Verifying finite state real-time discrete event processes*

J.S. Ostroff
Computer Science Department, York University
4700 Keele Street, North York, Ontario, Canada, M3J 1P3.
Tel: 416 - 736-5053 email: jonathan@ynyetiti.bitnet

Abstract

A timed transition model (TTM) is an extension of the notion of a fair transition system developed by Manna and Pnueli. A TTM is an abstract computational model for the representation of a variety of concrete models of hard real-time systems (including synchronous or asynchronous message passing, shared variables and Petri net models). For any finite state TTM, this paper presents decision procedures for checking a small but useful class of safety and real-time response properties (specified in real-time temporal logic). The procedures are linear in the size of the system reachability graph.

Keywords: Real-time distributed systems, verification, decision procedures, real-time temporal logic.

1 Introduction

In hard real-time discrete event processes, system correctness depends not only on the logical result of the system behaviour, but also on the time at which the results are produced. Such systems are at the heart of many safety critical applications such as nuclear reactors, communication networks and avionics.

In this paper, we present decision procedures for checking a small but useful class of properties for any finite state system consisting of real-time discrete event processes. A timed transition model (TTM) is used for representing real-time discrete event processes, and real-time temporal logic (RTTL) is the assertion language in which the property to be verified is stated. TTMs and RTTL will be discussed in greater detail in the sequel; however, in brief, the following steps are involved in using the decision procedures:

1. Modelling step. Given a system $M$ composed of a set of interacting real-time discrete event processes, construct the corresponding TTM $M$.
2. Compute the reachability graph $G_M$ of $M$.
3. Specify a formula $S_M$ in RTTL that describes the behaviour that $M$ must satisfy.
4. Apply the appropriate decision procedure for $S_M$. All decision procedures perform an analysis of the reachability graph $G_M$.

An important feature of TTMs is that they can be used to represent a variety of concurrent or distributed models of computation. The modelling step will not be discussed in this paper. However, the reader is referred to [Pnu86, MP89, Ost88, OW87a, OW87b, Ost89] for more details and examples.

In the rest of this section the relationship of TTMs to other formalisms will be summarized. A complete definition of TTMs is given in section 2, and an overview of RTTL is given in section 3. The construction of reachability graphs is discussed in section 4, and the corresponding decision procedures are given in sections 5 and 6.

*This work is supported by the Natural Sciences and Engineering Research Council of Canada.

1.1 TTMs and RTTL

Timed transition models and real-time temporal logic extend the fair transition systems of Manna and Pnueli [MP83a, Pnu86, MP89] by adding a time metric. TTMs inherit all the features of fair transition systems. In addition, TTMs model hard real-time behaviour of concurrent systems. In particular, there is an external (possibly conceptual) global clock which is assumed to tick infinitely often. Transitions consist of an enabling condition, a transformation function and lower and upper time bounds. The lower and upper time bounds are measured with respect to the ticks of the clock, and can thus be used for modelling real-time properties (including delays and timeouts). Once a transition becomes enabled, it must occur within the lower and upper time bounds from the moment of enablement, unless it is preempted by the occurrence of some other transition which makes its enabling condition false.

In contrast to the RTTL proof system [Ost89] which can be used to verify arbitrary (possibly infinite state) systems, the decision procedures in this paper are limited to finite state systems. For simplicity, attention is focused on properties that follow from the time bounds (rather than fairness properties). The methods used in this paper may, however, also be used for properties resulting from fairness execution [Ost89]. The decision procedures (first described in [Ost89]) are based on the construction of a reachability graph, similar to those described in [MP83b], but with suitable changes for the real-time features of TTMs (see section 1.3 for more background).

1.2 Literature on automated real-time verification

The literature on mechanized verification of real-time properties is relatively sparse. One active research area is the analysis of time (or timed) Petri nets [MS76, Ram74, LS87]. For systems with finite state spaces, reachability graphs (similar to the reachability graphs presented in this paper) are constructed (e.g. [Zub86, RP84]). However, the focus is on performance evaluation (e.g. average queue lengths) rather than on hard real-time properties. Thus, there is no assertion language (such as RTTL) for specifying the hard real-time property to be verified. Where properties of real-time systems are checked automatically (e.g. [Men85]), there are usually restrictions on the class of Petri nets that can be checked (e.g. no state may have multi-enabled transitions). The restrictions usually stem from the interleaving model adopted in the construction of the state reachability graph.

Perhaps the closest work to ours is the use of modecharts as a graphical representation language [JS88], together with the first order logic RTL [JMB86] as a specification (or assertion) language. Although RTL is not more expressive than RTTL, certain kinds of timing properties can be stated more compactly in RTL than the corresponding specification in RTTL. On the other hand, RTL cannot easily deal with fairness properties, nor can it be used to represent data variables. A comparison of the complexity of deciding TTM properties with that of deciding modechart properties cannot be given, as complexity is not discussed in [JS88].

1.3 Literature on temporal logic model checking

Although there is very little literature on decision procedures for real-time temporal logic, there is a substantial body of results for (non
real-time) temporal logic. The idea of temporal logic model checking originated with Clarke, Emerson and others [EL85, CES86, EH86] (see their references for the original articles). For linear time temporal logic, procedures of polynomial complexity in the size of S and M was developed in [MP83c]. The procedures are applicable to a useful class of temporal logic specifications for finite state programs. An extension to [MP83c] is reported in [LP84], in which algorithms are supplied for checking the satisfiability of an arbitrary temporal specification over a finite state shared variables concurrent program. The algorithms are in the worst case exponential in the size of the specification formula, but quadratic in the size of the program. Since many safety and liveness properties can be expressed by temporal formulas of small size, it follows that in practice the proposed algorithms are efficient.

The authors of [EL85, CES86, EH86] argue that their CTL temporal logic is better than linear time temporal logic for model checking, i.e. for verifying that a finite state program satisfies its temporal specification. In particular, the branching time algorithm is linear in both the size of the program and the size of the property to be verified. Real-time properties are not treated in [EL85, CES86, EH86].

2 Timed transition models

Let M be any system composed of a set of interacting real-time discrete event processes. A TTM (timed transition model) M of M consists of a set of variables V, two distinguished variables e and n, a set of states S, a set of transitions T and a set of initial states θ. We now define the components of a TTM one by one.

V is the set of variables used to describe the processes of the system M. Usually each process contributes a control variable (used to indicate the current point of execution in the process) and a set of data variables. Each variable v ∈ V has an associated range type(v), and R is the union of all these types.

