An Interleaving Model for Real Time[1,2]

Thomas A. Henzinger
Department of Computer Science
Stanford University

Zohar Manna
Department of Computer Science
Stanford University
and
Department of Applied Mathematics
The Weizmann Institute of Science

Amir Pnueli
Department of Applied Mathematics
The Weizmann Institute of Science

July 11, 1990

Abstract. The interleaving model is both adequate and sufficiently abstract to allow for the practical specification and verification of many properties of concurrent systems. We incorporate real time into this model by defining the abstract notion of a real-time transition system as a conservative extension of traditional transition systems: qualitative fairness requirements are replaced (and superseded) by quantitative lower-bound and upper-bound real-time requirements for transitions.

We present proof rules to establish lower and upper real-time bounds for response properties of real-time transition systems. This proof system can be used to verify bounded-invariance and bounded-response properties, such as timely termination of shared-variables multi-process systems, whose semantics is defined in terms of real-time transition systems.

[1]This research was supported in part by an IBM graduate fellowship, by the National Science Foundation grants CCR-89-11512 and CCR-89-13641, by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under contract AFOSR-90-0057, and by the European Community ESPRIT Basic Research Action project 3096 (SPEC).

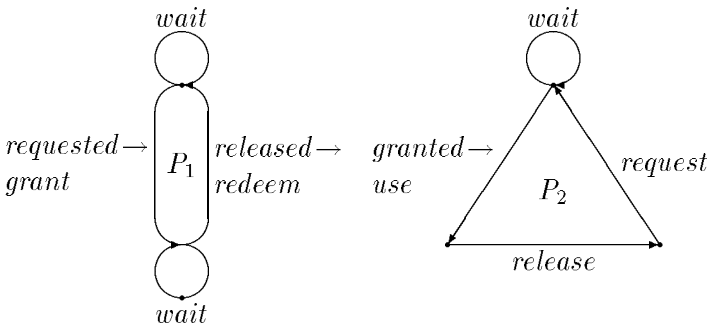
[2]An abbreviated version of this paper appears in the proceedings of the 5th *Jerusalem Conference on Information Technology* (1990).

1 Introduction

In order to develop and apply a formal methodology for the specification and verification of a class of systems, the members of the class have to be modeled by mathematical objects. Such a model should be both *adequate*, in that it distinguishes between systems whose behaviors differ in one of the aspects under consideration, and *abstract*, in that it omits unnecessary detail by identifying systems without such disagreements.

We study the class of *reactive* systems, which maintain an ongoing interaction with their environment. One well-established approach to the modeling of reactive systems uses the paradigm of *interleaving* to represent concurrent activity. Under this choice, the *linear* semantics of a system is the set of all possible behaviors, where each behavior is a (possibly infinite) sequence of states generated by performing the basic actions (transitions) of the system, one at a time. Concurrent actions are linearized; they may be performed in either order.

Consider, for example, the following concurrent system consisting of a resource allocator P_1 and a client process P_2:



Suppose that the client P_2 requests the resource (say, by setting a shared variable). Then P_2 waits until the resource is granted by P_1, at which point P_2 goes ahead and uses the resource. When it is finally released by P_2, the allocator P_1 retakes control of the resource and waits for another request.

If we consider *grant*, *redeem*, *request*, *use*, *release*, and *wait* to be atomic actions, some of the possible behaviors of this system are:

request grant use release redeem ...
request wait grant use release redeem ...
request wait wait grant use release redeem ...

To rule out shuffles of actions that are unfair to a particular transition, by preferring parallel transitions ad infinitum, the admissible behaviors of a system are constrained by *fairness* conditions. These conditions put, however, no restrictions on the relative speed of parallel processes within the system; they ensure only that every process will proceed "eventually."

In our resource allocation example, an appropriate fairness condition guarantees that whenever the resource is requested, it will eventually be granted; the fairness condition restricts the set of possible behaviors of the system by ruling out the behavior

$$request\ wait\ wait\ wait\ wait\ wait\ \ldots,$$

in which the client process waits forever for the resource to be granted.

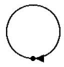
If the grain of atomicity of actions is chosen fine enough, this simple linear model turns out to be both adequate and convenient for the study of many qualitative properties of reactive systems — in particular the correctness of concurrent programs, independent of whether they are implemented in multiprogramming or multiprocessing environments. There are well-understood formal languages to specify the correctness properties of such systems, like linear temporal logic, as well as deductive and automatic methods for their verification ([Pn77], [OL82], [LP84], [MP89]).

Observe that the described model is abstract with respect to time; it identifies systems that admit the same sequences of actions, even if they do so at radically different speeds. This simplifies the treatment of speed-independent systems, while it is not adequate for *real-time* systems, whose correctness depends crucially on the actual times at which actions are performed. Many communication protocols and control circuits are examples of such systems.

Our goal is to refine the linear model to incorporate time, and to generalize the corresponding specification and verification methodology to enable the analysis of real-time systems.

For this purpose, we introduce a new process that represents a global, discrete clock:

tick



This *clock* process performs the action *tick* ad infinitum.

By adding the clock process to our resource allocation example, we obtain, as behaviors of the system, infinite sequences of interleaved actions of *three* processes — the allocator, the client, and the clock. This refined model allows us to put constraints on the times at which the actions of the original two processes happen.

For instance, a real-time resource allocator may be required to grant the resource, provided it is available, at most 2 time units after it is requested. Although, strictly speaking, we cannot specify this property within our model if we assume that actions take place in real (continuous) time, we can approximate it by requiring that between every request and corresponding grant of the resource there are at most 2 clock ticks. This real-time requirement constrains the set of admissible behaviors of the resource allocator, by ruling out behaviors such as

$$\begin{array}{l} \textit{request tick tick tick grant use tick release} \ldots \\ \textit{request tick wait tick tick wait tick grant} \ldots \end{array}$$

Thus we may view real-time requirements as *finitary* fairness requirements, which restrict the number of possible behaviors of a system.

Instead of adding the clock process explicitly to every system, we will define the notion of a *timed* behavior by associating times with the states of a behavior. The time of each state in a behavior is a natural number; it records the number of clock ticks that have happened until the state is reached. Thus, the time difference between two successive states may be 0, indicating that both states occur, in the given order, between successive clock ticks. All we require is fairness with respect to the clock process; that is, any possible behavior of a system contains infinitely many *tick* transitions.

In the following, we first develop the notion of a real-time transition system as a set of timed behaviors. To specify properties of such systems we use an extension of linear temporal logic by bounded temporal operators. Then we introduce proof rules for the verification of real-time properties with respect to a given system.

We present two very different verification styles to establish real-time properties. The first style resembles the proof-lattice technique used to show *liveness* properties of reactive systems; it uses a small set of basic rules and does not refer to the global clock explicitly. The second ("explicit-clock") proof style exploits the observation that when given access to the clock, every real-time property can be reformulated as a *safety* property; it uses the standard (timeless) temporal rules for establishing safety properties and relies heavily on invariances that include assertions about the global time.

2 Computational Model

We define the semantics of a shared-variables real-time system as a set of timed behaviors. This is done in two steps: first, we associate with any *concrete* shared-variables system an underlying *abstract* real-time transition system; secondly, we identify the possible timed behaviors (computations) of any real-time transition system. The latter step is presented first.

2.1 Abstract model: Real-time transition system

The basic computational model we use is that of a *transition system* ([MP89]), which we generalize by adding real-time requirements. We classify the real-time requirements into two categories: *lower-* and *upper-bound* requirements. They assure that transitions are taken neither too early nor too late, respectively.

A *real-time transition system* $S = \langle V, \Sigma, \Theta, \mathcal{T}, \mathcal{L}, \mathcal{U} \rangle$ consists of the following components:

- a finite set V of *variables*.
- a set Σ of *states*. Every state $\sigma \in \Sigma$ is an interpretation of V; that is, it assigns to every variable $u \in V$ a value $\sigma(u)$ in its domain.
- a set $\Theta \subseteq \Sigma$ of *initial states*.
- a finite set $\mathcal{T}$ of *transitions*, including the empty transition $\tau_\emptyset$ and the idle transition τ_I.

  Every transition $\tau \in \mathcal{T}$ is a binary accessibility relation on Σ; that is, it defines for every state $\sigma \in \Sigma$ a (possibly empty) set of τ-successors $\tau(\sigma) \subseteq \Sigma$. We say that τ is *enabled* on σ iff $\tau(\sigma) \neq \emptyset$; a set T of transitions is enabled on σ iff some transition in T is enabled on σ.

  The *empty* transition $\tau_\emptyset = \emptyset$ is not enabled on any state; the *idle* (stutter) transition
  $$\tau_I = \{(\sigma, \sigma) : \sigma \in \Sigma\}$$
  is enabled on every state.
- a finite set $\mathcal{L}$ of *lower-bound requirements*. Every lower-bound requirement $(\tau, T, l) \in \mathcal{L}$ for the transition $\tau \in \mathcal{T}$ contains, in addition to $\tau \neq \tau_I$, a set $T \subseteq \mathcal{T}$ of *trigger* transitions and a lower bound $l \in \mathsf{N}$.

- a finite set $\mathcal{U}$ of *upper-bound requirements.* Every upper-bound requirement $(\tau, u) \in \mathcal{U}$ for the transition $\tau \in \mathcal{T}$ contains, in addition to $\tau \neq \tau_I$, an upper bound $u \in \mathsf{N}^\infty$.[3]

A *timed state sequence* $\rho = (\sigma, \mathsf{T})$ consists of an infinite sequence σ of states $\sigma_i \in \Sigma$, $i \geq 0$, and an infinite sequence T of corresponding times $\mathsf{T}_i \in \mathsf{N}$, $i \geq 0$, that satisfy the following conditions:

- [*Initiality*] $\mathsf{T}_0 = 0$; that is, the time of the initial state is 0.

- [*Bounded monotonicity*] For all $i \geq 0$,

  either $\mathsf{T}_{i+1} = \mathsf{T}_i$,
  or $\mathsf{T}_{i+1} = \mathsf{T}_i + 1$ and $\sigma_{i+1} = \sigma_i$;

  that is, the time never decreases. It may increase, by at most 1, only between two consecutive states that are identical. The case that the time stays the same between two identical states is referred to as a *stuttering* step; the case that the time increases is called a *clock tick.*

- [*Progress*] For all $i \geq 0$ there is some $j > i$ such that $\mathsf{T}_i < \mathsf{T}_j$; that is, the time never stagnates. Thus there are infinitely many clock ticks.

