Verifying Clocked Transition Systems*

Yonit Kesten† Zohar Manna‡ Amir Pnueli†

Abstract. This paper presents a new computational model for real-time systems, called the *clocked transition system* (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 properties (including waiting-for and time-bounded response properties) of clocked transition systems, and separate rules for proving (time-unbounded) response properties. 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 properties of real-time systems.

1 Introduction

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

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

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

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

- *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: $p \ \wedge \ (T = t_0) \quad \Rightarrow \quad \diamondsuit(q \ \wedge \ T \leq t_0 + d).$
- Minimal separation: $p \ \wedge \ (T = t_0) \quad \Rightarrow \quad \square(T < t_0 + d \ \rightarrow \ \neg q).$

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

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], [KdR85], and [Koy90]) or the *age function* proposed in [MP93].

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 [AL94], which also recommends handling real time with a minimal extension of the reactive-systems formalism.

A model similar to the CTS model presented here was introduced in [AH94], and proof rules for establishing response properties for that 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], [AL94], and the survey in [AH92], for additional logics, models, and approaches to the verification of real-time systems. In the process algebra school, some of the representative approaches to real time are [NSY92], [MT90], and many others are listed in [Sif91].

The paper is organized as follows. In Section 2, we present the real-time computational model of *clocked transition systems* (CTS). In Section 3, we show how a program augmented with timing bounds for the execution of statements can be represented as a CTS. In Section 4, we present rules and verification diagrams for verifying safety properties of CTS's. In Section 5, we present rules and verification diagrams for establishing response properties of CTS's.

A fuller version of this paper is available as the technical report [KMP95].

2 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 $D = \{u_1, \ldots, u_m\}$ the set of *discrete variables* and $C = \{c_1, \ldots, c_k\}$ 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

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

  i.e., $T = 0$ at all initial states.
- $\mathcal{T}$: A finite set of *transitions*. Each transition $\tau \in \mathcal{T}$ is a function

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

  mapping each state $s \in \Sigma$ into a (possibly empty) set of τ-*successor* states $\tau(s) \subseteq \Sigma$.
  The function associated with a transition τ is represented by an assertion $\rho_\tau(V, V')$, called the *transition relation*, which relates a state $s \in \Sigma$ to its τ-successor $s' \in \tau(s)$ by referring to both unprimed and primed versions of the system variables. An unprimed version of a system variable refers to its value in s, while a primed version of the same variable refers to its value in s'. For example, the assertion $x' = x + 1$ states that the value of x in s' is greater by 1 than its value in s.
  For every $\tau \in \mathcal{T}$, it is required that

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

  i.e., the master clock is modified by no transition.
- Π : The *time-progress condition*. An assertion over V. The assertion is used to specify a global restriction over the progress of time.

Extended Transitions

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

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

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

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

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

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

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

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

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

Runs and Computations

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

- *Initiation*: $s_0 \models \Theta$
- *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 state s_i increases beyond any bound.

A Frequently Occurring Case

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

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

where I is some finite index set and, for each $i \in I$, the assertion p_i and the real-valued expression E_i do not depend on the clocks, and $c^i \in C$ 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 limiting formula $\Omega(\Delta)$ can be significantly simplified and assumes the following form:

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

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

Non-Zeno Systems

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

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

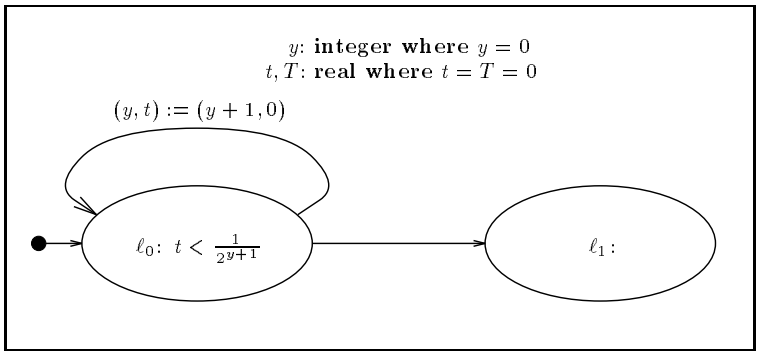

Fig. 1. CTS Φ_1.

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 CTS Φ_1 presented in Fig 1. In Fig 2, we present this CTS in textual form. The system variables include the control variable π, which assumes the values 0 or 1 to indicate that the system is currently at location ℓ_0 or ℓ_1, respectively. The predicate $pres(y, t, T)$ is an abbreviation for the assertion

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

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

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

Fig. 2. CTS Φ_1 in textual form.

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

The *tick* transition for CTS Φ_1 is given by

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

3 Programs as Clocked Transition Systems

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

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

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

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

We refer the reader to Chapter 0 of [MP95] for the construction of a *fair transition system* (FTS) Φ_P corresponding to an SPL program P. This construction introduces an explicit control variable π which ranges over sets of locations. System Φ_P associates a single transition with each *executable* statement, which includes all statements except *concatenation*, *selection*, and *block*. Here, we deviate from the construction of [MP95] in the transition associated with the *await* statement. Instead, we associate with each statement of the form

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

a transition τ_ℓ, whose transition relation ρ_ℓ is given by

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

This revised version ensures that the transition is always enabled when control is at ℓ, regardless of the value of condition c.