In addition to the variables in V, each TTM M has two distinguished variables: a clock variable t with type the set of natural numbers, and a next-transition variable n whose type is the set of transitions (transitions are defined below).

The clock variable represents the time on a (global) clock, external to the system M, and is used by an observer of the system to measure upper and lower time bounds of transitions.

The next-transition variable n is used to refer to the transitions of the system (as opposed to the variables in V which refer to the different states or activities of the system). ∆ is the set of all states of M. A state s is a mapping

\[ s: \{V\}/(n, t) \rightarrow (R_{\text{type}}(n), R_{\text{type}}(t)) \]

such that for each variable v in the domain we have that s(v) ∈ type(v).

Associated with each state s ∈ ∆ is a corresponding:

- state-assignment s*, which is the restriction of s to the domain (V\{n, t\}), (the set of all state-assignments is denoted by ∆*),
- state-map s#, which is the restriction of s to the domain V. The set of all state-maps is denoted ∆#.

T is the set of all transitions of M. A transition τ is a 4-tuple

\[ τ = (e, h, i, u) \]

where e is an enabling condition, h is a transformation function and i, u ∈ type(τ) are constants representing the (default) lower and upper time bounds respectively.

The enabling condition e is a boolean-valued expression in the variables of V. The transformation function h is a partial function h: ∆* → 2^n.

In addition to the transitions that derive from the events (or instructions) of M, there are in addition two distinguished transitions called the tick transition tick = (true, [t = t+1], φ, ∞), and the initial transition init = (true, [0], φ, θ). The tick transition must occur infinitely often, and at each tick the clock reading is incremented by one (all other variables remain the same).

As a brief illustration of the notion of a transition, consider a process of a real-time programming language with a delay statement such as

IF y EQ 10 THEN BEGIN delay(5); y := y + 1 END

occurring at some location in the process. The intended meaning of the statement is that if the local process variable y has a value of 10, then the process is delayed for 5 ticks of the clock after which y is incremented by one. The above delay instruction can be modelled by two transitions τ and τ' depicted graphically as:

\[ τ: y = 10 \rightarrow [y: y + 1] \quad \text{afterdelay} \]

\[ τ': y \neq 10 \rightarrow \text{skip} \]

Formally, τ is given by the 4-tuple

\[ (x = \text{adelay}; y := 10, [x: \text{afterdelay}, y: y + 1], 5, 8) \]

where x is a control variable and y represents the program variable y. If the assignment statement y := y + 1 takes longer than 3 ticks of the clock to execute, then the upper time bound is 8 ticks (i.e. 5+3).

The notation \( [t_1, t_2, t_3]\) is used for transformation functions (where \( t_1, t_2, t_3 \) ∈ V) meaning that on making the transition, the \( t_1 \) component of the state-assignment is replaced by the value of \( t_1 \) and \( t_2 \) by \( t_3 \) (all other components are left unchanged). The transition τ is given by \( (x = \text{adelay}; y := 10, [x: \text{afterdelay}, y: y + 1], 5, 8) \), assuming that the check on the guard \( y = 10 \) takes no longer than one clock tick.

θ is the set of initial states i.e. θ ⊆ ∆. Every s0 ∈ θ has s0(n) = initial. The set of initial state-maps θ# is defined as

\[ \theta# = \{ s0: s0 corresponds to a state s ∈ θ \} \]

3 Trajectories and real-time temporal logic

A trajectory σ is any infinite sequence s0s1s2... of states (si ∈ ∆). The set of all trajectories is denoted by ∆*. The set of all trajectories is denoted by ∆*.

The notation \( [s_0, s_1, s_2, \ldots] \) for a trajectory is often preferred as it allows us to picture a trajectory as a sequence of state-assignments with transitions taking us from one state-assignment to the next.

We can think of a trajectory as representing a possible behaviour or computation of a timed transition model of some system M. The sequence τ0τ1τ2... is an interleaving of transitions from the various processes of M.

Not all trajectories in ∆* represent actual behaviours of M. Some subset \( Σ \subseteq ∆* \) (called the legal trajectory set) will characterize the actual behaviour of M.

3.1 Initialized trajectories

Given a TTM M of a distributed system M, an initialized trajectory σ of M is any trajectory

\[ σ = s0s1s2... \]

satisfying the following requirements:

(a) initialization - s0 ∈ θ (thus s0(n) = initial), and initial never occurs again in σ.

(b) succession - For each i, if s_i[n] = τ (where transition τ has enabling condition e and transformation function h), then s_i(e) = true and \( h(s_i) \in δ(q_i) \).

(Notation: s_i(e) stands for the evaluation of e in the state s_i.)
(c) ticking - The clock always ticks sometime, i.e. there are an infinite number of states s in a such that s(\tau) = tick.

(d) upper time bound - Let \tau be any transition in T with finite upper time bound l \geq 0 and enabling condition e. For any i \leq l, let s_i(\exists t = T) = true (i.e. in state s_i, \tau is enabled and the clock variable reads T ticks of the clock), then there is some later state s_j in a (where j \geq i) such that

s_j((\tau \leq T + u) \land (\neg e \land s = r)) = true

Thus, \tau must occur within u ticks of the clock (from the time of its enablement), unless \tau is preempted by the occurrence of some other transition that causes the enabling condition of \tau to become false.

(e) lower time bound - Let \tau be any transition in T with a finite lower time bound l \geq 0 and enabling condition e. For any i < l, let s_j(\exists t = T) = true, and let s_k be the first state subsequent to s_i in a with s_k((T + l) = true (i.e. the clock has ticked l times). If

(i) either there is a predecessor state s_{i-1} (to s_i) for which

s_{i-1}(\neg e \land s = r) = true

(ii) or, s_i \notin \Theta,

then s_j(s \neq r) = true for all j = i, i+1, ..., k-1.

Conditions (i) and (ii) select states in trajectories from which the lower time bound is asserted (let's call these selected states choicepoints). Condition (i) asserts that a good choicepoint s_i is either when the transition first becomes enabled, or when the transition has just occurred and remains enabled. Condition (ii) asserts that any initial state is a good choicepoint.

One important cautionary note. A selfloop transition, which has both its upper and lower time bound equal to zero, must occur an infinite number of times before the clock ticks. Such zeroth transitions must therefore be disallowed if requirement (d) above is to be retained. In general, any sequence of zeroth transitions, where the sequence loops back on itself, must be disallowed. No such pathological conditions will occur for non-zeroth transitions.