The timed state sequence $\rho = (\sigma, \mathsf{T})$ is a *computation* (run) of the real-time transition system $S = \langle V, \Sigma, \Theta, \mathcal{T}, \mathcal{L}, \mathcal{U} \rangle$ iff it satisfies the following properties:

- [*Initiality*] $\sigma_0 \in \Theta$.

- [*Consecution*] For all $i \geq 0$ there is a transition $\tau \in \mathcal{T}$ such that $\sigma_{i+1} \in \tau(\sigma_i)$. We say that τ is *taken* at position i and *completed* at position $i+1$; a set T of transitions is taken (completed) at position j iff some transition in T is taken (completed) at j.

  The empty transition $\tau_\emptyset$ is assumed to be completed at position 0. At both stuttering steps and clock ticks, the idle transition τ_I may be taken.

- [*Lower bound*] Let a transition $\tau \in \mathcal{T}$ be *ready* at position i iff, for every lower-bound requirement $(\tau, T, l) \in \mathcal{L}$, there is no position j, $0 \leq j \leq i$, such that $\mathsf{T}_i < \mathsf{T}_j + l$ and T is completed at j. It follows

[3] Let $\mathsf{N}^\infty = \mathsf{N} \cup \{\infty\}$. For notational convenience, we assume that $m \leq n + \infty$ for all $m, n \in \mathsf{N}$.

that whenever a trigger transition from T is completed, τ is not ready for l time units.

The timed state sequence ρ satisfies the lower-bound property iff, for all $\tau \in \mathcal{T}$ and $i \geq 0$,

if τ is taken at position i,
then τ is ready at i;

that is, a transition can be taken only when it is ready (and enabled).

- [*Upper bound*] For every upper-bound requirement $(\tau, u) \in \mathcal{U}$ and all $i \geq 0$, there is some position $j \geq i$ with $\mathsf{T}_j \leq \mathsf{T}_i + u$ such that

  either τ is not ready at j,
  or τ is not enabled on σ_j,
  or τ is taken at j;

  that is, the transition τ cannot be continuously ready and enabled for u time units without being taken.

The set of computations of the system S is closed under stuttering: the addition or deletion of finitely many stuttering steps to a timed state sequence preserves the property of being a computation of S. We consider, however, all computations of S to be infinite; finite (terminating as well as deadlocking) computations can be represented by infinite extensions that add only clock ticks.

Also observe that while lower-bound requirements of the form $(\tau, T, 0)$ can be discarded without changing the computations of S, upper-bound requirements of the form (τ, ∞) add to S *weak-fairness* assumptions (in the sense of [MP89]).

2.2 Concrete model: Shared variables

The concrete real-time systems we consider consist of a fixed number of sequential programs that are executed in parallel, on separate processors, and communicate through a shared memory.

A *shared-variables multiprocessing system* P has the form

$$\{\theta\}[P_1 \| \ldots \| P_m].$$

Each *process* P_i, $1 \leq i \leq m$, is a sequential nondeterministic real-time program over the finite set U_i of *private* (local) *data variables*, and the finite set U_s of *shared data variables*. The formula θ, called the *data precondition* of P, restricts the initial values of the variables in $U = U_s \cup \bigcup_{1 \le i \le m} U_i$.

The real-time programs P_i can be alternatively presented in a textual programming language, or as transition diagrams. We shall use the latter, graphical, representation.

A *transition diagram* for the process P_i is a finite directed graph whose vertices $L^i = \{\ell^i_0, \ldots \ell^i_{n_i}\}$ are called *locations*; ℓ^i_0 is considered to be the *entry* location:

$$\longrightarrow \ell^i_0$$

The intended meaning of the entry location is that the control of the process P_i starts at the location ℓ^i_0 at time 0 (i.e., before the first clock tick).

Each edge in the graph is labeled by a guarded instruction, an *initial delay* $l \in \mathsf{N}$ and, optionally, a *delay increment* $u \in \mathsf{N}^\infty$:

$$\ell^i_j \xrightarrow[l + u^*]{c \;\rightarrow\; x := e} \ell^i_k$$

where c is a boolean expression, x a variable, and e an expression (the guard *true* and the initial delay 0 are usually suppressed; the instruction $c \;\rightarrow\; x := x$ is often abbreviated to c?).

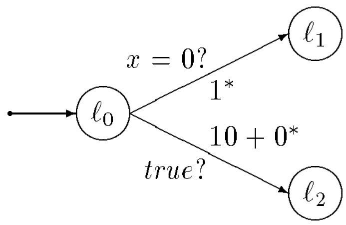
We say that the process P_i is *ready to proceed* from the location ℓ^i_j to the location ℓ^i_k iff its control has resided at ℓ^i_j for at least l time units (i.e., clock ticks). The intended operational meaning of the given edge is that whenever the process P_i is ready to proceed from ℓ^i_j to ℓ^i_k and the guard c is true, then P_i *may* proceed to ℓ^i_k. The delay increment u ensures that whenever the process P_i has been ready to proceed from ℓ^i_j to ℓ^i_k for u time units during which the guard c has been continuously true, then P_i *must* proceed to ℓ^i_k. In doing so, the control of P_i moves to the location ℓ^i_k "instantaneously," and the current value of e is assigned to x.

In other words, the execution of the given edge is first delayed for at least l time units, after which the guard c is repeatedly checked at least every u time units, until it is found to be true.

In general, a process may have been ready to proceed via several edges all of whose guards have been continuously true for their corresponding delay increments. In this case, any such edge is chosen nondeterministically.

We require that each cycle in a transition diagram contains an edge that is labeled either with a positive (nonzero) initial delay or a positive delay increment. This is because cycles that consume no time may prevent the time from progressing.

To demonstrate the scope of this model, we show how the typical real-time application of a *timeout* situation can be represented. Consider the process P with the following transition diagram:



When at the location ℓ_0, the process P attempts to proceed to the location ℓ_1 for 10 time units, by checking the value of x at least once every time unit. If the value of x is different from 0 at least once every time unit, then P may not succeed and has to proceed to the alternative location ℓ_2 at time 10 (note that the delay increment 0 ensures that the vacuous guard *true* is indeed checked after the initial delay of 10 time units).

This operational view of the concrete model can be captured by a simple translation. With the given shared-variables multiprocessing system P, we associate the following real-time transition system $S_P = \langle V, \Sigma, \Theta, \mathcal{T}, \mathcal{L}, \mathcal{U} \rangle$:

- $V = U \cup \{\pi_1, \ldots \pi_m\}$. Each *control variable* π_i, $1 \leq i \leq m$, ranges over the locations L^i of the corresponding process P_i.

- Σ contains all interpretations of V.

- $\Theta = \{\sigma \in \Sigma : \sigma \models \theta$, and $\sigma(\pi_i) = \ell_0^i$ for all $1 \leq i \leq m\}$.

- $\mathcal{T}$ contains, in addition to $\tau_\emptyset$ and τ_I, a transition τ_E for every edge E in the transition diagrams for $P_1, \ldots P_m$. If E connects the source location ℓ_j^i to the target location ℓ_k^i and is labeled by the instruction $c \;\rightarrow\; x := e$, then $\sigma' \in \tau_E(\sigma)$ iff

  - $\sigma(\pi_i) = \ell_j^i$ and $\sigma'(\pi_i) = \ell_k^i$,
  - $\sigma \models c$ and $\sigma'(x) = \sigma(e)$, and
  - $\sigma'(y) = \sigma(y)$ for all $y \in V - \{\pi_i, x\}$.

  If τ_E is uniquely determined by its source and target locations, we often write $\tau_{j \rightarrow k}^i$.

By $Pred(\tau_E)$ we denote the set of *syntactic-predecessor* transitions of τ_E; that is, all transitions $\tau_{E'}$ such that the target location of E' coincides with the source location of E. If the source location of E is an entry location, then $Pred(\tau_E) = \{\tau_\emptyset\}$.

- $\mathcal{L}$ contains a lower-bound requirement $(\tau_E, Pred(\tau_E), l)$ for every edge E labeled by the initial delay l.

- $\mathcal{U}$ contains an upper-bound requirement (τ_E, u) for every edge E labeled by the delay increment u.

This translation defines the set of possible computations of the concrete system P as a set of timed state sequences.

We remark that the translation is conservative over the untimed case. Suppose that the system P contains no delay labels (recall that, in this case, all initial delays are 0). Then the state components of the computations of S_P are precisely all the legal execution sequences of P, as defined in the interleaving model of concurrency ([MP89]).

If the delay increment ∞ is added to all edges of P, progress is guaranteed for every individual transition and, thus, for every process: no transition can be continuously enabled (and ready) without being taken. In this case, the computations of S_P correspond precisely to the execution sequences of P that are weakly fair with respect to every transition.

3 Specification Language

Having settled on our computational model, we need a sufficiently expressive language that is interpreted over timed state sequences in order to specify real-time systems. We distinguish between *state* formulas, which assert properties of individual states of a computation, and *temporal* formulas, which assert properties of entire computations.

3.1 State formulas

Given a real-time transition system $S = \langle V, \Sigma, \Theta, \mathcal{T}, \mathcal{L}, \mathcal{U} \rangle$, we assume a first-order language with equality that contains interpreted function and predicate symbols to express operations and relations on the domains of the variables in V. Formulas of this language are interpreted over the states in Σ, and called *state formulas*. If the state formula p is true in state σ, we say that σ is a p-state.

We use the following abbreviations for state formulas:

- The *starting* condition *start* holds precisely in the initial states Θ.
- For any transition $\tau \in \mathcal{T}$ and state formulas p and q, the *verification* condition $\{p\}\tau\{q\}$ asserts that if p is true of a state $\sigma \in \Sigma$, then q is true of all τ-successors of σ. For any set $T \subseteq \mathcal{T}$ of transitions, we write $\{p\}T\{q\}$ for the conjunction $\bigwedge_{\tau \in T} \Big(\{p\}\tau\{q\}\Big)$ of all individual verification conditions.

  Note that the special cases $\{p\}\tau_{\emptyset}\{q\}$ and $\{p\}\tau_I\{q\}$ are equivalent to *true* and $p \to q$, respectively.
- For any transition $\tau \in \mathcal{T}$, the conditions $\mathit{enabled}(\tau)$, $\mathit{ready}(\tau)$, and $\mathit{completed}(\tau)$ assert that τ is enabled, ready, and completed, respectively. For any set $T \subseteq \mathcal{T}$ of transitions, we write $\mathit{completed}(T)$ for the disjunction $\bigvee_{\tau \in T} \mathit{completed}(\tau)$.