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

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

For locations ℓ_i, ℓ_j, and ℓ_k, we denote:

$$\begin{array}{ll} at_\ell_i & : \ell_i \in \pi \\ at_\ell_{j,k} & : at_\ell_j \vee at_\ell_k \end{array}$$

3.1 The CTS corresponding to an SPL$_T$ program

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

As a first step, we construct the FTS $\Phi_P: \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ corresponding to program P. Note that if program P uses only the statements in the previously

presented subset, then $\mathcal{J} = \mathcal{T} - \{\tau_I\}$ and $\mathcal{C} = \emptyset$. Assume that program P is a parallel composition of m processes. That is, P has the form

$$\text{declarations;} \quad P_1 \parallel \cdots \parallel P_m,$$

where each P_i is a (sequential) statement.

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

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

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

  Thus, the transition can be taken only when t_i, the clock corresponding to process P_i, is not below l_τ, the lower bound associated with τ. When taken, the transition resets clock t_i to 0.
- *Time-progress condition Π:* For each executable statement $\ell\colon S$ in process P_i, Π includes the conjunct

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

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

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

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

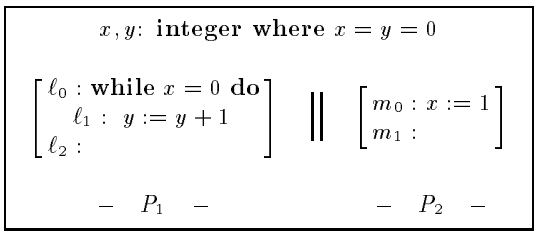

Fig. 3. Program ANY-Y.

To make it an SPL$_T$ program, we uniformly associate each of its executable statements with the time bounds $[3, 5]$. The CTS $\Phi_{\text{ANY-Y}_{[3,5]}}$ associated with ANY-Y$_{[3,5]}$ is defined as follows:

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

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

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

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

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

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

- *Time-progress condition:*

$$\Pi: \quad (at_\ell_{0,1} \rightarrow t_1 < 5) \wedge (at_m_0 \rightarrow t_2 < 5)$$

The *tick* transition for this system is given by

$$tick: \quad \exists \Delta > 0.\ pres(\pi, x, y) \ \wedge \ (t_1', t_2', T') = (t_1 + \Delta, t_2 + \Delta, T + \Delta) \wedge (at_\ell_{0,1} \rightarrow t_1 + \Delta \leq 5) \ \wedge \ (at_m_0 \rightarrow t_2 + \Delta \leq 5)$$

3.2 Specification Language

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

- *State formulas* (*assertions*) - any first-order formula, 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*.
- $p \Rightarrow \diamondsuit r$ — 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 $s \models p$ to indicate that p holds (is true) over s. Let $\sigma : s_0, s_1 \ldots$ be an infinite sequence of states, to which we refer as a *model*. For an assertion p, we say that $j \geq 0$, is a *p-position* if $s_j \models p$. Satisfaction of (the three considered) temporal formulas over a model σ is defined as follows:

- A model σ satisfies the invariance formula $\Box p$, 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 interval of contiguous q-positions, called a q-interval, can either extend to infinity or terminate in an r-position which is not in the interval. That is,

  $\sigma[i] \Vdash p$ implies $\sigma[j] \Vdash q$ for all $j \geq i$, or
  $\sigma[k] \Vdash r$ for some $k \geq i$ and $\sigma[j] \Vdash q$ for all j, $i \leq j < k$.

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

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

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

4 Verifying Safety Properties of Clocked Transition Systems

As explained in [MP91b], the properties expressible by temporal logic can be arranged in a hierarchy that identifies different classes of properties according to the form of formulas expressing them.

In this section, we present methods for verifying safety properties of clocked transitions systems. As representative of the safety class, we consider the properties that can be specified by a *waiting-for* formula of the form

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

We refer to such properties as *precedence properties*.

Rule WAIT, presented in Fig. 4, is the main tool for verifying precedence properties of a CTS Φ.

For assertions p, q, φ, and r,

U1. $\varphi \rightarrow q$

U2. $p \rightarrow r \vee \varphi$

U3. $\rho_\tau \wedge \varphi \rightarrow r' \vee \varphi'$ for every $\tau \in \mathcal{T}_T$

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

Fig. 4. Rule WAIT (waiting-for) applied to CTS P.

The rule uses an auxiliary assertion φ. Premise U1 of rule WAIT requires that the auxiliary assertion φ implies assertion q. Premise U2 requires that every p-state must satisfy r or φ. Premise U3 requires that every τ-successor of a φ-state satisfies r or φ, where τ is any transition in $\mathcal{T}_T^{\Phi}$ (the extended transitions of Φ). It follows that every p-state in a computation of Φ initiates a (possibly empty) φ-interval, which can be terminated only by an r-state. Due to implication U1, every φ-state is also a q-state and, therefore, formula $p \Rightarrow (q \,\mathcal{W}\, r)$ is Φ-valid.

Example 3. We use rule WAIT to establish the precedence property

$$\underbrace{\Theta}_{p} \quad \Rightarrow \quad \underbrace{at_\ell_{0,1}}_{q} \,\mathcal{W}\, \underbrace{at_m_1}_{r}$$

over program ANY-Y (Fig. 3). This formula claims that process P_1 cannot terminate before process P_2. That is, process P_1 cannot reach ℓ_2 before P_1 reaches location m_1.

We apply rule WAIT to this choice of p, q, and r, taking

$$\varphi: \quad x = 0 \wedge at_\ell_{0,1}$$

as the auxiliary assertion. It is not difficult to see that all premises of rule WAIT are state valid for this choice of p, q, r, and φ.