The set of legal trajectories S_M \subseteq \Sigma^* of a timed transition model M consists of the set of all initialized trajectories of M and all suffixes of its initialized trajectories.

3.2 Real-time temporal logic

We give here a brief summary of real-time temporal logic and refer the reader to [OW87a, OW87b, Ost88] for more detail. Real-time temporal logic is based on Manna-Pnueli temporal logic with additional proof rules for real-time properties. However, no new temporal operators over and above the standard ones are needed (thus making the extension from temporal logic to real-time temporal logic straightforward).

A state-formula is any first order predicate which does not have any temporal operators. For example, n \neq r \lor r \leq 10 v \neq 0 \land a t \land l v = 30 is a state-formula that has the evaluation false in the state

\{(x, c, d) | y, (11, 1, 33), (a, r)\}

If a state-formula \phi evaluates to true in a state s then \phi satisfies s. Temporal operators (e.g. O (next time), U (henceforth), O (eventually), P (precedes) and U (until)), can be applied to state-formulas to construct temporal formulas. Unlike a state-formula which can be evaluated in a single state, a temporal-formula must be evaluated over a sequence of states.

As an example, the formula \exists u \exists w \phi holds if \phi holds for some future sequence of states. The formula \forall u \exists w \phi holds if \phi holds for all future sequences of states. The formula \exists u \forall w \phi holds if \phi holds for some future sequence of states that includes all future states.

\exists u \forall w \phi holds if there exists an i, such that \exists i \forall w

where \sigma^j is the i-shifted suffix of \sigma given by \sigma^j = si_{i+1} ... . If \phi is a state-formula, then \exists u \forall w \phi satisfies s.

The formula \forall u \exists w \phi is an abbreviation for \neg \exists u \forall w \phi holds true. Thus \forall u \exists w \phi holds if \phi holds for some future sequence of states. The formula \exists u \forall w \phi holds if \phi holds for all future sequences of states. The formula \exists u \forall w \phi holds if \phi holds for some future sequence of states that includes all future states.

3.3 Class of formulas considered

For simplicity, this paper provides procedures that decide the M-validity of specifications given by (a) invariance properties: \psi i \rightarrow \phi, and
(b) real-time responses: \psi n \land T \rightarrow \phi(\phi(t + l \leq t \leq T + l) ) for some constants l and u where \psi i and \psi n are state-formulas having no occurrences of the time variable t. However, the principles used in the above mentioned procedures can also be used to decide the validity of formulas involving the other temporal operators (see [Ost88] for full details). The invariance formula can be used to specify various safety properties (e.g. mutual exclusion) and the real-time response formula can be used to specify some kinds of liveness properties (e.g. process termination within a specified time deadline).

4 Finite state reachability graphs

If each variable in V ranges over a finite type and there are a finite number of transitions, then M has a finite number of state-maps (the number of states is always infinite because t ranges over the natural numbers).

A state s of M is reachable if it occurs in a legal trajectory of M, i.e. if there is some prefix of a legal trajectory that start with an initial state and ends in s. Similarly, if \psi is a state-map corresponding to the state s, then \psi is reachable if it is reachable. A reachability graph is the set of all state-maps reachable from a given state. An M-reachable state has a finite number of state-maps, then there is of course a guarantee that the reachability graph is finite.

Two different procedures (RG1 and RG2) are presented for constructing reachability graphs. RG1 is used if every transition has a lower time bound of zero; the construction of a reachability graph for a system is then similar to the construction of a reachability graph for a program in [MPS87]. In the absence of lower time bounds, any state that is reachable, may be reached "immediately" (i.e. before the clock ticks).

RG2 is used when there are some non-zero lower time bounds. Actually, RG2 can be used for any finite state system. However, where the lower bounds are zero, it is preferable to use RG1 (as it has no repeated state-maps and will thus produce smaller reachability graphs than RG2.) Once transitions have lower time bounds, then a state-map no longer uniquely characterizes the "state" of M in the sense that knowing the current state uniquely determines all future behaviour. In fact, a history must be maintained of when each transition was first enabled, so that it can be determined when the transition becomes eligible for execution. To keep track of eligibility, new nodes must be added to the reachability graph computed by RG1. Some nodes may also be removed from the reachability graph computed in RG1 as the timing constraints may help to render some nodes unreachable.

The algorithms for constructing the reachability graph are presented below together with an example that will illustrate all the timing considerations.
4.1 Lower bounds of zero — RG1

RG1 starts with the the set of initial state-maps, and then systematically explores all developments from the initial set.

Reachability Graph 1 — RG1: Given the set of initial state-maps $\Theta_0$, and the set of transitions $T$ of a system $S$ such that every transition $\tau \in T$ has $l_\tau = 0$, compute the directed graph $G_M = (Q, E)$ where $Q \subseteq \Theta_0$ is the set of all state-maps of the system that are reachable from the initial set $\Theta_0$, and $E$ is the set of all edges $(q_1, \tau, q_2)$ such that $\tau \in T$ and $q_1 \in q_2(q_2)$.

1. Initially $G_M = (\Theta_0, \emptyset)$, i.e. node$(G_M) = \Theta_0$ and edge$(G_M)$ is the empty set. All initial nodes are unmarked.

2. WHILE there is an unmarked node in node$(G_M)$ DO
   (a) Select an unmarked node $q$ in node$(G_M)$ and mark it.
   (b) FOR all transitions $\tau \in T$ (tick, initial]) that are enabled in $q$ (i.e. $q(e_\tau) = \text{true}$) DO for each successor $q' \in h_q(q)$
      i. IF $q'$ is not already in node$(G_M)$ THEN
         node$(G_M) :=$ node$(G_M) \cup \{q\}$
      ii. edges$(G) :=$ edges$(G) \cup \{(q, \tau, q')\}$

The marking scheme used in step 2 ensures that no state-map is visited more than once. Each state-map visited is marked (in step 2b(ii)) such that the only state-maps added to the list (of visited state-maps) are ones that have not previously been visited. Thus, as long as $M$ is finite state, RG1 will terminate.