  Note that $\mathit{enabled}(\tau_{\emptyset})$ and $\mathit{enabled}(\tau_I)$ are equivalent to *false* and *true*, respectively.

For the case that the real-time transition system S is associated with a shared-variables multiprocessing system P, it is easy to see that the starting condition, verification conditions, and enabling conditions can indeed be expressed by state formulas.

Suppose that P consists of m processes P_i, $1 \leq i \leq m$. Let $at(\ell^i_j)$ stand for $\pi_i = \ell^i_j$; that is, the control of the process P_i is at the location ℓ^i_j. If θ is the data precondition of P, then the starting condition *start* is equivalent to the state formula

$$\theta \wedge \Big(\bigwedge_{1 \leq i \leq m} at(\ell^i_0)\Big) \wedge \mathit{completed}(\tau_{\emptyset}).$$

Let $\tau \in \mathcal{T}$ be a transition of S, and E the corresponding edge in the transition diagram for P; assume that E connects the location ℓ^i_j to the location ℓ^i_k and is labeled by the instruction $c \;\to\; x := e$. Then, the enabling condition $\mathit{enabled}(\tau)$ is equivalent to the state formula

$$at(\ell^i_j) \wedge c,$$

and the verification condition $\{p\}\tau\{q\}$ is equivalent to

$$\left(\begin{array}{c} p \wedge \mathit{ready}(\tau) \wedge \mathit{enabled}(\tau) \wedge at(\ell^i_k)' \wedge \mathit{completed}(\tau)' \wedge \\ (x' = e) \wedge \bigwedge_{y \in V - \{\pi_i, x\}} (y' = y) \end{array}\right) \to q',$$

where q' is obtained from q by replacing every variable with its primed version (for example, $at(\ell_k^i)'$ stands for $\pi_i' = \ell_k^i$).

The reader may guess, correctly, that the two conditions $ready(\tau)$ and $completed(\tau)$ can, in general, not be expressed by state formulas of S, because their truth values in a state σ of a timed state sequence depend on the transitions preceding σ. This is the case even if the real-time transition system S originates from a shared-variables multiprocessing system. In Section 5, we will show how the notion of *state* can be extended, by encoding information about past transitions, to allow for the definition of the conditions $ready(\tau)$ and $completed(\tau)$ as state formulas.

In the following it suffices, however, to view both abbreviations $ready(\tau)$ and $completed(\tau)$, for any transition $\tau \in \mathcal{T}$, as primitive, nonrigid, *propositions* that satisfy certain axioms (the truth value of a *nonrigid*, or *flexible*, proposition is evaluated at a position of a timed state sequence; it may differ at two positions i and j of a sequence $\rho = (\sigma, \mathsf{T})$ even if $\sigma_i = \sigma_j$ and $\mathsf{T}_i = \mathsf{T}_j$).

The axiom **COMP** asserts that, at any position of a timed state sequence, precisely one transition has been completed:

$$\bigoplus_{\tau \in \mathcal{T}} completed(\tau)$$

(read the connective $\oplus$ as *exclusive-or*). The axiom schema **READY-INV** states that the condition $ready(\tau)$ is preserved by all transitions that do not trigger a lower-bound requirement for τ:

$$(ready(\tau) \wedge completed(\hat{\tau})') \;\rightarrow\; ready(\tau)'$$

if $\hat{\tau} \notin T$ for all $(\tau, T, l) \in \mathcal{L}$ (as with variables, we introduce a primed version for every proposition). These axioms turn out to be sufficient for our purpose.

Thus, whenever we speak of *state formulas*, we shall admit the propositions $ready(\tau)$ and $completed(\tau)$ (formally meaning *extended*-state formulas in the sense of Section 5). In particular, we will never talk about the truth value of a state formula with respect to a particular state, but only with respect to a particular position within a timed state sequence. For instance, we may say that the i-th state σ_i of the timed state sequence $\rho = (\sigma, \mathsf{T})$ is a $ready(\tau)$-state, meaning that the transition τ is ready at position i of ρ.

3.2 Temporal formulas

Temporal formulas are constructed from state formulas by boolean connectives and bounded temporal operators; they are interpreted over timed state sequences. In this paper, we are mostly interested in proving two important classes of real-time properties — bounded-response and bounded-invariance properties. Thus we restrict ourselves to three kinds of temporal formulas.

- A *bounded-response* property asserts that something will happen within a certain amount of time. A typical application of bounded response is to state an upper bound u on the termination of a system S: if started at time 0, then S is guaranteed to reach a final state no later than at time u.

  Formally, we express bounded-response properties by temporal formulas of the form

  $$p \;\Rightarrow\; \Diamond_{\leq u}\, q,$$

  for state formulas p and q and $u \in \mathsf{N}$. The formula $p \Rightarrow \Diamond_{\leq u}\, q$ is true over the timed state sequence $\rho = (\sigma, \mathsf{T})$ iff, for all $i \geq 0$,

  if σ_i is a p-state,
  then there is some q-state σ_j, $j \geq i$, such that $\mathsf{T}_j \leq \mathsf{T}_i + u$;

  that is, every p-state is followed by a q-state within time u.

- A *bounded-invariance* property asserts that something will hold continuously for a certain amount of time; it is often used to specify that something will not happen for a certain amount of time. A typical application of bounded invariance is to state a lower bound l on the termination of a system S: if started at time 0, then S will not reach a final state before time l.

  Formally, we express bounded-invariance properties by temporal formulas of the form

  $$p \;\Rightarrow\; \Box_{<l}\, q,$$

  for state formulas p and q and $l \in \mathsf{N}$. The formula $p \Rightarrow \Box_{<l}\, q$ is true over the timed state sequence $\rho = (\sigma, \mathsf{T})$ iff, for all $i \geq 0$ and $j \geq i$,

  if σ_i is a p-state and $\mathsf{T}_j < \mathsf{T}_i + l$,
  then σ_j is a q-state;

  that is, no p-state is followed by a $\neg q$-state within time less than l.

- To prove bounded-invariance properties, we sometimes need to be able to express a stronger assertion than bounded invariance, that a p-state can be followed by a $\neg q$-state only if *two* conditions are met: the time difference is at least l, and there is an intermediate r-state. This *bounded-unless* property is expressed by the temporal formula

  $$p \;\Rightarrow\; q\,\mathsf{U}_{\geq l}\,r,$$

  for state formulas p, q, and r, and $l \in \mathsf{N}$; it is true over the timed state sequence $\rho = (\sigma, \mathsf{T})$ iff, for all $i \geq 0$,

  if σ_i is a p-state,
  then either all subsequent σ_j, $j \geq i$, are q-states,
  or there is some r-state σ_j, $j \geq i$, such that $\mathsf{T}_j \geq \mathsf{T}_i + l$ and
  all intermediate σ_k, $i \leq k < j$, are q-states;

  that is, every p-state is followed by a (possibly infinite) sequence of q-states until there is an r-state, which cannot be closer than time l.

  It is not hard to see that the bounded-invariance formula $p \Rightarrow \Box_{<l}\, q$ is equivalent to the bounded-unless formula $p \Rightarrow q\,\mathsf{U}_{\geq l}\,(\neg q)$.

While temporal-logic aficionados will readily recognize the three classes of formulas we have introduced as time-bounded versions of conventional, composite, *invariance*, *response*, and *unless* formulas ([MP89]), for our purpose it suffices to consider them primitive (for a general addition of time-bounded operators to linear temporal logic, see [AH90]).

We say that a temporal formula is *S-valid* iff it is true over all computations of the real-time transition system S; for state formulas we do not distinguish between S-validity and (general) validity (i.e., truth under every interpretation). A proof rule is called *S-sound* iff the S-validity of all premises implies the S-validity of the conclusion.

Any S-sound rule can be used for verifying properties of the system S. Consider the following *bounded-invariance* rule **BD-INV**, which allows us to conclude the bounded-invariance formula $p \Rightarrow \Box_{<l}\, q$ from the bounded-unless formula $p \Rightarrow q\,\mathsf{U}_{\geq l}\, r$, for any state formulas p, q, and r:

$$\boxed{\textbf{BD-INV} \qquad \frac{p \;\Rightarrow\; q\,\mathsf{U}_{\geq l}\,r}{p \;\Rightarrow\; \Box_{<l}\,q}}$$

It is not hard to convince ourselves that this rule is S-sound for every real-time transition system S.

4 Verification rules

We show how to prove that a given deterministic real-time transition system $S = \langle V, \Sigma, \Theta, \mathcal{T}, \mathcal{L}, \mathcal{U} \rangle$ satisfies its specification. In particular, we present a deductive system to establish the S-validity of bounded-invariance and bounded-response properties. The proof rules fall into four categories: the *single-step* rules derive real-time properties that follow from a single real-time requirement, while the *transitivity*, *disjunction*, and *induction* rules combine real-time properties into more complicated ones.

4.1 Single-step rules

First, we present basic *single-step* rules, which establish bounded-invariance and bounded-response properties that are enforced by a single lower-bound or upper-bound requirement, respectively.

The *single-step lower-bound* rule, **SS-LB**, uses a lower-bound requirement $(\tau, T, l) \in \mathcal{L}$:

$$
\begin{array}{lll}
\textbf{SS-LB} & (1) & p \rightarrow \mathit{completed}(T) \\
 & (2) & p \rightarrow \varphi \\
 & (3) & \{\varphi\}\, \mathcal{T} - \tau\, \{\varphi\} \\
 & (4) & \varphi \rightarrow q \\
\hline
 & & p \;\Rightarrow\; \Box_{<l}\, q
\end{array}
$$

By $\mathcal{T} - \tau$ we denote the set difference $\mathcal{T} - \{\tau\}$. The state formula φ is called the *invariant* of the rule.

We point out that the rule **SS-LB** derives a *temporal* (bounded-invariance) formula from premises all of which are *state* formulas. Note that the premise (3) is always valid for the empty transition $\tau_\emptyset$ and the idle transition τ_I. This is because $\tau_\emptyset$ is never enabled, and τ_I preserves every invariant.

To see that the rule **SS-LB** is S-sound, suppose that the premises (1) through (4) are valid, and consider an arbitrary computation of S containing a p-state σ_i. By premise (1), some trigger transition from T is completed at position i; thus τ cannot be taken at any position $j \geq i$ within time less than l. ¿From the premises (2) and (3) it follows that φ holds at σ_i and all subsequent states until the transition τ is taken; hence φ holds in particular at all states within time less than l. Since φ implies q by premise (4), the given conclusion follows.