This establishes that the considered waiting-for formula is valid over program ANY-Y, i.e.,

$$\text{ANY-Y} \quad \models \quad \Theta \;\Rightarrow\; (at_\ell_{0,1} \,\mathcal{W}\, at_m_1). \;\blacksquare$$

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

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

Verification Diagrams

An effective way of presenting the verification of a precedence property by rule WAIT is provided by the graphical formalism of *verification diagrams* [MP94]. A *waiting* (*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.
  Node φ_0 is considered a *terminal* node and is graphically identified by being drawn with bold-face boundaries.
- *Edges* in the graph represent transitions between assertions. Each edge is labeled by the name of a transition in the program. We refer to an edge labeled by τ as a *τ-edge*. No edges depart form the terminal node.
- Some of the nodes are designated as *initial nodes*. They are annotated by an entry arrow .

For example, in Fig. 5 we present a verification diagram for program ANY-Y.

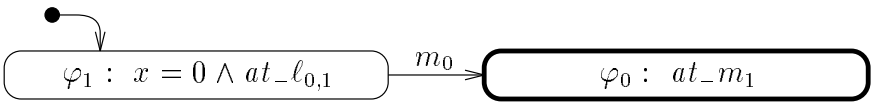

Fig. 5. A Waiting Verification Diagram.

Verification Conditions Implied by a Waiting Diagram

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

$$\rho_\tau \ \wedge \ \varphi \ \rightarrow \ \varphi' \ \vee \ \varphi'_1 \ \vee \ \cdots \ \vee \ \varphi'_k .$$

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

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

Valid Waiting Diagrams

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

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

Claim 1 *A Φ-valid waiting diagram with nodes $\varphi_0, \ldots, \varphi_m$ establishes*

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

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

$$\text{D1:} \quad \Phi \models (\bigvee_{j=1}^{m} \varphi_j) \ \rightarrow \ q, \qquad \textit{and} \qquad \text{D2:} \quad \Phi \models p \ \rightarrow \ \bigvee_{j=0}^{m} \varphi_j ,$$

then we can conclude:

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

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

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

The second part of the claim considers different p and q but explicitly requires, using D1 and D2, that premises U1 and U2 are Φ-state valid. The conclusion follows by rule WAIT. ▟

For example, all the verification conditions associated with the diagram of Fig. 5 are valid over program ANY-Y and so are $\varphi_0 = r$, and conditions D1, D2 for the choice of $p{:}\,\Theta$, $q{:}\,at_\ell_{0,1}$, and $r{:}\,at_m_1$. Consequently. the verification diagram establishes

$$\text{ANY-Y} \quad \models \quad \Theta \;\Rightarrow\; (at_\ell_{0,1}\,\mathcal{W}\,at_m_1).$$

Encapsulation Conventions

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

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

These encapsulation conventions are illustrated in the following example.

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

$$\Theta \quad \Rightarrow \quad (T \leq 15)\,\mathcal{W}\,(at_\ell_2 \wedge at_m_1)$$

This formula claims that, as long as the program has not terminated, T is bounded by 15. Since in all computations, time must eventually exceed 15, this guarantees terminations within 15 time units.

The proof is presented by the waiting diagram of Fig. 6. 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 in the diagram. ∎

Completeness of Rule WAIT

Rule WAIT is complete for verifying precedence properties of non-zeno clocked transition systems. This is stated by the following claim.

Claim 2 *If formula $p \Rightarrow (q\,\mathcal{W}\,r)$ is valid over non-zeno* CTS *Φ, then there exists an assertion φ such that premises* U1-U3 *of rule* WAIT *are Φ-state valid.*

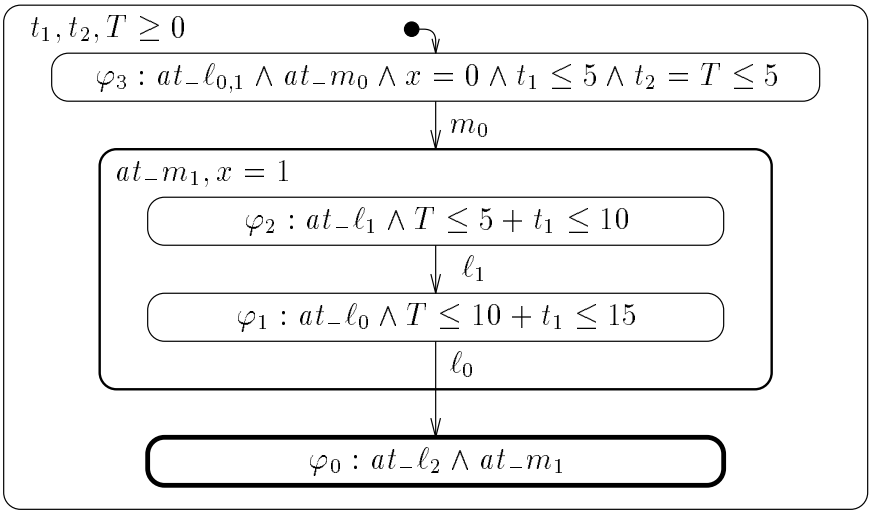

Fig. 6. Termination of ANY-Y$_{[3,5]}$ within 15 time units.

Justification Consider a non-zeno CTS Φ and assertions p, q, and r, such that $p \Rightarrow (q \mathcal{W} r)$ is Φ-valid.

We define an (*execution*) *segment* (of Φ) to be a sequence $\sigma : s_1, \ldots, s_k$, $k > 0$, such that, for every $i \in \{1, \ldots, k-1\}$, $s_{i+1} \in \tau(s_i)$ for some $\tau \in \mathcal{T}_T$. We say that the segment σ *departs from* s_1. For a segment $\sigma : s_1, \ldots, s_k$ and assertion ψ, we define $earliest_\sigma(\psi)$ to be the smallest $j \in \{1, \ldots, k\}$ such that s_j satisfies ψ, or $k+1$ if there exists no such j (i.e., no state in σ satisfies ψ). A segment σ is called a $(q \mathcal{W} r)$-*segment* if

$$earliest_\sigma(r) \leq earliest_\sigma(\neg q).$$

Otherwise, σ is called a $\neg(q \mathcal{W} r)$-*segment*. It is not difficult to see that all models extending a given $\neg(q \mathcal{W} r)$-segment σ satisfy $\neg(q \mathcal{W} r)$. A state s is defined to be a $(q \mathcal{W} r)$-*state* if all execution segments departing from s are $(q \mathcal{W} r)$-segments.