Let $|S| \leq |\Theta_0|$, where $|\Theta_0|$ is the total number of state-maps. Usually, the reachable set is much smaller than the total set. In RG1, when a state-map is visited, only those edges that are enabled are added to the set of edges in 2b(ii). Thus, at each state-map there are at most $|T|$ transitions to be checked for successor states, where $|T|$ is the total number of transitions. Therefore, for each state-map, the reachability graph $G_M$ is constructed. Let $G_M = (N, E)$, i.e. $N =$ node$(G_M)$ and $E =$ edges$(G_M)$. Let $|N|$ be the number of nodes in $N$ and $|E|$ be the number of nodes in $E$. A procedure is said to be linear in the size of the graph $G_M$ if the procedure takes no more than $O(|N| + |E|)$ steps.

4.2 Arbitrary lower bounds — RG2

To illustrate the procedure of zero lower time bounds, consider the process in Figure 1 together with its transition table. If all the lower time bounds are zero, then procedure RG1 yields the reachability graph shown in Figure 2. The state-map $q_2$ (in which $r = \text{off}$) is reachable.

Consider the bounds indicated in Case 1 of the table in Figure 1. The transition $\tau$ has a lower and upper time bound of 2 ticks. All other transitions have a lower time bound of 0 and an upper time bound of 1 tick. If $\Theta_0 = \{q_0\}$, then the process may cycle in the component $(q_0, q_1)$ of the reachability graph shown in Figure 2 until the second tick of the clock. During this time $\tau$ cannot occur because of its lower time bound. Before the clock ticks a second time, the transition $\gamma$ must occur (it is enabled both in $q_0$ and $q_1$) with the result that the process lands in the state-map $q_2$. From $q_2$ there is no transition that leads to $q_0$, i.e. the case in which all the lower time bounds are zero, the process will never reach the activity off. The reachability graph computed via RG1 is therefore inaccurate, and the state-map $q_0$ may now be removed.

If all the time bounds are zero (call this case 0) then $q_0$ is reachable (but there is no guarantee that the process will eventually get there). For the bounds given in case 1, the state-map $q_0$ is never reached. For the bounds given in case 2, the state-map $q_0$ is not reachable but every trajectory of the process passes through $q_0$. Although RG1 correctly constructs the reachability graph for case 0, a more complicated procedure (RG2) is needed for cases 1 and 2.

We now present the procedure RG2. In Section 7, further explanation will be provided on how to use the reachability graphs computed by RG2.

It has already been noted that when the lower time bounds are non-zero the computation of the reachability graph must distinguish between those transitions that are enabled but not eligible for execution, and those that are eligible for execution because the default number of lower bound clock ticks has occurred since the transition was first enabled.

For each initial state-map $q \in \Theta_0$, the set of enabled transitions will be defined. This enabled set will then be partitioned into two sets: those transitions that are eligible for immediate execution, and those that are pending, i.e., those transitions that must be delayed from occurring until the lower bound number of clock ticks have occurred.

Let $q \in \Theta_0$. Then $\text{enabled}(q) \equiv \{(\tau, l, u), \tau \in T | q(e_\tau) = \text{true}\}$ and $\text{enabled}(q)$ can be partitioned as follows:

$$\text{enabled}(q) = \text{enabled}(q) \cup \text{pending}(q)$$

where

$$\text{enabled}(q) = \{(\tau, l, u), \tau \in \Theta_0 | q(e_\tau) = \text{true}\}$$

and

$$\text{pending}(q) = \{(\tau, l, u), \tau \in \text{enabled}(q) | l > 0\}$$

So far, enabled(q) has been defined only for initial state-maps. Let $q'$ be a successor of $q$, i.e. $q' \in h_q(q)$, and $(\tau', l', u') \in \text{enabled}(q)$. Assume enabled(q) is given. The following inductive definition may then be used to obtain enabled(q) from enabled(q):

$$\text{enabled}(q) \equiv \text{enabled}(q) \cup \text{pending}(q)$$

where

$$\text{enabled}(q) \equiv \{(\tau_1, l_1, u_1), \tau_1 \in \text{enabled}(q) | l_1 = 0\}$$

and

$$\text{pending}(q) \equiv \{(\tau_1, l_1, u_1), \tau_1 \in \text{enabled}(q) | l_1 > 0\}$$

$$\text{inherited}(q) \equiv \{(\tau_1, l_1, u_1), \tau_1 \in \text{pending}(q) | l_1 = 0\}$$

and

$$\text{newlyenabled}(q) \equiv \{(\tau, l, u), \tau \in \text{enabled}(q) | l = 0\}$$

The transitions that must happen prior to the next clock tick can be defined as $\text{mst}(q) \subseteq \text{enabled}(q)$ where

$$\text{mst}(q) \equiv \{(\tau, l, u), \tau \in \text{enabled}(q) | l = 0\}$$

The RG2 algorithm starts from the initial state-maps and computes successors as in RG1. A major difference is that the clock tick is included as one of the possible transitions. When the clock ticks, the lower and upper time bounds are decremented by one. If there are must transitions then the clock tick is removed as an eligible transition. Another major change must be made in step 2b(i) of RG1. It is no longer sufficient to check that the new state-map is equal to some already processed state-maps, as two equal state-maps may have different time histories on their bounds with the effect that the resulting behaviour may be different. Thus a new definition is needed for when two states may be treated as equivalent.

Two nodes $q$ and $q'$ are equivalent (written: $q \equiv q'$) if $q = q'$ and their enabled transitions have the same lower and upper time bounds, i.e.

$$q \equiv q' \equiv \{(q = q) \land \text{enabled}(q) = \text{enabled}(q')\}$$

The time bound histories are checked by the condition $\text{mst}(q) = \text{mst}(q')$.
$y = 0 \rightarrow \alpha[y : 1]$

$y \neq 2 \rightarrow \beta[y : 0]$

$y = 1 \rightarrow \beta[y : 0]$

<table>
<thead>
<tr>
<th>Name</th>
<th>Enabling Condition</th>
<th>Transition</th>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$z = \text{on} \land y = 0$</td>
<td>$[y : 1]$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$z = \text{on}$</td>
<td>$[y : 2]$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$z = \text{on} \land y = 1$</td>
<td>$[y : 0]$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\theta$</td>
<td>$z = \text{on} \land y \neq 2$</td>
<td>$[y : 0, z : \text{off}]$</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 1: A sample process

Figure 2: Reachability graph computed via RG1

Figure 3: Reachability graph for bounds of case 1
In the computation of successor nodes a new (unmarked) state-map is added to the list of nodes of the reachability graph only if there are no equivalent nodes already in the list. The RG2 procedure is described below.