To demonstrate a typical application of the single-step lower-bound rule, consider the single-process system P with the data precondition $x = 0$ and the following transition diagram:

$$\{x = 0\} \quad \longrightarrow \ell_0 \xrightarrow[2+1^*]{x = 0?} \ell_1$$

The process P confirms that $x = 0$ and proceeds to the location ℓ_1. Because of the initial delay 2 of the transition $\tau_{0\to 1}$, the final location ℓ_1 cannot be reached before time 2. Since the transition $\tau_{0\to 1}$ has to be attempted at most 1 time unit after the initial delay, and x is guaranteed to be 0 at this point, the final location ℓ_1 must be reached by time 3.

Let us carry out a formal proof of this analysis. First we show the bounded-invariance property

$$\mathit{start} \;\Rightarrow\; \Box_{<2}\, \neg \mathit{at}(\ell_1);$$

that is, the final location ℓ_1 cannot be reached before time 2. Since S_P contains the lower-bound requirement $(\tau_{0\to 1}, \{\tau_\emptyset\}, 2)$, by **SS-LB** it suffices to show the premises (let the invariant φ be $\mathit{at}(\ell_0)$)

$$\begin{array}{ll} (1) & \mathit{start} \rightarrow \mathit{completed}(\tau_\emptyset), \\ (2) & \mathit{start} \rightarrow \mathit{at}(\ell_0), \\ (4) & \mathit{at}(\ell_0) \rightarrow \neg \mathit{at}(\ell_1), \end{array}$$

all of which are trivially valid.

The *single-step upper-bound* rule, **SS-UB**, uses an upper-bound requirement $(\tau, u) \in \mathcal{U}$:

$$\boxed{\begin{array}{lll} \textbf{SS-UB} & (1) & p \rightarrow (\varphi \vee q) \\ & (2) & \varphi \rightarrow \mathit{ready}(\tau) \\ & (3) & \varphi \rightarrow \mathit{enabled}(\tau) \\ & (4) & \{\varphi\}\, \mathcal{T} - \tau\, \{\varphi \vee q\} \\ & (5) & \{\varphi\}\, \tau\, \{q\} \\ \hline & & p \Rightarrow \Diamond_{\leq u}\, q \end{array}}$$

This rule derives a *temporal* bounded-response formula from premises all of which are *state* formulas. The state formula φ is called the *invariant* of the rule. The premise (4) is always valid for the empty transition $\tau_\emptyset$ and the idle transition τ_I.

To see that the rule **SS-UB** is S-sound, suppose that the premises (1) through (5) are valid, and consider an arbitrary computation of S containing a p-state σ_i. ¿From the premises (1), (4), and (5) it follows that φ holds at σ_i and all subsequent states until a q-state is reached. Hence, the premises (2) and (3) imply that τ is continuously *ready* and *enabled* until a q-state is reached. Thus either a q-state is reached within time u, or the transition τ is taken within time u, which by premise (5) results again in a q-state within time u. The desired conclusion follows.

Consider again the single-process system P from above; we show the bounded-response property

$$(\mathit{ready\text{-}at}(\tau_{0\to 1}) \wedge x = 0) \;\Rightarrow\; \Diamond_{\leq 1}\, \mathit{at}(\ell_1), \qquad (\dagger)$$

where the abbreviation $\mathit{ready\text{-}at}(\tau_{j\to k})$ stands for the state formula

$$\mathit{at}(\ell_j) \wedge \mathit{ready}(\tau_{j\to k}).$$

Since S_P contains the upper-bound requirement $(\tau_{0\to 1}, 1)$, by **SS-UB** it suffices to show the premises (let the invariant φ be p)

$$\begin{array}{ll}
(2) & (\mathit{ready\text{-}at}(\tau_{0\to 1}) \wedge x = 0) \;\to\; \mathit{ready}(\tau_{0\to 1}), \\
(3) & (\mathit{ready\text{-}at}(\tau_{0\to 1}) \wedge x = 0) \;\to\; \mathit{enabled}(\tau_{0\to 1}), \\
(5) & \{\mathit{ready\text{-}at}(\tau_{0\to 1}) \wedge x = 0\}\; \tau_{0\to 1}\; \{\mathit{at}(\ell_1)\},
\end{array}$$

all of which are easily derived.

Bounded-response properties about the readiness of transitions follow from *lower-bound* requirements. Suppose that $(\tau, T, l) \in \mathcal{L}$ is the only lower-bound requirement for the transition τ; then:

$$\boxed{\begin{array}{lll}
\textbf{READY} & (1) & p \to q \\
 & (2) & p \;\Rightarrow\; \Box_{<l}\, q \\
 & (3) & q \to \neg \mathit{enabled}(T) \\
\hline
 & & p \;\Rightarrow\; \Diamond_{\leq l}\, (q \wedge \mathit{ready}(\tau))
\end{array}}$$

The premises (1) and (3) are state formulas; the premise (2) can be established using the single-step lower-bound rule.

To see that the rule **READY** is S-sound, suppose that the premises (1) through (3) are S-valid, and consider an arbitrary computation of S containing a p-state σ_i. ¿From the premises it follows that q holds and no trigger transition from T is enabled, and hence taken, at σ_i and all subsequent states within time less than l. Thus τ becomes ready by the first state σ_j with $j \geq i$ that is not within time less than l of σ_i. If $l = 0$, then $\sigma_j = \sigma_i$ is a q-state by premise (1). Otherwise, a clock tick is completed at position j, in which case σ_j is identical to its predecessor, and thus a q-state by premise (2).

In our example, $(\tau_{0\to 1}, \{\tau_\emptyset\}, 2)$ is the only lower-bound requirement for the transition $\tau_{0\to 1}$. Therefore we can use the rule **READY** to establish

$$start \;\Rightarrow\; \Diamond_{\leq 2}\,(\mathit{ready\text{-}at}(\tau_{0\to 1}) \,\wedge\, x = 0) \qquad (\ddagger)$$

from the premises

$$\begin{array}{ll} (1) & start \;\to\; (at(\ell_0) \,\wedge\, x = 0), \\ (2) & start \;\Rightarrow\; \Box_{<2}\,(at(\ell_0) \,\wedge\, x = 0), \\ (3) & at(\ell_0) \;\to\; \neg enabled(\tau_\emptyset). \end{array}$$

The state formulas (1) and (3) are trivially valid; the bounded-invariance formula (2) can be derived by the single-step lower-bound rule **SS-LB**, using the lower-bound requirement $(\tau_{0\to 1}, \{\tau_\emptyset\}, 2)$ and the invariant $at(\ell_0) \wedge x = 0$.

Next, we present a rule that allows us to prove bounded-response properties that result from combining a finite number of successive bounded-response properties — the *transitive upper-bound* rule:

$$\boxed{\begin{array}{lll} \textbf{TRANS-UB} & (1) & p \Rightarrow \Diamond_{\leq u_1} r \\ & (2) & r \Rightarrow \Diamond_{\leq u_2} q \\ \hline & & p \Rightarrow \Diamond_{\leq u_1 + u_2} q \end{array}}$$

It is not hard to see that this rule is S-sound for every real-time transition system S.

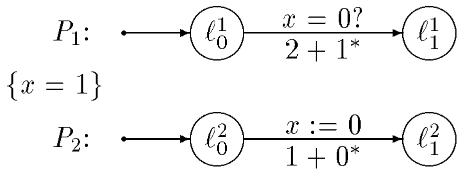
In our example, we use the transitive upper-bound rule **TRANS-UB** to combine the two properties ($\ddagger$) and ($\dagger$). Thus we conclude, at last, the bounded-response property that the final location ℓ_1 is reached by time 3:

$$start \;\Rightarrow\; \Diamond_{\leq 3}\, at(\ell_1).$$

4.2 Multiple processes

So far we have only examined a single-process example. In general, several processes that communicate through shared variables interfere with each other.

Consider the two-process system with the data precondition $x = 1$ and the following transition diagrams:



The first process, P_1, is identical to our previous example; after an initial delay of 2 time units, it confirms that $x = 0$ and proceeds to location ℓ_1^1. However, this time the value of x is not 0 from the very beginning, but set to 0 by the second process, P_2, only at time 1 (note that due to the initial delay 1 and the delay increment 0, the value of x is set to 0 exactly at time 1). Since P_1 does not check the value of x before time 2, it is guaranteed to find the value of x to be 0. Thus, P_1 reaches its final location ℓ_1^1 again at the earliest at time 2 and at the latest at time 3.

Let us conduct a formal proof. First we show the bounded-invariance property

$$\mathit{start} \;\Rightarrow\; \Box_{<2}\, \neg at(\ell_1^1);$$

that is, P_1 does not terminate before time 2. The proof proceeds as in the case of a single process, using the single-step lower-bound rule **SS-LB** and the lower-bound requirement $(\tau^1_{0\to1}, \{\tau_\emptyset\}, 2)$ (take the invariant $at(\ell_0^1)$). Since the new process P_2 contributes the transition $\tau^2_{0\to1}$, we have to establish an additional *noninterference* premise,

$$(3) \qquad \{at(\ell_0^1)\}\ \tau^2_{0\to1}\ \{at(\ell_0^1)\},$$

which is easily derived.

To prove the corresponding upper bound on termination in the two-process case, we need a stronger rule than the transitive upper-bound rule **TRANS-UB**. The rule **OVERLAP** allows us to prove bounded-response properties that result from combining a finite number of *parallel* (overlapping) bounded-response properties:

$$\begin{array}{lll}
\textbf{OVERLAP} & (1) & p \;\Rightarrow\; \Diamond_{\leq u_1}\, r \\
 & (2) & p \;\Rightarrow\; \Diamond_{\leq u_2}\, s \\
 & (3) & \{r\}\ \mathcal{T}\ \{r \vee q\} \\
 & (4) & \{s\}\ \mathcal{T}\ \{s \vee q\} \\
 & (5) & (r \wedge s) \;\Rightarrow\; \Diamond_{\leq u_3}\, q \\
\hline
 & & p \;\Rightarrow\; \Diamond_{\leq max(u_1,u_2)+u_3}\, q
\end{array}$$

Note that the premises (3) and (4) are always S-valid for the empty transition $\tau_\emptyset$ and the idle transition τ_I.

