge \ C.rest \ \wedge \ x_c = x_m = X_0 \ \wedge \ y = 0}_{\Theta} \wedge T = 0 \quad \rightarrow$$
$$\underbrace{M.rest \ \wedge \ C.rest \ \wedge \ x_c = x_m = X_0 \ \wedge \ y = T = 0}_{\varphi_0} \tag{7}$$

$$\varphi_0 \ \vee \ \cdots \ \vee \ \varphi_3 \rightarrow \Big(x_c = x_m \ \rightarrow \ x_c = X_0 \ \vee \ x_m = 0 \Big). \tag{8}$$

Implication (7) is obviously valid. To check implication (8), we observe that both φ_0 and φ_1 imply $x_c = X_0$, φ_2 implies $x_c > x_m$ (using the assumption $\Delta > 0$), and φ_3 implies $x_m = 0$.

This shows that under assumption (4), property

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

is valid for the Cat and Mouse system.

Similar to the clocked transition systems, it is possible to formulate appropriate proof rules for the verification of waiting-for and response properties over phase transition systems. Note that all the rules that rely on the uniform progress of clocks, must refer to either the master clock T, or other variables which progress at a constant rate at all times.

Checking non-zenoness of hybrid systems. Since all hybrid systems contain the master clock T among their integrators, we can apply all the techniques presented in Sect. 6 for establishing that a given PTS is Non-Zeno.

8 Conclusions

In this paper we have presented the real-time model of clocked transition systems (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. We presented verification rules for invariance properties which are identical to the invariance verification rules of fair transition systems [MP95]. For response properties, we presented rules similar to the CHAIN and W-RESP rules of fair transition systems [MP91]. The main differences between the timed and the untimed versions of these rules is that the timed version does not use the concept of a helpful transition but replaces it with the concept of a *helpful clock*, whose boundedness and the fact that it is not reset while its associated assertion holds, implies that we can stay in states that satisfy this assertion only for a bounded time, and must move elsewhere.

We proceeded with the presentation of an approach for verifying that an arbitrary CTS satisfies the non-zeno restriction. We use branching-time TL (CTL) to formulate the non-zeno property, and give a single proof rule to establish the CTL formula.

We concluded with an extension of the CTS model to hybrid systems, and presentation of a rule for verifying safety properties of such systems.

As previously mentioned, the model of Clocked Transition Systems and its verification rules have been successfully implemented in the STeP system [BBC+95]. Many of the examples presented in this paper have been verified within STeP. Implementation of the Hybrid Systems extension is under way.

Acknowledgements. We gratefully acknowledge the careful reading and critical comments of previous versions of this paper by Oded Maler and Tomás Uribe. Nikolaj Bjørner implemented the clocked transition system model and its verification rules within STeP. Special thanks are due to Henny Sipma for a thorough examination of the current (and previous) versions, the detection and correction of some subtle errors, and the STeP mechanical verification of some of the more difficult examples.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine: The algorithmic analysis of hybrid systems. *Theor. Comp. Sci.*, 138:3–34 (1995)

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

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

[AH92] R. Alur, T. Henzinger: Logics and models of real time: A survey. In: J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg (eds.) *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, Vol. 600 of LNCS, pp 74–106. Berlin Heidelberg New York: Springer, 1992

[AH94] R. Alur, T.A. Henzinger: Real-time system = discrete system + clock variables. In: T. Rus, C. Rattray, editors, *Theories and Experiences for Real-time System Development*, AMAST Series in Computing 2, pages 1–29. World Scientific, 1994.

[AL91] M. Abadi, L. Lamport: An old-fashioned recipe for real time. In *Real-Time: Theory in Practice*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 1–27. Springer-Verlag, 1991.

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

[BBC+95] N. Bjørner, I.A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, T.E. Uribe: STeP: The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, November 1995.

[BMS95] I.A. Browne, Z. Manna, H.B. Sipma: Generalized verification diagrams. In *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lect. Notes in Comp. Sci.*, pages 484–498, December 1995.

[BMSU97] N.S. Bjørner, Z. Manna, H.B. Sipma, T.E. Uribe: Deductive verification of real-time systems using STeP. In *4th Intl. AMAST Workshop on Real-Time Systems*, volume 1231 of *Lect. Notes in Comp. Sci.*, pages 22–43. Springer-Verlag, May 1997. To appear in Theoretical Computer Science.

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

[FG96] L. Fix, O. Grumberg: Verificaiton of temporal properties. *Logic and Computation*, 6(3):343–362, 1996.

[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.

[HHWT95] T.A. Henzinger, P.-H. Ho, H. Wong-Toi: A user guide to HYTECH. In *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Aarhus, Denmark, 1995. To appear.

[HK94] T.A. Henzinger, P.W. Kopke: Verification methods for the divergent runs of clock systems. In: H. Langmaack, W.-P. de Roever, J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 351–372. Springer-Verlag, 1994.

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

[KdR83] R. Koymans, W.-P. de Roever: Examples of a real-time temporal logic specifications. In *The Analysis of Concurrent Systems*, pages 231–252. Springer-Verlag, 1983.

[KMP98] Y. Kesten, Z. Manna, A. Pnueli: Verification of clocked and hybrid systems. In: G. Rozenberg, F.W. Vaandrager, editors, *Lectures on Embedded Systems*, volume 1494 of *Lect. Notes in Comp. Sci.*, pages 4 – 73. Springer-Verlag, 1998.

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

[KVdR83] R. Koymans, J. Vytopyl, 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.

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

[MP91] Z. Manna, A. Pnueli: Completing the temporal picture. *Theor. Comp. Sci.*, 83(1):97–130, 1991.

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

[MP93b] Z. Manna, A. Pnueli: *The Temporal Logic of Reactive Systems*. Springer Verlag, New York, 1993.

[MP93c] Z. Manna, A. Pnueli: Verifying hybrid systems. In: R.L. Grossman, A. Nerode, A. Ravn, H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lect. Notes in Comp. Sci.*, pages 4–35. Springer-Verlag, 1993.

[MP94] Z. Manna, A. Pnueli: Temporal verification diagrams. In: T. Ito, 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, A. Pnueli: *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[MS98] Z. Manna, H. Sipma: Deductive verification of hybrid systems using STeP. In *Hybrid Systems: Computation and Control*, volume 1386 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1998.

[MT90] F. Moller, C. Tofts: A temporal calculus of communicating systems. In: J.C.M. Baeten, 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, S. Yovine: From ATP to timed graphs and hybrid systems. In: J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg, editors, *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.

[SBM92] F. B. Schneider, B. Bloom, K. Marzullo: Putting time into proof outlines. In: J.W. de Bakker, C. Huizing, W.P. de Roever, 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, A. Skou, editors, *3rd Computer Aided Verification Workshop*, volume 575 of *Lect. Notes in Comp. Sci.*, pages 376–398. Springer-Verlag, 1991.