RG2 - Reachability Graph 2: Given the set of initial state-assignment \( \Theta_0 \) and the set of all state-maps that are reachable from the initial set \( \Theta_0 \), compute the graph \( G = (Q, E) \) where \( Q \subseteq \Theta_0 \) is the set of all state-maps that are reachable from the initial set \( \Theta_0 \), and \( E \) is the set of all edges \((\tau, n, q)\) such that \( \tau \in T \) and \( p \in h_q(n) \).

1. Initially \( G = (\Theta_0, \emptyset) \), i.e. \( n \in \Theta_0 \) and \( \text{edges}(G) \) is the empty set. All initial nodes are unmarked.
2. WHILE there is an unmarked node \( q \in \text{nodes}(G) \) DO
   (a) Select any unmarked node \( q \in \text{nodes}(G) \) and mark \( q \).
   (b) IF \( \text{tick} \) ineqible (if some transition must occur)
       \( \text{IF } \text{must}(q) \neq \emptyset \text{ THEN } \text{eligible}(q) := \text{eligible}(q) - \{(\text{tick}, 0, \infty)\} \)
   (c) Compute all successors \( q' \) of \( q \)
       FOR all triples \((\tau, l, u) \in \text{eligible}(q) \) and for each successor \( q' \in h_q(n) \)
       i. IF \( \tau = \text{tick} \)
          \( \text{THEN decrement every finite } l, u > 0 \text{ in enabled}(q) \) by one, and compute \( \text{enabled}(q') \) from the decremented \( \text{enabled}(q) \) using equation (1).
       ELSE compute \( \text{enabled}(q') \) from equation (1).
       ii. Compute \( \text{eligible}(q') \) and \( \text{must}(q') \).
       iii. \( \text{IF } q' \text{ is not equivalent to any node in } \text{nodes}(G) \text{ THEN } \text{nodes}(G) := \text{nodes}(G) \cup \{q'\} \)
           iv. \( \text{edges}(G) := \text{edges}(G) \cup \{(q, \tau, q')\} \).

The resulting reachability graphs for Case 1 of the example process is shown in Figure 3. The pending transitions are shown with dotted lines. An edge labelled \(\gamma_i/\gamma_2\) means that the current upper time bound is \(\gamma_i\) and the current lower time bound is \(\gamma_2\).

Let \( \text{time}(G) \) be the largest upper time bound. Since a new state-map is added to the nodes of \( G \) each time the clock ticks until the upper time bound is reached (and the transition is forced to then occur), it follows that, in the worst case, each state-map may have up to \( \text{time}(G) \) duplicates of itself in the reachability graph. Therefore the reachability graph may take up to \( \text{time}(G) | S | \) steps to compute.

The RG2 procedure can be made more efficient by collapsing equal state-maps into one state-map (node). For example, step 2c(i) in RG2 may be replaced with the following:

2c(i)′ IF \( \tau = \text{tick} \)
   THEN IF \( \text{eligible}(q) = \text{must}(q) = \emptyset \)
       THEN \( m := \text{min}\{l | (\tau, l, u) \in \text{pending}(q) \}\),
       decrement every finite \( l, u > 0 \) \text{ in } enabled(q) \text{ by } m,
       and compute \( enabled(q') \) from the decremented \( enabled(q) \) using equation (1).
   ELSE compute \( enabled(q') \) from equation (1).

If \( \text{eligible}(q) = \text{must}(q) = \emptyset \), then the only transition that can occur is the clock tick as all other enabled transitions are pending. Instead of creating a new state-map \( q' \) for each tick of the clock, 2c(i)′ lets m ticks of the clock transpire before creating a new state. Since \( m \) is the least lower time bound (of transition \( \text{time}(\text{node}) \) say) in the pending set, it follows that after \( m \) clock ticks, the transition \( \text{time}(\text{node}) \) in the pending set becomes eligible, and the clock tick is no longer the sole transition that may occur. Further simplifications (such as 2c(i)′) of the reachability graph can be made, but then there will be a tradeoff between the complexity of RG2 and the size of \( G \).

4.3 Paths in reachability graphs

A path in a reachability graph \( G_M \) is any sequence \( q_0 \rightarrow q_2 \rightarrow q_3 \rightarrow \ldots \rightarrow q_n \) of nodes of \( G_M \) so long as all successive nodes \( q_i \) and \( q_{i+1} \) in the path satisfy \((q_i, \tau, q_{i+1}) \in \text{edges}(G_M) \) for some \( \tau \).

A path in a reachability graph can be seen as a "projection" of a legal trajectory. Formally, let \( \Psi = q_0 \rightarrow q_2 \rightarrow q_3 \rightarrow \ldots \rightarrow q_n \) be any legal trajectory of \( M \) and let \( q \) be the state-map corresponding to a state \( s_i \) of \( M \). Let \( q = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_n \rightarrow q_{n+1} \rightarrow \ldots \rightarrow q_{n+m} \rightarrow q_{n+m} \) be \( q \) (i.e. project out all duplicate state-maps). Duplicate state-maps arise because the time variable is not part of a state-map. If the tick transition transforms the state \( s \) into \( s' \), then \( s \neq s' \) (because the time components differ by one), but \( q = q' \) (because the corresponding state-maps do not have a time component).

We call \( \Psi \) a projection of the legal trajectory \( \Psi \). It follows that \( \Psi \) is a finite path of \( G_M \) iff \( \Psi \) is a proper prefix of some projection of a legal trajectory of \( M \).

Projections are never actually constructed, as their main purpose in the sequel is for analyzing the properties of paths in reachability graphs.

5 Decision procedure for invariances

Deciding an invariance property is straightforward. Consider the invariance property \( S \models \Psi_{\phi} \) where \( \Psi = (\phi \neq \emptyset) \) for the sample process in Figure 1. To check \( S \models \Psi \) for \( M \)-validity on \( G_M \), just visit each node \( q \) of \( G_M \) once and check that \( \phi(q) = \text{true} \).

Thus \( S \) is \( M \)-valid in case 0 (see reachability graph produced by RG1), because \( \phi(q) = \text{false} \), i.e. there is one reachable state \((q_0, q_0)\) for which the invariance fails to hold. For case 1 (see reachability graph produced by RG2), each node \( q \) of the reachability graph indeed satisfies \( \varphi \), i.e. \( \varphi(q) = \text{true} \) for each node \( q \), and thus \( S \) is \( M \)-valid.

In general, the invariance property \( \psi_1 \rightarrow \psi_2 \) of a TTM \( M \) can be decided with complexity linear in the size of the reachability graph \( G_M \).