To see that the rule **OVERLAP** is S-sound, suppose that the premises (1) through (5) are S-valid, and consider an arbitrary computation of S containing a p-state σ_i. By the premises (1) and (2), σ_i is followed by an r-state σ_j within time u_1, and an s-state σ_k within time u_2. Without loss of generality we assume that $j \leq k$. Because of the premise (3), either there is a q-state within time u_2, thus implying the desired conclusion, or both r and s hold at σ_k. In the latter case, it follows from the premise (5) that there is a q-state within time $u_2 + u_3$, which again implies the conclusion of the rule.

In our example, we use the rule **OVERLAP** to establish the bounded-response property that P_1 terminates within 3 time units:

$$start \;\Rightarrow\; \Diamond_{\leq 3}\, at(\ell_1^1).$$

It suffices to show the premises

$$\begin{array}{ll}
(1) & start \;\Rightarrow\; \Diamond_{\leq 2}\, \mathit{ready\text{-}at}(\tau^1_{0\to 1}),\\
(2) & start \;\Rightarrow\; \Diamond_{\leq 1}(x=0),\\
(3) & \{\mathit{ready\text{-}at}(\tau^1_{0\to 1})\}\ \tau^1_{0\to 1}\ \{at(\ell_1^1)\},\\
(3') & \{\mathit{ready\text{-}at}(\tau^1_{0\to 1})\}\ \tau^2_{0\to 1}\ \{\mathit{ready\text{-}at}(\tau^1_{0\to 1})\},\\
(4) & \{x=0\}\ \tau^1_{0\to 1}\ \{at(\ell_1^1)\},\\
(4') & \{x=0\}\ \tau^2_{0\to 1}\ \{x=0\},\\
(5) & (\mathit{ready\text{-}at}(\tau^1_{0\to 1}) \wedge x=0) \;\Rightarrow\; \Diamond_{\leq 1}\, at(\ell_1^1).
\end{array}$$

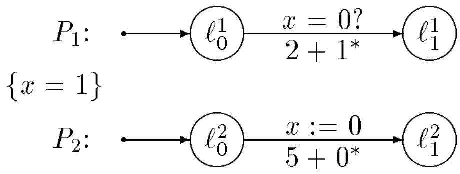
The premise (1) can be derived by the rule **READY** similarly to the corresponding property (‡) in the case of a single process. The premise (5) is identical to the property (†), only that the proof of the necessary new noninterference conditions requires the axiom **READY-INV**, as does the proof of the noninterference premise (3′). The noninterference premises (3), (4), and (4′) are trivially valid.

The essential difference between the single-process and the present system is manifested by the premise (2): while the precondition of the former already includes $x = 0$, in the current system $x = 0$ is established, “in time,” by the second process P_2. We show the premise (2) by an application of the transitive upper-bound rule **TRANS-UB** to the properties

$$\begin{array}{ll}
(2.1) & start \;\Rightarrow\; \Diamond_{\leq 1}\, \mathit{ready\text{-}at}(\tau^2_{0\to 1}),\\
(2.1) & \mathit{ready\text{-}at}(\tau^2_{0\to 1}) \;\Rightarrow\; \Diamond_{\leq 0}\,(x=0).
\end{array}$$

The first condition, property (2.1), is shown by single-step reasoning using the rules **READY** and **SS-LB** with respect to the lower-bound requirement $(\tau^2_{0\to 1}, \{\tau_\emptyset\}, 1)$; the second condition, (2.2), can be established by the rule **SS-UB**, employing the upper-bound requirement $(\tau^2_{0\to 1}, 0)$.

In the example we just considered, the transition $\tau^1_{0\to 1}$ becomes enabled, with the help of the process P_2, *before* it is ready. Now let us turn this situation around, and have $\tau^1_{0\to 1}$ be ready before it is enabled by P_2:



(The data precondition is again $x = 1$.)

In this new system, the second process P_2 sets x to 0 only at time 5, *after* process 1 has checked the value of x at least twice. Since the first process P_1 keeps testing whether $x = 0$ at least every time unit, it will reach its final location either at time 5 or at time 6.

The formal proof of the bounded-invariance property

$$start \;\Rightarrow\; \Box_{<5}\, \neg at(\ell^1_1),$$

that P_1 does not terminate before time 5, uses again the the single-step lower-bound rule **SS-LB**; however, this time the crucial lower-bound requirement is the one for $\tau^2_{0\to 1}$, namely $(\tau^2_{0\to 1}, \{\tau_\emptyset\}, 5)$. We need to show the premises

$$\begin{array}{ll}
(1) & start \rightarrow completed(\tau_\emptyset), \\
(2) & start \rightarrow (at(\ell^1_0) \wedge x = 1), \\
(3) & \{at(\ell^1_0) \wedge x = 1\}\ \tau^1_{0\to 1}\ \{at(\ell^1_0) \wedge x = 1\}, \\
(4) & (at(\ell^1_0) \wedge x = 1) \rightarrow \neg at(\ell^1_1),
\end{array}$$

all of which can be concluded easily.

The corresponding bounded-response property that P_1 terminates within 6 time units:

$$start \;\Rightarrow\; \Diamond_{\leq 6}\, at(\ell^1_1),$$

can be inferred by the rule **OVERLAP** just as in the previous example. All of the premises

$$
\begin{array}{ll}
(1) & \mathit{start} \;\Rightarrow\; \Diamond_{\leq 2}\, \mathit{ready\text{-}at}(\tau^1_{0\to 1}), \\
(2) & \mathit{start} \;\Rightarrow\; \Diamond_{\leq 5}(x = 0), \\
(3) & \{\mathit{ready\text{-}at}(\tau^1_{0\to 1})\}\ \tau^1_{0\to 1}\ \{\mathit{at}(\ell^1_1)\}, \\
(3') & \{\mathit{ready\text{-}at}(\tau^1_{0\to 1})\}\ \tau^2_{0\to 1}\ \{\mathit{ready\text{-}at}(\tau^1_{0\to 1})\}, \\
(4) & \{x = 0\}\ \tau^1_{0\to 1}\ \{\mathit{at}(\ell^1_1)\}, \\
(4') & \{x = 0\}\ \tau^2_{0\to 1}\ \{x = 0\}, \\
(5) & (\mathit{ready\text{-}at}(\tau^1_{0\to 1}) \wedge x = 0) \;\Rightarrow\; \Diamond_{\leq 1}\, \mathit{at}(\ell^1_1)
\end{array}
$$

can be derived as before.

4.3 Transitivity rules

After having looked at the parallel composition of transitions, we illustrate how to prove bounded-invariance and bounded-response properties of a chain of sequentially composed transitions. We use two *transitivity* rules to combine a finite number of nonoverlapping real-time properties.

The transitive upper-bound rule **TRANS-UB** has been given above; the *transitive lower-bound* rule **TRANS-LB** combines two bounded-unless properties:

$$
\boxed{
\begin{array}{lll}
\textbf{TRANS-LB} & (1) & p \Rightarrow q\ \mathsf{U}_{\geq l_1}\ r \\
 & (2) & r \Rightarrow q\ \mathsf{U}_{\geq l_2}\ s \\
\hline
 & & p \Rightarrow q\ \mathsf{U}_{\geq l_1 + l_2}\ s
\end{array}
}
$$

This rule is easily seen to be S-sound for every real-time transition system S.

Recall that from the conclusion of this rule we can infer the bounded-invariance property

$$p \Rightarrow \Box_{< l_1 + l_2}\, q$$

by an application of the bounded-invariance rule **BD-INV**. We shall often neglect to mention this simple proof step explicitly.

To establish the premises of the rule **TRANS-LB**, we need to strengthen the single-step lower-bound rule **SS-LB** to infer a bounded-unless property. For any lower-bound requirement $(\tau, T, l) \in \mathcal{L}$:

$$
\begin{array}{|lll|}
\hline
\textbf{SS-LBU} & (1) & p \rightarrow \mathit{completed}(T) \\
 & (2) & p \rightarrow \varphi \\
 & (3) & \{\varphi\}\, \mathcal{T} - \tau\, \{\varphi\} \\
 & (4) & \varphi \rightarrow q \\
 & (5) & \{\varphi\}\, \tau\, \{\varphi \vee r\} \\
\cline{2-3}
 & & p \;\Rightarrow\; q\, \mathsf{U}_{\geq l}\, r \\
\hline
\end{array}
$$

To see that the new, stronger, single-step lower-bound rule **SS-LBU** is S-sound, simply observe that we have added the premise (5) to the original rule **SS-LB**. This additional condition guarantees that the invariant φ — as well as q, due to the premise (4) — holds until an r-state is encountered (if ever). Thus the original rule **SS-LB** covers the special case in which the state formula r is chosen to be *true*.

We demonstrate the application of the transitivity rules by examining the single-process system P with the following transition diagram:

$$
\begin{tikzcd}
{} \arrow[r] & \ell_0 \arrow[r, "\mathit{true}?", "2+1^*"'] & \ell_1 \arrow[r, "\mathit{true}?", "2+1^*"'] & \ell_2
\end{tikzcd}
$$

We want to show that P terminates not before time 4 and not after time 6.

Given a transition diagram containing the location ℓ, let $\mathit{In}(\ell)$ be the set of transitions that correspond to incoming edges (i.e., edges whose target location is ℓ); if ℓ is an entry location, let $\mathit{In}(\ell) = \{\tau_\emptyset\}$. In particular, $\tau_I \notin \mathit{In}(\ell)$ for any location ℓ. We introduce the abbreviation $\mathit{enter}(l)$ to stand for the state formula

$$\mathit{at}(\ell) \wedge \mathit{completed}(\mathit{In}(\ell)).$$

For instance, $\mathit{enter}(\ell_1)$ stands, in our example, for $\mathit{at}(\ell_1) \wedge \mathit{completed}(\tau_{0\rightarrow 1})$.

Now let us derive the real-time bounds on the termination of P. First, we prove the lower bound

$$\mathit{start} \;\Rightarrow\; \Box_{<4}\, \neg \mathit{at}(\ell_2).$$

By the transitive lower-bound rule **TRANS-LB** (and a tacit application of **BD-INV**), it suffices to show the premises

$$
\begin{array}{ll}
(1) & \mathit{start} \;\Rightarrow\; (\neg \mathit{at}(\ell_2))\, \mathsf{U}_{\geq 2}\, \mathit{enter}(\ell_1), \\
(2) & \mathit{enter}(\ell_1) \;\Rightarrow\; (\neg \mathit{at}(\ell_2))\, \mathsf{U}_{\geq 2}\, \mathit{true}.
\end{array}
$$

Both of the premises can be established by the single-step lower-bound rule **SS-LBU**. To show the premise (1), we use the lower-bound requirement $(\tau_{0\to 1}, \{\tau_\emptyset\}, 2)$ and the invariant $at(\ell_0)$; the premise (2) follows from the lower-bound requirement $(\tau_{1\to 2}, \{\tau_{0\to 1}\}, 2)$ and the invariant $at(\ell_1)$.