Following are some properties of $(q \mathcal{W} r)$-states:

P1. Every r-state is a $(q \mathcal{W} r)$-state.
P2. If s is a $(q \mathcal{W} r)$-state which does not satisfy r, then s satisfies q and every τ-successor of s is also a $(q \mathcal{W} r)$-state.
P3. Under the assumption that $p \Rightarrow (q \mathcal{W} r)$ is Φ-valid, every accessible p-state is a $(q \mathcal{W} r)$-state.

To establish property P3, assume to the contrary that there exists an accessible p-state $\widehat{s}$ which is not a $(q \mathcal{W} r)$-state. This implies that there exists a finite run $\sigma: s_0 \ldots, s_a, \ldots, s_b$, such that $s_a = \widehat{s}$, and $s_a, \ldots, s_b$ is a $\neg(q \mathcal{W} r)$-segment. Since Φ is non-zeno, the run σ can be extended into a computation σ^*. However, position a of σ^* satisfies p (since $s_a = \widehat{s}$ is a p-state) but does not satisfy $q \mathcal{W} r$, since no extension of $s_a \ldots, s_b$ can satisfy $q \mathcal{W} r$. This contradicts the assumption that $p \Rightarrow (q \mathcal{W} r)$ is Φ-valid.

Using the techniques of [MP91a] or Chapter 2 of [MP95], we can construct a (first-order) assertion *waiting* which precisely characterizes the set of $(q\,\mathcal{W}\,r)$-states, i.e., a state satisfies the assertion *waiting* iff it is a $(q\,\mathcal{W}\,r)$-state. We take φ to be $\mathit{waiting} \wedge \neg r$ and show that all premises of rule WAIT are Φ-state valid. Since we are only proving *relative* completeness, it is enough to show Φ-state validity of the premises, assuming an oracle that provides proofs or otherwise verifies all generally Φ-state valid assertions.

U1. $\varphi \rightarrow q$
This is a direct consequence of property P2, since a state satisfying the assertion φ: $\mathit{waiting} \wedge \neg r$, is a $(q\,\mathcal{W}\,r)$-state which does not satisfy r.

U2. $p \rightarrow r \vee \varphi$
By the definition of φ and property P1, $r \vee \varphi$ is equivalent to *waiting*. The premise then follows from property P3.

U3. $\rho_\tau \wedge \varphi \quad \rightarrow \quad r' \vee \varphi'$, for each $\tau \in \mathcal{T}$.
Observing that $r' \vee \varphi'$ is equivalent to $\mathit{waiting}'$, this premise follows again from property P2.

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 in Fig. 7. Clearly, CTS Φ_3

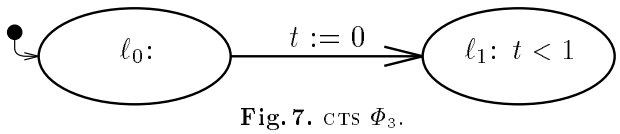

Fig. 7. CTS Φ_3.

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 waiting-for formula $\Theta \Rightarrow (at_\ell_0\,\mathcal{W}\,\mathrm{F})$ is Φ_3-valid. Note, however, that this formula is not valid over all *runs* of Φ_3. In particular, any run that enters state ℓ_1 falsifies $\Theta \Rightarrow (at_\ell_0\,\mathcal{W}\,\mathrm{F})$.

Examination of the arguments for the soundness of rule WAIT shows that, in fact, the rule establishes the validity of $p \Rightarrow (q\,\mathcal{W}\,r)$ over all runs of Φ. Consequently, rule WAIT cannot be used to verify $\Theta \Rightarrow (at_\ell_0\,\mathcal{W}\,\mathrm{F})$ which is valid over all computations of Φ_3 but not over all runs of Φ_3. ∎

The invariance formula $\Box p$ can be viewed (and verified) as the waiting-for formula $\Theta \Rightarrow (p\,\mathcal{W}\,\mathrm{F})$. We refer readers to [MP95] for detailed discussions of methods for verifying invariance formulas.

5 Verifying Response Properties of Clocked Transition Systems

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

5.1 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. 8. The rule uses auxiliary assertions $\varphi_1, \ldots, \varphi_m$ and refers to assertion q also as φ_0. With each assertion φ_i we associate one of the clocks $c_i \in C$, to which we refer as the *helpful clock*, and a real-valued upper bound b_i. The intention is that while remaining in states satisfying φ_i, the clock c_i is bounded by b_i. Since time in a computation grows beyond any bound, this will imply that we cannot continually stay at φ_i-state for too long.