Procedure 1 (Invariances)

Given the invariance property \( \psi_1 \rightarrow \psi_2 \) and the reachability graph \( G_M \). Decide if the invariance property is \( M \)-valid. Let \( \psi_1 \) stand for the set of all states satisfying \( \psi_1 \).

1. Initially set \( I := \text{nodes}(G_M) \) \( \setminus \psi_1 \), and assume all the state-maps in \( I \) are unmarked. (If \( I \) is the empty set then the property is trivially \( M \)-valid.)
2. WHILE there is an unmarked node in \( I \) DO
   (a) Select an unmarked state-map \( \psi \in I \) and mark it.
   (b) IF \( \psi_2 = \text{false} \) THEN fail and exit.
   3. Procedure succeeds.

Proposition: Procedure 1 succeeds iff \( \psi_1 \rightarrow \psi_2 \) is \( M \)-valid. (For proof see Ost88, Ost89.)

In step 2 of the invariance procedure, each node is visited and checked exactly once. Step 1 may also be computed by visiting each node \( q \) of \( G_M \) once, and adding \( q \) to \( I \) if \( \psi(q) = \text{false} \). It is therefore followed that the invariance property can be checked for \( M \)-validity with complexity linear in the size of \( G_M \).

6 Deciding real-time response

In this section, assume that all the reachability graphs are computed by RG1. The necessary modifications for using RG2 will be given in the next section.

We now define various subgraphs of \( G_M \) which will be useful for the decision procedures. If \( q \) is a node of \( G_M \), then \( G_M(q) \) is defined as the
subgraph of the reachability graph $G_M$ where $nodes(G_M(q))$ consists of $q$ together with all nodes of $G_M$ that are reachable from $q$, and where $edges(G_M(q))$ is the set of all edges $(q, r, q)$ in $edges(G_M)$ such that $q, q, q$ are nodes($G_M(q)$).

A finite path $q_1, \ldots, q_n$ in the reachability graph $G_M$ is said to be a non-$\psi$ path if none of $q_1, \ldots, q_{n-1}$ satisfy $\psi$. The state-map $q_n$ is allowed to satisfy $\psi$. The subgraph of $G_M$ formed from the set of all non-$\psi$ paths of $G_M(q)$.

The graph $G_M(q, \psi)$ can be constructed efficiently (linear in the size of $G_M$) as follows:

**Procedure 2** (construction of $G_M(q, \psi)$)

1. Initially the nodes of $G_M(q, \psi)$ is the singleton set $\{q\}$ and $q$ is unmarked.

2. WHILE there are still unmarked nodes in $G_M(q, \psi)$ DO
   (a) Select an unmarked node $q_i$ in the node set of $G_M(q, \psi)$ and mark it.
   (b) IF $q_i(\psi) = false$ THEN FOR each edge $(q_i, r, q_3)$ of $G_M$ add $q_3$ to the nodes of $G_M(q, \psi)$ (if not already in) and $(q_i, r, q_3)$ to the edges of $G_M(q, \psi)$.

3. **The dominant step is clearly 2(b)** because of the inner FOR loop. Because no node of $G_M$ is visited more than once, step 2(b) is reached no more than $|V|$ times, and the total number of iterations of the FOR loop is no more than $|E|$, the number of edges in the reachability graph (i.e., each edge is processed no more than once). Thus the total number of iterations of the inner loop in step 2(b) is no more than order $|V| |E|$.

4. Stated more formally we get: Procedure 2 for computing $G_M(q, \psi)$ is of complexity linear in the size of $G_M$. A simpler version of Procedure 2 (without the check in step 2(b) in the IF statement) can be used to construct $G_M(q, \psi)$ with linear complexity.

A graph such as the reachability graph $G_M = (N, E)$ is strongly connected if there is a path from each node in $N$ to any other node in $N$. A strongly connected component (SCC) of a digraph is a maximal strongly connected subgraph. A strongly connected component of a graph will be called terminal if no node in the component has an edge leaving the component. The strongly connected components of a digraph $(N, E)$ can be found in order of $|N + |E||$ steps (see [Meh84]).

Let $C_i$ be a SCC of $G_M$ and let $\tau$ be a transition with finite upper time bound. Then $\tau$ is called a progress edge if (a) for every state-map $q$ of $C_i$, $q(\tau) = true$, and (b) for each successor $q'$ in $h(q)$, $q'$ is not a node of the SCC $C_i$ (i.e., $\tau$ always leads away from $C_i$ to some other SCC).

By the definition of upper time bounds of legal trajectories, once the system lands in the SCC $C_i$, the progress edge or some other edge (leading out of $C_i$) must be taken within $u_\tau$ ticks of the clock.

When computing the strongly connected components $C_1, \ldots, C_m$ of a reachability graph $(N, E)$, a new graph $(N_c, E_c)$ is induced by selecting the elements of strongly connected components to a single point, i.e., $N_c = \{C_1, \ldots, C_m\}$. For each edge $(q_i, r, q_3)$ in $edges(G_M)$, such that $q_i$ is in a SCC $C_i$, let $q_i'$ be a point in a SCC $C_i'$. There is an induced edge $(v, r, v')$ in $E_c$, where $v$ is the set of all induced edges. The induced graph $(N_c, E_c)$ is partially ordered, and is therefore an acyclic digraph.

Since the induced graph $(N_c, E_c)$ is acyclic, a topological sort can be done with complexity linear in the size of $G_M$ (see [Meh84]). This fact will be exploited in the next procedure for eventuality checking. Thus, we can always reorder the SCCs of the induced digraph into a topological sorting order $K_1, \ldots, K_j$ such that, if there is an edge from $K_i$ to $K_j$, then $i \leq j$.

See Figure 4 for an example of the graph induced from the reachability graph of Figure 2. There are three strongly connected components called A, B, and C. There are two possible topological orderings: ABC or ACB. The B and C components are terminal. If $u_\tau = 3$ and $u_\psi = \infty$, then $\tau$ is a progress edge, but $\psi$ is not. If the system is in A, then within 3 ticks of the clock, either $\tau$ or $\psi$ must be taken.