The corresponding upper bound

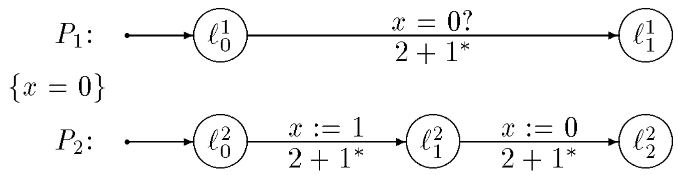
$$start \;\Rightarrow\; \Diamond_{\leq 6}\, at(\ell_2)$$

is derived by the transitive upper-bound rule **TRANS-UB**. It suffices to show the premises

$$\begin{array}{ll} (1) & start \;\Rightarrow\; \Diamond_{\leq 3}\, enter(\ell_1), \\ (2) & enter(\ell_1) \;\Rightarrow\; \Diamond_{\leq 3}\, at(\ell_2), \end{array}$$

both of which can be established by single-step upper-bound reasoning as demonstrated in the previous subsections.

A more interesting case involving transitive reasoning can be illustrated on the following two-process system with the data precondition $x = 0$:



In any analysis of this system, we have to distinguish two cases. The first process, P_1, may reach its final location ℓ_1^1 before the second process, P_2, sets the value of x to 1, or vice versa. In the latter case, P_1 has to wait until P_2 resets the value of x to 0. Consequently, P_1 may terminate as early as at time 2 or as late as at time 7.

The lower bound,

$$start \;\Rightarrow\; \Box_{<2}\, \neg at(\ell_1^1),$$

follows by single-step reasoning with respect to the lower-bound requirement $(\tau_{0\to 1}^1, \{\tau_\emptyset\}, 2)$. The upper bound,

$$start \;\Rightarrow\; \Diamond_{\leq 7}\, at(\ell_1^1),$$

is established by the rule **OVERLAP**. All of the premises

$$
\begin{array}{ll}
(1) & start \;\Rightarrow\; \Diamond_{\leq 2}\, ready\text{-}at(\tau^1_{0\to 1}), \\
(2) & start \;\Rightarrow\; \Diamond_{\leq 6}\, (at(\ell^2_2) \,\wedge\, x = 0), \\
(3) & \{ready\text{-}at(\tau^1_{0\to 1})\}\; \tau^1_{0\to 1}\; \{at(\ell^1_1)\}, \\
(3') & \{ready\text{-}at(\tau^1_{0\to 1})\}\; \tau^2_{0\to 1}\; \{ready\text{-}at(\tau^1_{0\to 1})\}, \\
(3'') & \{ready\text{-}at(\tau^1_{0\to 1})\}\; \tau^2_{1\to 2}\; \{ready\text{-}at(\tau^1_{0\to 1})\}, \\
(4) & \{at(\ell^2_2) \,\wedge\, x = 0\}\; \tau^1_{0\to 1}\; \{at(\ell^1_1)\}, \\
(4') & \{at(\ell^2_2) \,\wedge\, x = 0\}\; \tau^2_{0\to 1}\; \{at(\ell^2_2) \,\wedge\, x = 0\}, \\
(4'') & \{at(\ell^2_2) \,\wedge\, x = 0\}\; \tau^2_{1\to 2}\; \{at(\ell^2_2) \,\wedge\, x = 0\}, \\
(5) & (ready\text{-}at(\tau^1_{0\to 1}) \,\wedge\, at(\ell^2_2) \,\wedge\, x = 0) \;\Rightarrow\; \Diamond_{\leq 1}\, at(\ell^1_1),
\end{array}
$$

can be derived by single-step reasoning, except for (2), which follows by the transitive upper-bound rule **TRANS-UB** from the single-step upper bounds

$$start \;\Rightarrow\; \Diamond_{\leq 3}\, enter(\ell^2_1)$$

and

$$enter(\ell^2_1) \;\Rightarrow\; \Diamond_{\leq 3}\, (at(\ell^2_2) \,\wedge\, x = 0).$$

4.4 Disjunction rules

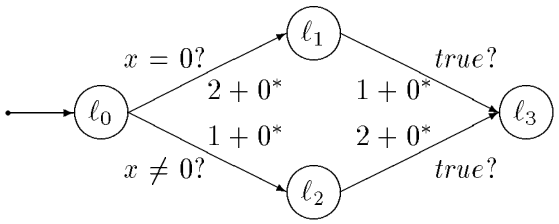
The simple transitivity rules are not powerful enough to handle programs with branching structures that are more complicated than trees; reasoning about confluent branches requires a case analysis. The *disjunctive lower-bound* rule **DIS-LB** and the *disjunctive upper-bound* rule **DIS-UB** provide the means for combining the parts of a case splitting:

$$
\begin{array}{lll}
\textbf{DIS-LB} & (1) & p_1 \;\Rightarrow\; q\, \mathsf{U}_{\geq l_1}\, r \\
 & (2) & p_2 \;\Rightarrow\; q\, \mathsf{U}_{\geq l_2}\, r \\
\hline
 & & (p_1 \vee p_2) \;\Rightarrow\; q\, \mathsf{U}_{\geq min(l_1,l_2)}\, r
\end{array}
$$

$$
\begin{array}{lll}
\textbf{DIS-UB} & (1) & p_1 \;\Rightarrow\; \Diamond_{\leq u_1}\, q \\
 & (2) & p_2 \;\Rightarrow\; \Diamond_{\leq u_2}\, q \\
\hline
 & & (p_1 \vee p_2) \;\Rightarrow\; \Diamond_{\leq max(u_1,u_2)}\, q
\end{array}
$$

Both disjunction rules are easily seen to be S-sound for every real-time transition system S.

For an application of the disjunction rules, consider the process P with the following transition diagram:



We show that P terminates at time 3, independently of the initial value of x.

Any proof considers two cases: either x is initially 0 and the transition $\tau_{0\to1}$ is taken, or x is initially different from 0 and the alternative transition $\tau_{0\to2}$ is taken. The lower bound,

$$start \;\Rightarrow\; \Box_{<3} \neg at(\ell_3),$$

follows by the disjunctive lower-bound rule **DIS-LB** from the premises

$$\begin{array}{ll} (1) & (start \wedge x = 0) \;\Rightarrow\; (\neg at(\ell_3)) \,\mathsf{U}_{\geq 3}\, true, \\ (2) & (start \wedge x \neq 0) \;\Rightarrow\; (\neg at(\ell_3)) \,\mathsf{U}_{\geq 3}\, true; \end{array}$$

the upper bound,

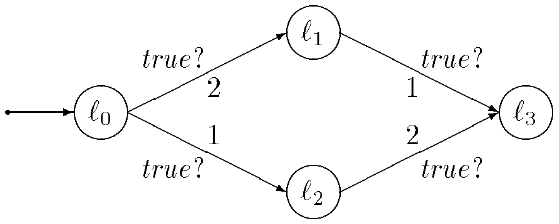
$$start \;\Rightarrow\; \Diamond_{\leq 3}\, at(\ell_3),$$

follows by the disjunctive upper-bound rule **DIS-UB** from the premises

$$\begin{array}{ll} (1) & (start \wedge x = 0) \;\Rightarrow\; \Diamond_{\leq 3}\, enter(\ell_1), \\ (2) & (start \wedge x \neq 0) \;\Rightarrow\; \Diamond_{\leq 3}\, enter(\ell_1). \end{array}$$

All four of the premises can be established by single-step and transitive reasoning as demonstrated in the previous subsections.

We remark at this point that our proof system is not strong enough to show tight bounds on *nondeterministic* systems. To see this, consider the following nondeterministic variant P' of the process P:



During an execution of P', one (if any) of the two transitions $\tau_{0\to 1}$ and $\tau_{0\to 2}$ is chosen nondeterministically (in general, a vertex of a transition diagram is *nondeterministic* iff any two guards that are associated with outgoing edges are not disjoint).

To show the lower bound

$$start \ \Rightarrow\ \Box_{<3}\, \neg at(\ell_3)$$

on the termination of P', we need to express and prove more complex temporal properties than are permitted in our simple specification language. For instance, the property that the control of P' resides at the location ℓ_0 either forever or until, no sooner than at time 2, it proceeds to the location ℓ_1 *or* until, no sooner than at time 1, it proceeds to the location ℓ_2, cannot be stated as a simple bounded-unless formula. On the other hand, in temporal logic with time-bounded operators this property can be expressed by the formula

$$start \ \Rightarrow\ \Big(at(\ell_0)\, \mathsf{U}_{\geq 2}\, enter(\ell_1) \ \vee\ at(\ell_0)\, \mathsf{U}_{\geq 1}\, enter(\ell_2)\Big).$$

An extension of our specification language to disjunctive properties of this form and the detailed treatment of nondeterminism is deferred to a separate paper ([HMP91]).

4.5 Induction rules

To prove lower and upper bounds on the execution time of program loops, we need to combine a state-dependent number of bounded-invariance or bounded-response properties. For this purpose we introduce two *induction* rules — the *inductive lower-bound* rule **IND-LB** and the *inductive upper-bound* rule **IND-UB**.

Assume that the variable $i \in V$ ranges over the natural numbers N; for any $n \in \mathsf{N}$:

$$\begin{array}{lll} \textbf{IND-LB} & (1) & p \rightarrow \varphi(n) \\ & (2) & (\varphi(i) \wedge i > 0) \Rightarrow q\, \mathsf{U}_{\geq l}\, \varphi(i-1) \\ & (3) & \varphi(0) \rightarrow r \\ \hline & & p \Rightarrow q\, \mathsf{U}_{\geq n \cdot l}\, r \end{array}$$

By $\varphi(i-1)$ we denote the state formula that results from the inductive invariant $\varphi(i)$ by replacing all occurrences of the variable i with the expression $i-1$; the formulas $\varphi(n)$ and $\varphi(0)$ are obtained analogously.