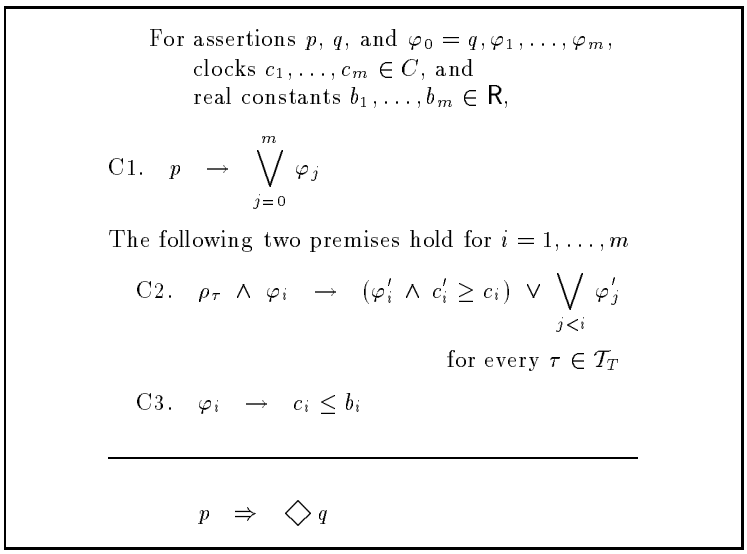

Fig. 8. Rule CB-CHAIN (clock-bounded chain rule for response).

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 c_i.

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

The following claim states the soundness of the rule:

Claim 3 *Rule* CB-CHAIN *is sound for proving that a response formula is Φ-valid.*

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

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

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

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

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

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

Since we cannot have an infinite non-increasing sequence of natural numbers which decreases infinitely many times, there must exist a position $r \geq k$ such that

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

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

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

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

Example 6. 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

$$(\underbrace{at_\ell_0 \wedge at_m_0 \wedge x = t_1 = t_2 = T = 0}_{p}) \quad \Rightarrow \quad \diamondsuit(\underbrace{at_\ell_2 \wedge at_m_1}_{q}).$$

To apply rule CB-CHAIN, we use the following auxiliary assertions, helpful clocks, and bounds:

φ_0: $at_\ell_2 \wedge at_m_1$	–	–
φ_1: $at_\ell_0 \wedge at_m_1 \wedge x = 1 \wedge t_1 \leq 5$	c_1: t_1	b_1: 5
φ_2: $at_\ell_1 \wedge at_m_1 \wedge x = 1 \wedge t_1 \leq 5$	c_2: t_1	b_1: 5
φ_3: $at_\ell_{0,1} \wedge at_m_0 \wedge x = 0 \wedge t_1 \leq 5 \wedge t_2 \leq 5$	c_3: t_2	b_1: 5

We can check that all premises of rule CB-CHAIN are state valid. It follows that the response formula

$$(at_\ell_0 \wedge at_m_0 \wedge x = t_1 = t_2 = T = 0) \quad \Rightarrow \quad \diamondsuit(at_\ell_2 \wedge at_m_1)$$

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* to be a directed labeled graph constructed as follows:

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

Verification Conditions Implied by a Chain Diagram

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

$$\rho_\tau \ \wedge \ \varphi \ \rightarrow \ (\varphi' \wedge c' \geq c) \ \vee \ \varphi'_1 \ \vee \ \cdots \ \vee \ \varphi'_k.$$

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

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

Valid Chain Diagrams

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

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

Claim 4 *A Φ-valid chain diagram with nodes $\varphi_0, \ldots, \varphi_m$ establishes*

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

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

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

then we can conclude:

$$\Phi \quad \models \quad p \;\Rightarrow\; \Diamond q.$$

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

Example 7. In Fig. 9, 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$_{[3,5]}$.

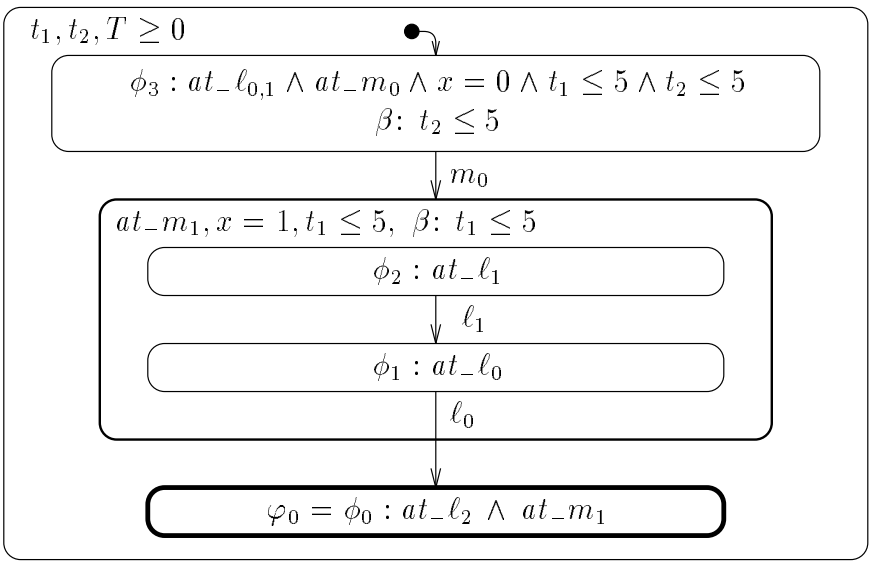

Fig. 9. Verifying termination of program ANY-Y$_{[3,5]}$.

Note the use of encapsulation in labeling the compound node by the common clock bound $t_1 \leq 5$, 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 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 helpful steps which is a priori bounded. For example, in verifying termination of program ANY-Y$_{[3,5]}$, there were 3 helpful step leading to termination. These are represented in the chain diagram of Fig. 9 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 domains.