**Procedure 3** ($\psi$-liveness test) Let $\psi$ be a state-formula whose only satisfying state-map is $q$ (e.g., $\psi = on_{\psi} = 1$ is the state-formula corresponding to the state-map $(\{x, \alpha, (y, 1)\})$). Given a node $q$ of $G_M$ and a goal predicate $\psi$, compute if successful, the topologically ordered graph $G_M^\psi(q, \psi)$ induced from $G_M(q, \psi)$. The procedure is successful if all projections of legal trajectories with initial state-map $q$ satisfy $q \rightarrow \psi$. It

1. Compute $G_M(q, \psi)$.
2. Decompose $G_M(q, \psi)$ into its strongly connected components $C_1, \ldots, C_m$.
3. Sort the strongly connected components into topological order to obtain the induced graph $G_M^{\text{ind}}(q, \psi) = (N_c, E_c)$ where $N_c = \{C_1, \ldots, C_m\}$.
4. FOR $i = 1, \ldots, m$ DO
   (a) IF $C_i$ is terminal THEN the procedure fails (and exits) if the state-map in $C_i$ does not satisfy $\psi$
   (b) ELSE (if $C_i$ is non-terminal) the procedure fails (and exits) if there is no progress edge of $C_i$ (leading away from $C_i$).
5. Procedure succeeds with output $G_M^{\text{ind}}(q, \psi)$.

**Procedure 4** (decision procedure for qualitative eventualities)

**Given** - the qualitative eventuality $\psi = \rightarrow \psi$

1. Compute the reachability graph $G_M$ and compute $I := nodes(G_M) \cap V_0$.
2. All state-maps in $I$ are initially unmarked.
3. Initially $\phi := \psi$ and $G^\phi := \emptyset$.
3. WHILE there is an unmarked state-map in $I$ DO
   (a) Select an unmarked $q$ in $I$ and mark it.
   (b) IF Procedure 3 succeeds in producing $G_M^{\text{ind}}(q, \psi)$ THEN $G^\phi := G^\phi \cup G_M^{\text{ind}}(q, \psi)$ ELSE fail and exit.
4. (c) Redefine $\phi$ to be satisfied in $q$ iff either $\psi$ is satisfied in $q$ or $(q$ is in a SCC of $G^\phi$).
4. The procedure succeeds with output $G^\phi$.

Step 3(c) in the above procedure ensures that there is no duplication of checks already performed. Whenever a path in $G_M(q, \psi)$ (or the corresponding induced path in $G_M^\psi(q, \psi)$) reaches a state-map which has already been checked out as achieving the goal predicate, then there is no need to check any successors of that state-map. In step 3(c), the goal predicate is redefined to be satisfied also in a state-map of the
already checked part of the graph $G_a$. Thus, as soon as the computation reaches an already marked node, the node and its successors do not have to be processed further. Therefore, no node or edge in the reachability graph is visited more than once and the procedure is thus linear in the size of the reachability graph.

Proposition: Procedure 4 is successful if the eventuality property given by $(\phi_M \equiv T) \rightarrow O(\phi_M \equiv T + u)$ is $M$-valid. (For proof see [Gus88, Gus89].)

6.1 Real-time response

We can now present the decision procedure real-time response properties such as

$$(\phi_M \equiv T) \rightarrow O(\phi_M \equiv T + I \leq T + u)$$

The procedure first decides if the corresponding qualitative eventuality property $\phi_M \rightarrow O\phi_M$ is $M$-valid. Then an upper bound $U$ is computed by taking the maximum time it takes for any path to reach the goal predicate $\psi$. If $U \leq u$, then the qualitative (real-time) property is $M$-valid.

If $G_M$ is computed via $RG_1$, then $I = 0$ as there are no guaranteed delays. See the discussion in the next section on computing a minimal time delay $I$ for $RG_2$ graphs.

The real-time response procedure (given below) first computes the graph $G_a$ of SCCs from the reachability graph of the TTM $M$. Since $G_a$ is an acyclic graph, it may be sorted into topological order to produce a new graph $(V, E)$. An initial SCC is any SCC in $V$ that contains a state-map satisfying $\phi_M$. Such an initial SCC is always the starting point for any legal trajectory that has initial state satisfying $\psi$. The maximum clock time $U$ to reach a subsequent state satisfying the goal predicate $\psi$ must be computed. If $latest[v]$ is the latest time to reach the SCC $v$ from an initial SCC, then $U$ may be computed by taking the maximum of $latest[v]$ of any terminal SCC $v$ (each terminal SCC consist of a single state-map satisfying $\psi$). Thus the standard algorithm (with some modifications) for computing maximum cost paths, in this case from initial SCCs to terminal SCCs, may be used. The maximum cost algorithm is known to be linear in the size of $(V, E)$ because $(V, E)$ is acyclic [Mel84, p45].

In the maximum cost algorithm, if $(r, v, w)$ is the last edge on a maximum cost path from an initial SCC to a SCC $v$, then the maximum cost to reach $v$ is

$latest[v] = latest[r] + cost(r, v)$

where $cost(r, v)$ is the cost of the edge $(r, v, w)$, which may be thought of as the maximum time $dwell[r]$ that the system is allowed to dwell in the SCC $r$ before it is forced to move to $v$. An obvious (but incorrect) choice for the cost is $cost(r, v) = u$. Since $\tau$ may already have been enabled in the previous SCC $r$, the true cost may be less than $u$, by some offset value. In the procedure below, $cost(r, v)$ is replaced by $dwell[r]$ together with an offset where applicable. The computation of dwelling and offset times will be explained further after the presentation of the procedure.

**Procedure 5 (real-time response)** Given the real-time response property $(\phi_M \equiv T) \rightarrow O(\phi_M \equiv T + u)$, decide whether the property is $M$-valid.

1. Do Procedure 4 to produce $G_a$.
2. Do a topological sort of $G_a$ to produce the graph $(V, E)$ with the SCCs in $V$ labelled as $V = \{1, \ldots, n\}$, in topological order. Each edge $(u, v, w)$ in $E$ has associated with it an offset denoted $offset(u, v, w)$ which is initially set to zero.
3. Initialize the integer arrays $latest[1..n]$, $dwell[1..n]$ and $source[2..n]$ to zero.
4. For each initial SCC $i \in V$, compute the maximum time that is spent in $i$ by:

$$dwell[i] := \min_{progress\ edge\ (r, v) \in E} \{u_r\}$$

where $u_r$ is the upper time bound of $r$.

$latest[i]$ is computed by taking the minimum of all the upper time bounds of progress edges exiting from the SCC $i$ to some later SCC $v$. Some edge exiting $i$ must be traversed by the end of this dwelling time.