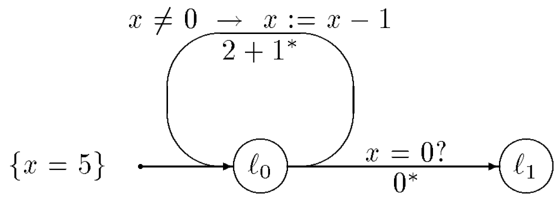
To see that the rule **IND-LB** is S-sound, for any system S, suppose that the premises (1) through (3) are S-valid, and consider an arbitrary computation of S containing a p-state σ_{k_n}. Then $\varphi(n)$ holds at σ_{k_n} by premise (1). ¿From the main premise (2) it follows that either q is true at σ_{k_n} and all subsequent states, or there is a sequence of positions $k_n \leq k_{n-1} \leq \ldots \leq k_0$ such that each σ_{k_i}, $0 \leq i \leq n$, is a $\varphi(i)$-state and q is true at all intermediate states σ_j, $k_n \leq j < k_0$; furthermore, the premise (2) implies that any two positions k_i and k_j, $i \neq j$, in this sequence are at least time l apart. In either case, the desired conclusion follows.

The inductive upper-bound rule uses again a variable $i \in V$ that ranges over the natural numbers N. For any $n \in \mathsf{N}$:

$$\boxed{\begin{array}{lll} \textbf{IND-UB} & (1) & p \rightarrow \varphi(n) \\ & (2) & (\varphi(i) \wedge i > 0) \Rightarrow \Diamond_{\leq u}\, \varphi(i-1) \\ & (3) & \varphi(0) \rightarrow q \\ \hline & & p \Rightarrow \Diamond_{\leq n \cdot u}\, q \end{array}}$$

It is not hard to convince ourselves that this rule is S-sound for every real-time transition system S as well.

We demonstrate the application of the induction rules by analyzing the single-process system P with the data precondition $x = 5$ and the following transition diagram:



The process P decrements the value of x until it is 0, at which point P proceeds to the location ℓ_1. Since x starts out with the value 5, and each decrement operation takes at least 2 and at most 3 time units, while the tests are instantaneous, the final location ℓ_1 is reached not before time 10 and not after time 15.

To prove the bounded-invariance property

$$start \;\Rightarrow\; \Box_{<10}\, \neg at(\ell_1),$$

we first apply the bounded-invariance rule **BD-INV**, showing instead

$$start \;\Rightarrow\; (\neg at(\ell_1))\, \mathsf{U}_{\geq 10}\, (enter(\ell_0) \wedge x = 0).$$

By the inductive lower-bound rule **IND-LB**, it suffices to show the premises

$$\begin{array}{ll} (1) & start \rightarrow (enter(\ell_0) \wedge x = 5), \\ (2) & (enter(\ell_0) \wedge x = i \wedge i > 0) \Rightarrow \\ & (\neg at(\ell_1)) \, \mathsf{U}_{\geq 2} \, (enter(\ell_0) \wedge x = i - 1). \end{array}$$

The first premise, (1), is trivially valid; the second premise, (2), follows from the single-step lower-bound rule **SS-LBU** with respect to the lower-bound requirement $(\tau_{0 \to 0}, \{\tau_\emptyset, \tau_{0 \to 0}\}, 2)$.

Now let us prove the bounded-response property

$$start \Rightarrow \Diamond_{\leq 15} \, at(\ell_1).$$

Applying the transitive upper-bound rule **TRANS-UB**, it suffices to show

$$\begin{array}{ll} (1) & start \Rightarrow \Diamond_{\leq 15} \, (enter(\ell_0) \wedge x = 0), \\ (2) & (enter(\ell_0) \wedge x = 0) \Rightarrow \Diamond_{\leq 0} \, at(\ell_1). \end{array}$$

The second premise, (2), can be concluded by single-step upper-bound reasoning with respect to the lower-bound requirement $(\tau_{0 \to 1}, \{\tau_\emptyset, \tau_{0 \to 0}\}, 0)$ and the upper-bound requirement $(\tau_{0 \to 1}, 0)$; we elaborate only on how the first premise, (1), can be derived by an application of the inductive upper-bound rule **IND-UB**. It suffices to show the premises

$$\begin{array}{ll} (1.1) & start \rightarrow (enter(\ell_0) \wedge x = 5), \\ (1.2) & (enter(\ell_0) \wedge x = i \wedge i > 0) \Rightarrow \Diamond_{\leq 3} \, (enter(\ell_0) \wedge x = i - 1). \end{array}$$

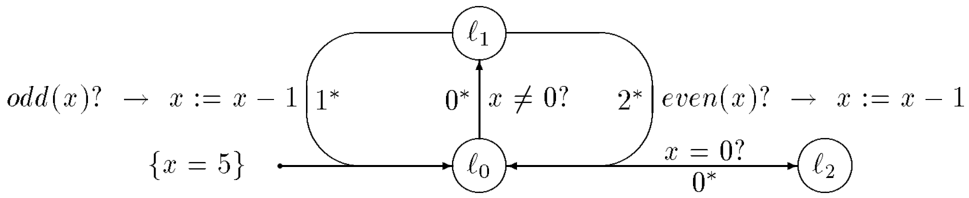
While the condition (1.1) is trivially valid, the main premise, (1.2), follows by single-step upper-bound reasoning from the lower-bound requirement $(\tau_{0 \to 0}, \{\tau_\emptyset, \tau_{0 \to 0}\}, 2)$ and the upper-bound requirement $(\tau_{0 \to 0}, 1)$.

The induction rules can be generalized, by letting the bounds l and u vary as functions of i. We state only the general inductive upper-bound rule, **IND-GUB**. For any $n \in \mathbb{N}$:

$$\begin{array}{|lll|} \hline \textbf{IND-GUB} & (1) & p \rightarrow \varphi(n) \\ & (2) & (\varphi(i) \wedge i > 0) \Rightarrow \Diamond_{\leq u_i} \, \varphi(i-1) \\ & (3) & \varphi(0) \rightarrow q \\ \cline{2-3} & & p \Rightarrow \Diamond_{\leq \Sigma_{0 < i \leq n} u_i} \, q \\ \hline \end{array}$$

This general rule is still S-sound for every real-time transition system S.

The general inductive upper-bound rule is needed to prove upper bounds for programs with loops whose execution time is state-dependent. Consider the following process P:



To show that P terminates within 7 time units, we apply the rule **IND-GUB** to the inductive invariant $enter(\ell_0) \wedge x = i$ and let u_i be 1 (2) if i is odd (even, respectively). The main premise,

$$(enter(\ell_0) \wedge x = i \wedge i > 0) \Rightarrow \Diamond_{\leq u_i}(enter(\ell_0) \wedge x = i - 1),$$

follows by the disjunctive upper-bound rule **DIS-UB** from the two conditions

$$(enter(\ell_0) \wedge x = i \wedge i > 0 \wedge odd(i)) \Rightarrow \Diamond_{\leq 1}(enter(\ell_0) \wedge x = i - 1)$$

and

$$(enter(\ell_0) \wedge x = i \wedge i > 0 \wedge even(i)) \Rightarrow \Diamond_{\leq 2}(enter(\ell_0) \wedge x = i - 1).$$

5 Explicit-Clock Reasoning

Consider a real-time transition system $S = \langle V, \Sigma, \Theta, \mathcal{T}, \mathcal{L}, \mathcal{U} \rangle$. The conditions $ready(\tau)$ and $completed(\tau)$ can be expressed by state formulas only if every state of S is situated in a computation and contains information about the history of the computation. To add this information, we define the extension of S by adding a clock variable whose value represents, in every state of a computation, the corresponding time.

We then proceed to show how this explicit access to the current time through the clock variable can be utilized by a real-time verification technique that differs substantially from the one presented in the previous section, in that it relies only on one simple inference rule.

5.1 Extended real-time transition systems

Let us introduce the following new variables:

- The *clock variable* $\mathtt{t}$ ranges over the natural numbers N; it records, in every state σ_i of a computation $\rho = (\sigma, \mathsf{T})$, the corresponding time T_i.

- The *transition variable* λ ranges over the transition set $\mathcal{T}$; it records, in every state of a computation, which transition has been completed in the preceding step.

- The *minimal-delay counters* δ_τ, $\tau \in \mathcal{T}$, range over N; they record, in every state of a computation, how many clock ticks must happen before the transition τ becomes ready.

- The *maximal-delay counters* Δ_τ, $\tau \in \mathcal{T}$, range over N^∞; they record, in every state of a computation, how many clock ticks may happen before the transition τ must be taken provided that it is continuously ready and enabled.

First-order formulas over this extended vocabulary are called *extended-state formulas.* The conditions $ready(\tau)$ and $completed(\tau)$ are obviously equivalent to the extended-state formulas $\delta_\tau = 0$ and $\lambda = \tau$, respectively.

The *extension* $S^* = \langle V^*, \Sigma^*, \Theta^*, \mathcal{T}^*, \mathcal{L}^*, \mathcal{U}^* \rangle$ of S is defined to be the following real-time transition system:

- $V^* = V \cup \{\mathbf{t}, \lambda\} \cup \{\delta_\tau : \tau \in \mathcal{T}\} \cup \{\Delta_\tau : \tau \in \mathcal{T}\}$.

- Σ^* contains all interpretations of V^*.

- Let $l(\tau, \hat{\tau})$ be the maximal l such that $(\tau, T, l) \in \mathcal{L}$ and $\hat{\tau} \in T$; if no such lower-bound requirement exists, let $l(\tau, \hat{\tau}) = 0$. Furthermore, let $u(\tau)$ be the minimal u such that $(\tau, u) \in \mathcal{U}$; $u(\tau) = \infty$ if no such upper-bound requirement exists.

  Θ^* contains all extensions σ^* of interpretations $\sigma \in \Theta$ such that, for all $\tau \in \mathcal{T}$,

  $$\begin{aligned} &\sigma^*(\mathbf{t}) = 0, \\ &\sigma^*(\lambda) = \tau_\emptyset, \\ &\sigma^*(\delta_\tau) = l(\tau, \tau_\emptyset), \\ &\sigma^*(\Delta_\tau) = u(\tau). \end{aligned}$$

- $\mathcal{T}^*$ contains, for every $\tau \in \mathcal{T}$, a transition τ^* such that $(\sigma_1^*, \sigma_2^*) \in \tau^*$ iff, for all $\tau' \in \mathcal{T}$,

  $$\begin{aligned} &(\sigma_1, \sigma_2) \in \tau, \\ &\sigma_1^*(\delta_\tau) = 0, \\ &\sigma_2^*(\mathbf{t}) = \sigma_1^*(\mathbf{t}), \\ &\sigma_2^*(\lambda) = \tau, \end{aligned}$$

$\sigma_2^*(\delta_{\tau'}) = max(l(\tau', \tau), \sigma_1^*(\delta_{\tau'})),$

$$\sigma_2^*(\Delta_{\tau'}) = \begin{cases} \sigma_1^*(\Delta_{\tau'}) & \text{if } \tau' \text{ is enabled on } \sigma_2 \text{ and } \sigma_2^*(\delta_{\tau'}) = 0 \\ u(\tau') & \text{otherwise.} \end{cases}$$

Note that the second clause, $\sigma_1^*(\delta_\tau) = 0$, enforces all lower bounds.