Well-founded Domains

We define a *well-founded domain* $(\mathcal{A}, \succ)$ to consist of a set $\mathcal{A}$ and a *well-founded* binary relation $\succ$ on $\mathcal{A}$. The binary 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

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

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

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

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

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

Rule CB-WELL

In Fig. 10, 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 $c_i \in C$, to which we refer as the *helpful clock*, and an upper bound B_i, which is a real-valued expression. Also required are a well-founded domain $(\mathcal{A}, \succ)$, and ranking functions δ_i: $\Sigma \mapsto$

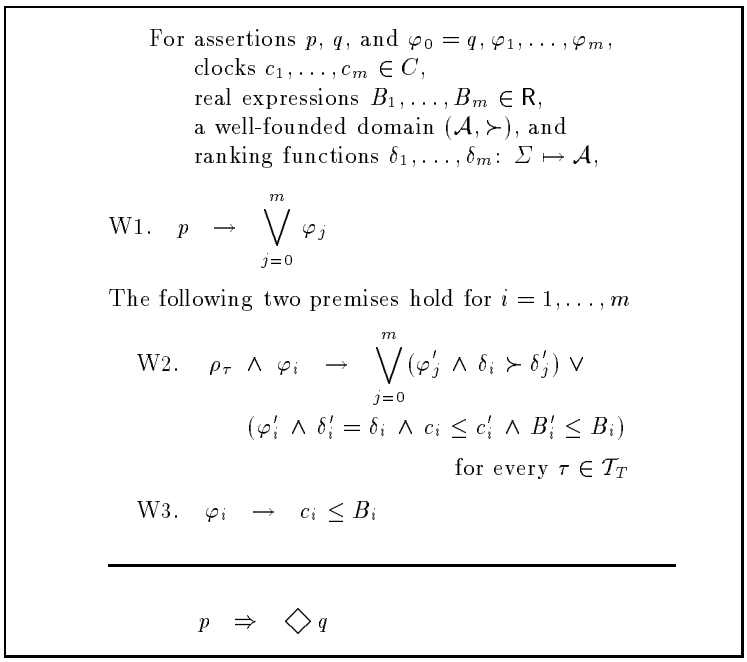

Fig. 10. Rule CB-WELL (clock-bounded well-founded rule for response).

$\mathcal{A}$, $i = 1, \ldots, m$, 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 $\varphi_0 = q, \varphi_1, \ldots, \varphi_m$.

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

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

The following claim states the soundness of the rule:

Claim 5 *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

$$p \Rightarrow \diamondsuit q.$$

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. Thus, $u_k = \delta_{j_k}(s_k)$, which implies that φ_{j_k} holds at 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 $\delta_{j_r}(s_r) = u$. Since the rank never decreases beyond r, all states beyond this position satisfy φ_{j_r}, and the value of c_{j_r} never decreases.

Consider the value of the expression $B_{j_r} - c_{j_r}$ at states s_i, $i \geq r$. By W3, this expression is always non-negative. By W2, its value never increases. Also, whenever a *tick* transition with increment Δ is taken, c_{j_r} increases (as do all clocks) by Δ. Since B_{j_r} does not increase (by W2 and the assumption that the rank remains the same), the value of $B_{j_r} - c_{j_r}$ decreases by Δ in such a *tick* transition. It follows that the master clock T cannot increase by more than $s_r[B_{j_r} - c_{j_r}]$[4] 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 6 *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 $p \Rightarrow \diamondsuit q$ 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 [MP91a] and take for (a single) φ the assertion $pending_q$, constructed in such a way that

$s \models pending_q$ iff s is a $\neg q$-follower of a Φ-accessible p-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 $p \Rightarrow \diamondsuit q$.

Based on a transcendentally inductive construction, we can define a ranking function $\delta : \Sigma \mapsto \mathcal{O}rd$, 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 $\delta(s) > \delta(s')$.

[4] The value of $B_{j_r} - c_{j_r}$ at state s_r

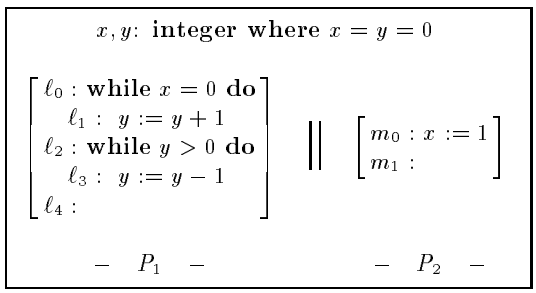

Fig. 11. Program UP-DOWN.

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 $m = 1$, $\varphi_1 = \mathit{pending}_q$, $c_1 = T$, $B_1 = B(s)$, $(\mathcal{A}, \succ) = (\mathcal{O}rd, >)$, and $\delta_1 = \delta$ as defined above.

A fuller version of this proof is presented in [KMP95]. ∎

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

Example 8. Consider program UP-DOWN presented in Fig. 11.

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

$$\underbrace{(at_\ell_0 \wedge at_m_0 \wedge x = y = t_1 = t_2 = T = 0)}_{p} \quad \Rightarrow \quad \diamondsuit(\underbrace{at_\ell_4 \wedge at_m_1}_{q}).$$

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

$\varphi_0\colon$	$at_\ell_4 \wedge at_m_1$		$\delta_0\colon\ 0$
$\varphi_1\colon$	$at_\ell_3 \wedge at_m_1 \wedge x = 1 \wedge y > 0 \wedge t_1 \leq U$	$c_1\colon\ t_1$	$\delta_1\colon\ (1, 2y)$
$\varphi_2\colon$	$at_\ell_2 \wedge at_m_1 \wedge x = 1 \wedge y \geq 0 \wedge t_1 \leq U$	$c_2\colon\ t_1$	$\delta_2\colon\ (1, 2y+1)$
$\varphi_3\colon$	$at_\ell_0 \wedge at_m_1 \wedge x = 1 \wedge y \geq 0 \wedge t_1 \leq U$	$c_3\colon\ t_1$	$\delta_3\colon\ 2$
$\varphi_4\colon$	$at_\ell_1 \wedge at_m_1 \wedge x = 1 \wedge y \geq 0 \wedge t_1 \leq U$	$c_4\colon\ t_1$	$\delta_4\colon\ 3$
$\varphi_5\colon$	$at_\ell_{0,1} \wedge at_m_0 \wedge x = 0 \wedge y \geq 0 \wedge t_1 \leq U \wedge t_2 \leq U$		
		$c_5\colon\ t_2$	$\delta_5\colon\ 4$