5. For all $(i, r, v) \in E$ where $i$ is an initial SCC DO

(a) $latest[i] := dwell[i]$ { $latest[i]$ is the current latest time to reach SCC $v$.}

(b) $source[i] := i$ { $source[i]$ points to the current source of the SCC $v$, i.e. the last edge on the currently computed maximum time path through $v$ is $(i, r, v)$.

6. FOR $r = 2, \ldots, n$ (going from SCC to SCC in topological order) DO

(a) Compute the maximum dwelling time in SCC $r$ by

$$dwell[r] := \min_{progress\ edge\ (r, v) \in E} \{u_v\}$$

(i.e. compute the minimum upper time bound of all progress edges exiting from the SCC $r$ to some later SCC $v$.)

(b) FOR all $(r, r, v) \in E$ (for all edges leaving the SCC $r$) DO

IF $(r, r, v)$ is a progress edge and there is a progress edge $(source[r], r, w) \in E$ (i.e. $\tau$ was previously enabled as a progress edge exiting from $source[r]$ to some SCC $w$)

THEN


ii. IF $latest[r] < newPathTime$ { there is a new critical path }

THEN $latest[r] := newPathTime;

$offset(source[r], r, w) := offset(source[r], r, w) + dwell[r];$ $source[u] := r$

ELSE


ii. IF $latest[r] < newPathTime$ { there is a new critical path }

THEN $latest[r] := newPathTime; source[u] := r$

7. Compute the maximum time $U$ it takes to get to a terminal SCC:

$$U := \max_{i \in terminal} \{latest[i]\}$$

8. The procedure succeeds if $U \leq u$.

In step 1, the qualitative eventuality property is verified using Procedure 4, and the graph $G_a$ is computed. The qualitative eventuality must of course be true as a prerequisite for the real-time response to be true. Since $G_a$ is acyclic, it may be sorted into a topological order in step 2. Thus $(V, E)$ is a topologically ordered graph of strongly connected components of that part of the reachability graph that can be accessed from the initial states in $\phi_M \cap mode(G_M)$.

Steps 3 to 6 modify the standard algorithm for computing the maximum cost path in an acyclic graph. Given an initial node in the graph, the maximum cost path from initial to terminal nodes is determined (the maximum cost paths are called critical paths). As mentioned earlier, the default upper time bounds must be offset by the amount of time that has ticked since the transition was first enabled. The details are explained below.

In step 3, the declared arrays have the following meaning on a path $source[r] \rightarrow r \leftarrow v$

in the graph $(V, E)$:
lатьe[r] = the latest time to reach SCC r from the first SCC on the current critical path through r. In step 6, the latest time is updated if r is on a longer path than in the previous computation of the latest time it takes to reach SCC r from the first SCC.

dwell[r] is the maximum dwelling time in SCC r, i.e., the maximum time from when r is first enabled until some edge is traversed that leads away from r.

source[r] is the node immediately preceding r on the currently computed critical path.

The algorithm used in Procedure 5 is based on the following observation. If (r, r, v) is the last edge on a critical path which starts at the first node and passes through source[r] → r → v then:

\[
lатьe[r] = \text{latest}[r] + \text{dwell}[r]
\]

i.e., the latest time to reach some v from r is the latest time to r plus the longest time the system may dwell in r before the transition r occurs. The dwelling time is computed by the minimisation operation shown in steps 4 (for initial SCCs) and 6a (for subsequent SCCs). The correctness of the minimisation follows directly from the upper time bound requirement of linear trajectories. If more than one transition having a finite upper time bound is enabled in a SCC, then within the least of the upper time bounds some transition leading away from the SCC must be taken. In step 5 the latest time to reach a SCC, whose source is the first SCC, is computed.

Step 6 proceeds in topological order from SCC to SCC and updates latest times to reach the SCC, the source of the SCC and the offset on the edges leading away from the SCC. The offset of an edge (r, r, v) is the amount of time elapsed since r was first enabled on a critical path through r and v. Since we proceed in topological order to the SCC r it follows that on reaching r all edges going into r will already have been evaluated for maximum cost. Thus on reaching some r in step 6b, latest[r] is the maximum cost from the first SCC to r.

If r was not enabled as a progress edge in the SCC source[r], then no offset needs to be added to the dwelling time in r computed in 6a from the default upper time bounds. So latest[r] for the current path is calculated as in equation (2) above, and compared to the previously computed value of latest[r], the greater of these 2 quantities being used for the updated value of latest[r] (see ELSE clause on 6b). The THEN clause of step 6b is taken in case an offset must be subtracted from the dwelling time owing to the fact that the transition was first enabled in an earlier SCC.

The terminal SCCs contain those state-maps in which the goal predicate is first realized on paths starting from the first SCC. Thus in step 7, the maximum time U to realize the goal predicate can be computed by finding the maximum latest time for a terminal SCC. Finally, the procedure succeeds in step 8 if U is less than the time u specified in the real-time response property to reach the goal predicate.

The complexity of the procedure is linear in GM. Steps 1 and 2 have already been shown to be linear. During steps 3 to 6, each node and edge in G0 is visited once and therefore these steps taken together are linear in GM.

7 Modifications for RG2

The real-time response and eventuality procedures must be modified when RG2 is used to produce the reachability graphs. If each tick edge in a reachability graph G (computed via RG2) is given a weight of one and all other edges a weight of zero, then standard shortest path (longest path) algorithms can be applied to G0 to compute the least time L (maximum time U) to the goal predicate in the real-time response property. Since G0 is acyclic, the shortest path (longest path) algorithms are linear in the size of G.

8 Discussion

Procedures have been given for verifying invariances and real-time liveness properties for finite state distributed systems that are linear in the size of the reachability graph. The problem of combinatorial explosion of state-maps (as the number of processes of M increases) limits the practical application of the procedures to those cases where either a "small" but crucial core of the main system can be isolated, or by focusing on the "synchronization part" of the system rather than on the "functional part" (as in [MW84]). For example, that part of a distributed program that ensures mutual exclusion between sections of code is the synchronization part, whereas the code that is made mutually exclusive is the functional part.

The subject of compositional verification methods for modular specification and development is currently under investigation (see [Pnu86] for some preliminary approaches to modularity for real-time systems). If the system can be decomposed into "small" modules, then decision procedures may obviously be useful in verifying parts of the system. It is not clear at this point to what extent temporal logic must be adapted (e.g., via the addition of past operators) for the purposes of compositional verification, and thus it is also unclear to what extent the decision procedures will need to be modified.

References


Figure 4: Topologically sorted strongly connected components induced from Figure 2.