In addition, $\mathcal{T}^*$ contains the *tick* transition τ_T such that $(\sigma_1^*, \sigma_2^*) \in \tau_T$ iff, for all $\tau' \in \mathcal{T}$,

$\sigma_1 = \sigma_2,$
$\sigma_2^*(\mathbf{t}) = \sigma_1^*(\mathbf{t}) + 1,$
$\sigma_2^*(\lambda) = \tau_I,$
$\sigma_2^*(\delta_{\tau'}) = max(0, \sigma_1^*(\delta_{\tau'}) - 1),$

$$\sigma_2^*(\Delta_{\tau'}) = \begin{cases} \sigma_1^*(\Delta_{\tau'}) - 1 & \text{if } \tau' \text{ is enabled on } \sigma_1 \text{ and } \sigma_1^*(\delta_{\tau'}) = 0 \\ u(\tau') & \text{otherwise,} \end{cases}$$

$\sigma_2^*(\Delta_{\tau'}) \geq 0.$[4]

The last clause enforces all finite upper bounds.

- $\mathcal{L}^* = \emptyset$.
- $\mathcal{U}^*$ contains (τ^*, ∞) for all $(\tau, \infty) \in \mathcal{U}$. These remaining upper-bound requirements are weak-fairness conditions that enforce all infinite upper bounds.

The real-time transition system S and its extension S^* are equivalent in the following sense: for every computation $\rho = (\sigma, \mathsf{T})$ of S, there is a computation $\rho^* = (\sigma^*, \mathsf{T})$ of S^* such that every state σ_i^*, $i \geq 0$, is an extension of σ_i to V^* (and $\sigma_i^*(\mathbf{t}) = \mathsf{T}_i$ for all $i \geq 0$); and for every computation (σ^*, T) of S^*, the timed state sequence (σ, T) is a computation of S if every state σ_i, $i \geq 0$, is the restriction of σ_i^* to V.

Thus, to show a temporal formula ϕ (over V) to be S-valid, it suffices to show the S^*-validity of ϕ.

5.2 Explicit-clock verification

We point out that, once we are given explicit access to the global clock through the clock variable $\mathbf{t}$, both bounded-invariance and bounded-response properties over V can alternatively be formulated as (unbounded) unless properties over V^*.

[4] In evaluating expressions, let $\infty - 1 = \infty \geq 0$.

Unless properties assert that something will hold continuously either forever, or until terminated by the occurrence of another event. They are expressed by temporal formulas of the form

$$p \;\Rightarrow\; q \,\mathsf{U}\, r,$$

for state formulas p, q, and r. The formula $p \Rightarrow q \,\mathsf{U}\, r$ is true over the timed state sequence $\rho = (\sigma, \mathsf{T})$ iff, for all $i \geq 0$,

> if σ_i is a p-state,
> then either all subsequent σ_j, $j \geq i$, are q-states,
> or there is some r-state σ_j, $j \geq i$, such that all intermediate σ_k,
> $i \leq k < j$, are q-states;

that is, every p-state is followed by a (possibly infinite) sequence of q-states until there is an r-state.

In particular, the bounded-invariance property

$$p \;\Rightarrow\; \Box_{<l}\, q$$

is S^*-equivalent to the unbounded unless property

$$(p \wedge \mathbf{t} = T) \;\Rightarrow\; q \,\mathsf{U}\, (\mathbf{t} \geq T + l),$$

(i.e., both formulas are true over the same computations of S^*), and the bounded-response property

$$p \;\Rightarrow\; \Diamond_{\leq u}\, q$$

is S^*-equivalent to the unless property

$$(p \wedge \mathbf{t} = T) \;\Rightarrow\; (\mathbf{t} \leq T + u) \,\mathsf{U}\, q$$

if q is a state formula over V. Both unless formulas make use of the *rigid* (static) variable T (i.e., $T = T'$) to record the time of the p-state. Note that the latter equivalence is based on the fact that the time is guaranteed to reach and surpass $T + u$, for any value of T.

These observations lead to an alternative and quite different approach to the verification of real-time properties: to prove the S-validity of a real-time property ϕ (over V), we establish instead the S^*-validity of an S^*-equivalent unless property ϕ^* (over V^*). This can be done by applying the timeless *unless* rule:

$$\begin{array}{|llll|}\hline \textbf{UNLESS} & (1) & p \rightarrow (\varphi \vee r) & \\ & (2) & \{\varphi\}\, \mathcal{T}^* \, \{\varphi \vee r\} & \\ & (3) & \varphi \rightarrow q & \\ \cline{2-3} & & p \;\Rightarrow\; q \,\mathsf{U}\, r & \\ \hline \end{array}$$

We emphasize that all premises are state formulas; the state formula φ is called the *intermediate assertion* of the rule. It is easy to see that the unless rule is S^*-sound.

To demonstrate this kind of “explicit-clock” real-time reasoning, consider again the single-process system P with the data precondition $x = 0$ and the following transition diagram:

$$\{x = 0\} \qquad \longrightarrow \textcircled{\ell_0} \xrightarrow[2+1^*]{x = 0?} \textcircled{\ell_1}$$

Both the lower and the upper bound on the termination of P,

$$start \;\Rightarrow\; (\neg at(\ell_1))\,\mathsf{U}\,(\mathbf{t} \geq 2)$$

and

$$start \;\Rightarrow\; (\mathbf{t} \leq 3)\,\mathsf{U}\, at(\ell_1),$$

respectively, can be derived by the unless rule **UNLESS**. To establish the lower bound, we use the intermediate assertion

$$at(\ell_0) \wedge (0 \leq \mathbf{t} \leq 2) \wedge (\mathbf{t} + \delta_{\tau_{0\rightarrow 1}} = 2).$$

The upper bound follows from the intermediate assertion

$$\begin{gathered} at(\ell_0) \wedge (x = 0) \wedge (0 \leq \mathbf{t} \leq 3) \wedge \\ (0 \leq \delta_{\tau_{0\rightarrow 1}} \leq 2) \wedge (0 \leq \Delta_{\tau_{0\rightarrow 1}} \leq 1) \wedge (\mathbf{t} + \delta_{\tau_{0\rightarrow 1}} + \Delta_{\tau_{0\rightarrow 1}} = 3). \end{gathered}$$

While the verification style presented in Section 4 refers to time only through time-bounded temporal operators, explicit-clock reasoning uses ordinary, timeless, temporal operators and refers to the time in state formulas. Both styles trade off the complexity of the temporal proof structure against the complexity of the state invariants: the method of Section 4 relies on complex proof structures similar to the proof lattices used to establish ordinary (timeless) *liveness* properties ([OL82], [MP89]), and uses relatively simple invariants; the method of the present section employs only the simple unless rule — a *safety* rule —, but requires powerful intermediate assertions.

Acknowledgements. We thank Rajeev Alur for many helpful discussions.

Related work. There has been an increasing amount of literature on the formal analysis of real-time systems in recent years, and as the number of researchers has proliferated, so has the number of computational models.

We restrict ourselves to pointing to some of the work that builds on the timing model we have used — that of a global, discrete, and asynchronous clock: [Os90] uses an explicit clock variable for real-time reasoning, and includes many interesting applications; [Ko89] introduced the bounded temporal operators we use in our specification language; [Ha88] and [PH88] contrast the interleaving model with a synchronous model; [DW90] use finite automata for the specification (and synthesis) of real-time systems.

More theoretical accounts on specification languages for timed state sequences can be found in [AH89], [AH90], and [HLP90], all of which include methods for the automatic verification of *finite*-state real-time systems. [He90] gives a complete deductive system for a propositional real-time logic.

References

[AH89] R. Alur, T.A. Henzinger, "A really temporal logic," 30th IEEE FOCS, 1989.

[AH90] R. Alur, T.A. Henzinger, "Real-time logics: complexity and expressiveness," 5th IEEE LICS, 1990.

[DW90] D.L. Dill, H. Wong-Toi, "Synthesizing processes and schedulers from temporal specifications", 2nd Workshop on Computer-Aided Verification, 1990.

[Ha88] E. Harel, *Temporal Analysis of Real-time Systems*, M.S. Thesis, Weizmann Institute, 1988.

[He90] T.A. Henzinger, "Half-order modal logic: how to prove real-time properties," 9th ACM PODC, 1990.

[HLP90] E. Harel, O. Lichtenstein, A. Pnueli, "Explicit-clock temporal logic," 5th IEEE LICS, 1990.

[HMP91] T.A. Henzinger, Z. Manna, A. Pnueli, "Temporal proof methodologies for real-time systems," 18th ACM POPL, 1991.

[Ko89] R. Koymans, *Specifying Message Passing and Time-critical Systems with Temporal Logic*, Ph.D. Thesis, Eindhoven Univ. of Tech., 1989.

[LP84] O. Lichtenstein, A. Pnueli, "Checking that finite-state concurrent programs satisfy their linear specification," 11th ACM POPL, 1984.

[MP89] Z. Manna, A. Pnueli, "The anchored version of the temporal framework," *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency* (J.W. deBakker, W.-P. de Roever, and G. Rozenberg, eds.), Springer *LNCS* **354**, 1989.

[OL82] S. Owicki, L. Lamport, "Proving liveness properties of concurrent programs," ACM *TOPLAS* **4**, 1982.

[Os90] J.S. Ostroff, *Temporal Logic for Real-time Systems*, Research Studies Press, 1989.

[PH88] A. Pnueli, E. Harel, "Applications of temporal logic to the specification of real-time systems," Formal Techniques in Real-time and Fault-tolerant Systems, Springer *LNCS* **331**, 1988.

[Pn77] A. Pnueli, "The temporal logic of programs," 18th IEEE FOCS, 1977.