It can be shown that all premises of rule CB-WELL are state valid. This establishes that the response property

$$(at_\ell_0 \wedge at_m_0 \wedge x = y = t_1 = t_2 = T = 0) \quad \Rightarrow \quad \diamondsuit(at_\ell_4 \wedge at_m_1).$$

is valid over program UP-DOWN. ∎

5.4 Ranked Diagrams

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

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

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

Verification Conditions Implied by a Ranked Diagram

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

$$\rho_\tau \ \wedge \ \varphi \ \rightarrow \ (\varphi' \wedge \delta' = \delta \wedge c \leq c' \leq B' \leq B) \ \vee \ (\varphi' \wedge \delta \succ \delta') \vee (\varphi'_1 \wedge \delta \succ \delta'_1) \ \vee \ \cdots \ \vee \ (\varphi'_k \wedge \delta \succ \delta'_k).$$

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

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

Valid Ranked Diagrams

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

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

Claim 7 *A Φ-valid ranked diagram with nodes $\varphi_0, \ldots, \varphi_m$, establishes*

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

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

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

then we can conclude:

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

Example 9. In Fig. 12, we present a ranked diagram which establishes that the response property

$$(at_\ell_0 \wedge at_m_0 \wedge x = y = t_1 = t_2 = T = 0) \quad \Rightarrow \quad \Diamond(at_\ell_4 \wedge at_m_1).$$

is valid over program UP-DOWN. ▟

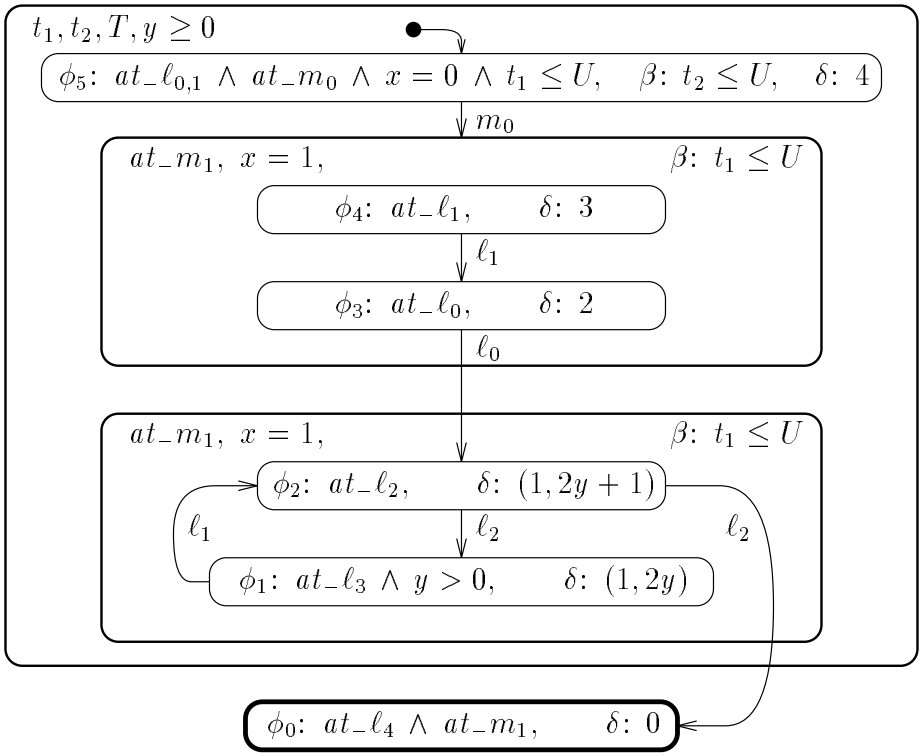

Fig. 12. A ranked diagram, establishing the formula
$(at_\ell_0 \wedge at_m_0 \wedge x = y = t_1 = t_2 = T = 0) \quad \Rightarrow \quad \Diamond(at_\ell_4 \wedge at_m_1)$

5.5 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. 13, infers a response formula from a waiting-for formula of a special form.

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

$$\frac{p \;\Rightarrow\; (q \wedge T \leq B)\,\mathcal{W}\, r}{p \;\Rightarrow\; \diamondsuit\, r}$$

Fig. 13. Rule W→R (from waiting-for to response formulas).

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

Example 10. 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

$$(at_\ell_0 \wedge at_m_0 \wedge x = y = T = 0) \quad \Rightarrow \quad \diamondsuit(at_\ell_4 \wedge at_m_1 \wedge T \leq 50)$$

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

In Fig. 14, we present a waiting diagram which establishes the UP-DOWN$_{[1,5]}$-validity of the waiting-for formula

$$(at_\ell_0 \wedge at_m_0 \wedge x = y = T = 0) \quad \Rightarrow \quad (T \leq 50)\,\mathcal{W}\,(at_\ell_4 \wedge at_m_1 \wedge T \leq 50)$$

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:

$$\square\big((at_\ell_0 \;\rightarrow\; 0 \leq t_1 \leq 5) \;\wedge\; (at_m_0 \;\rightarrow\; 0 \leq t_2 \leq 5)\big).$$

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

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

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

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

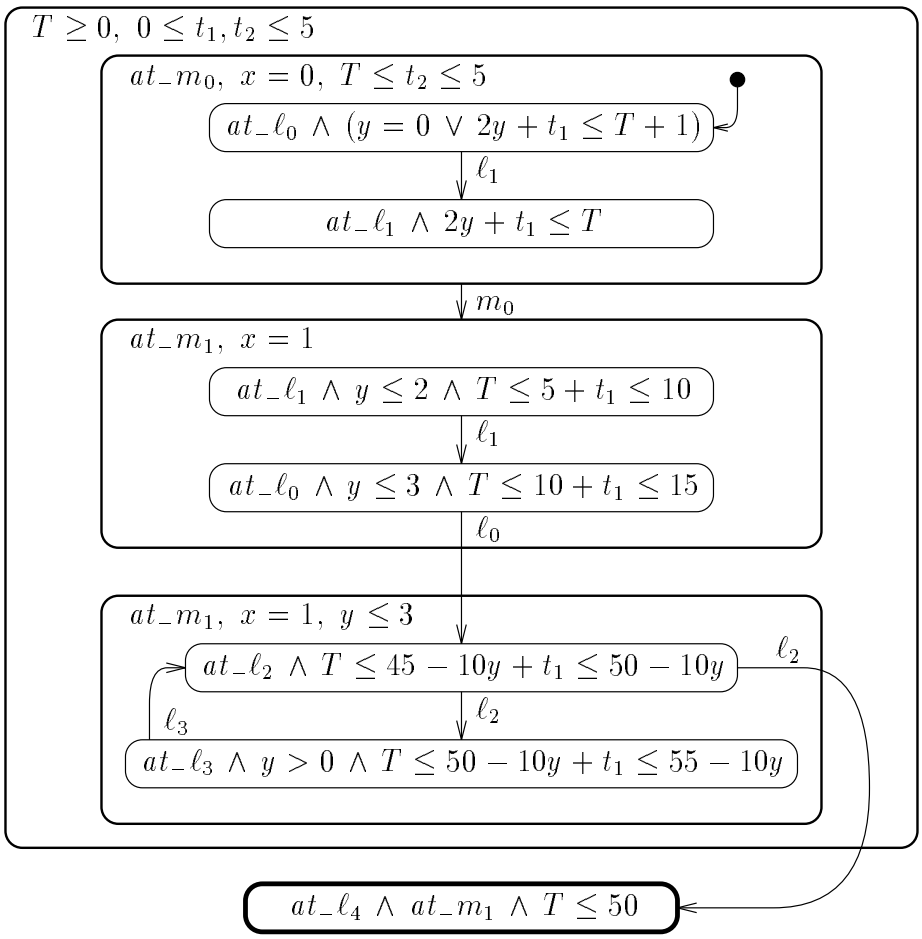

Fig. 14. A waiting diagram, establishing the formula
$(at_\ell_0 \wedge at_m_0 \wedge x = y = T = 0) \quad \Rightarrow \quad (T \leq 50)\, \mathcal{W}\, (at_\ell_4 \wedge at_m_1 \wedge T \leq 50)$

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 10, and establish the waiting-for formula

$$(at_\ell_0 \wedge at_m_0 \wedge x = y = T = 0) \quad \Rightarrow \quad (T \leq B)\, \mathcal{W}\, (at_\ell_4 \wedge at_m_1 \wedge T \leq B),$$

where

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

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

$$(at_\ell_0 \wedge at_m_0 \wedge x = y = T = 0) \quad \Rightarrow \quad \diamondsuit(at_\ell_4 \wedge at_m_1 \wedge T \leq B),$$

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→R. On the other hand, in Example 8, we established termination of UP-DOWN, using rule CB-WELL, in a proof that is valid for all $L \geq 0$. This illustrates the case of a response property that cannot be proven by rule W→R, but is provable by rule CB-WELL.

As the second answer justifying the introduction of rule CB-WELL, we propose to compare the verification diagram of Fig. 12 with that of Fig. 14, both establishing termination of UP-DOWN for the time bounds $[1, 5]$ (Fig. 12 actually established it for general $[L, U]$). It is obvious that diagram 14 require a much more detailed analysis of the precise time interval which we can spend at each of the diagram nodes. In comparison, the diagram of Fig. 12 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→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 $\diamondsuit$ 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.

Acknowledgment

We gratefully acknowledge the careful reading and critical comments of Oded Maler, Henny Sipma, and Tomás Uribe.

References

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

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

[AH92] R. Alur and T. Henzinger. Logics and models of real time: A survey. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lect. Notes in Comp. Sci.*, pages 74–106. Springer-Verlag, 1992.

[AH94] R. Alur and T.A. Henzinger. Real-time system = discrete system + clock variables. In T. Rus and C. Rattray, editors, *Theories and Experiences for*

Real-time System Development, AMAST Series in Computing 2, pages 1–29. World Scientific, 1994.

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

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

[HK94] T.A. Henzinger and P.W. Kopke. Verification methods for the divergent runs of clock systems. In H. Langmaack, W.-P. de Roever, and 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, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Inf. and Comp.*, 112(2):273–337, 1994.

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

[KMP95] Y. Kesten, Z. Manna, and A. Pnueli. Clocked transition systems. Technical report, Dept. of Comp. Sci., Stanford University, 1995.

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

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

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

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

[MP93] Z. Manna and A. Pnueli. Models for reactivity. *Acta Informatica*, 30:609–678, 1993.

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

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

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

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

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

[Sif91] J. Sifakis. An overview and synthesis on timed process algebra. In K.G. Larsen and A. Skou, editors, *3rd Computer Aided Verification Workshop*, volume 575 of *Lect. Notes in Comp. Sci.*, pages 376–398. Springer-Verlag, 1991.