Formal Specification of Real-Time Systems

Farnam Jahanian†, Aloysius K. Mok† and Douglas A. Stuart†

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Key words: *Real Time Logic, real-time systems, specification, verification, timing constraints*

Abstract

This paper presents the formal syntax and semantics of Real Time Logic (RTL), a logic for the specification of real-time systems. An example illustrating the specification of a system in RTL is presented, and natural deduction is used to verify that the system satisfies a given safety property. RTL is shown to be undecidable by a reduction from the acceptance problem for two-counter machines. Decidable subclasses of the logic are also discussed.

1. Introduction

As more real-time safety-critical functions come under software control, e.g., the European Airbus, the reliability of software that must meet not only functional but also stringent timing requirements has become an important issue. One major area of research addressing software reliability has been formal verification methods which attempt to prove the correctness of programs with respect to systems specifications. Past research, however, has concentrated on the relative timing of events and there has been relatively little work on verifying properties which involve absolute (wall-clock) time. Since absolute timing properties can be important in the safe operation of systems, there is a need for applying formal techniques to verifying absolute timing properties. (Leveson and Harvey [Leveson & Harvey 83] have documented a case study where a NASA satellite could have been damaged if the time interval between two occurrences of an event is too short. For a comprehensive survey of *software safety* issues, see [Leveson 86].)

† Supported in part by a research grant from the Office of Naval Research under ONR contract number N00014-85-K-0117, and by a University Research Initiative grant under ONR contract number N00014-86-K-0763.

First presented in [Jahanian & Mok 86a], RTL (Real Time Logic) is one such formalism. It is a first order logic with a predicate which relates the events of a system to their time of occurrence. RTL was developed as part of the SARTOR project [Mok 85], the goal of which is to provide a unified environment for the development and verification of correct real time programs. Since then, a procedure has been developed to determine whether the formulas of one useful subset of RTL are satisfiable [Jahanian & Mok 87], and it has been used to define the formal semantics of a technique to simplify both specification and analysis of large systems [Jahanian et al 88b]. RTL has also been employed by Wing and Barbacci [Barbacci & Wing 87] in separating timing constraints from the funtional specification of abstract data types, and extended by MacEwen [MacEwen 87] into a second order language for the specification of hardware.

This paper defines RTL in terms of an uninterpreted predicate. This avoids the difficulties of using an uninterpreted function to define the time of occurrence of events that never happen ([Jahanian & Mok 86a]). We also identify the core set of axioms which must be satisfied by an RTL specification.‡ This paper also includes examples of system specification and verification in RTL and some results about the decidability of RTL.

Temporal logic has been widely used for the specification and verification of concurrent systems (e.g.[Lamport 83], [Pnueli 81]). However, these temporal logics only allow qualitative reasoning about time. Several extensions have been proposed for expressing and reasoning about real-time systems. These include [Bernstein & Harter 81], [Ostroff & Wonham 87], [Shankar & Lam 87]. These approaches use interleaving to model concurrency. Petri nets have also been extended to model time-dependent systems (e.g. [Merlin & Farber 76], [Ramchandani 74], [Razouk & Phelps 84], [Zuberek 1980]). These techniques also use interleaving.

RTL reflects a different approach to reasoning about time-dependent systems. RTL makes no mention of states or transitions, either in the language or in its model. Although an RTL interpretation can be regarded as a sequence of event sets, it is not necessary to do so, and this is still very different from most other techniques of modeling computer systems, which rely on transition systems. In the transition system approach, the behavior of the system is represented by a sequence of states, which are intended to represent the system configurations. Two consequences of this distinction are discussed below.

‡ This paper thus constitutes the formal definition of RTL.

First, in RTL time passes between sets of events, and the actual sets of events are instantaneous, while in transition systems, time passes in states, and transitions between them are instantaneous. Second, transition systems are generally defined by their transitions, so that if two actions of the modeled system occur concurrently, a transition must be explicitly included to reflect that concurrent act, while such concurrency is implicit in RTL.

This last difference has two further results. Any computation of a transition system induces a total order on event occurrences, even those that occur *at the same time*. The fact that interleaving guarantees the existence of other computations where the ordering is different does not change this for any particular computation. Also, if two actions begin and end at the same time, transition systems introduce faux-states in which one action has completed and the other has not. This last point deserves more elaboration. Consider the following scenario.

There are two sensors, X and Y, in a certain distributed system. Sensor X (Y) is activated to respectively update a variable x (y). Both x and y are set to 0 initially. Each sensor takes 1 second to complete an update. A timing constraint has been specified to activate both sensors at time=0 and update the variables x and y by time=1. Obviously, both x and y must be updated in parallel. At time=1, x (y) is respectively updated to a new value x' (y'). In the interleaving model, the parallel execution of X and Y is equivalent to the two execution sequences: (1) X; Y. (2) Y; X. We characterize the state of the system by the vector (clock,x,y). The trajectories of the system state corresponding to the two execution sequences are:

(1) $(0,0,0) \rightarrow (?,x',0) \rightarrow (1,x',y')$

(2) $(0,0,0) \rightarrow (?,0,y') \rightarrow (1,x',y')$

There are two obvious choices: 0 or 1 in assigning a value to the clock variable "?" in the second state in both trajectories. Neither one makes sense because we know that both x and y are 0 when clock=0, and both x and y are 1 when clock=1. The second states in both trajectories are faux-states.

To avoid faux-states in the interleaving model, one could use a very fine time scale and insist that only one transition can occur at any instant of time, and revise the timing constraints so that the sensors take not one time unit, but between 0.9 and 1.0 time units. Thus, for example, sensor X might complete updating at time=0.9 and sensor Y at time=1.0, the basic time unit being revised downwards to 0.1 second. However, this approach puts a bound on the number of event occurrences in a finite interval, and this bound is on the number of events in the entire system, not just the sensor subsystem. The

0.1 second precision will not be sufficient if there are 100 instead of 2 sensors, or more importantly, if the number of events per second increases in some other part of the system. This would severely inhibit modularity of both design and analysis.

An alternative approach for avoiding faux-states is to insist that events that happen simultaneouly must be grouped into a single event. However, a combinatorial explosion in the number of joint events may result from this approach, since a single event may not be sufficient because a specification may require two events to happen simultaneously some of the time but not all of the time. Therefore, in the worst case, we might have to consider all comibinations of potentially simultaneous events. As will be seen, by its very nature, RTL avoids this difficulty.

2. Real Time Logic

The first subsection describes an informal overview of RTL. The subsequent two subsections provide the formal syntax and the semantics of the logic. The final subsection introduces a convenient notation in RTL for describing a property of a system over a time interval.

At this point it is appropriate to mention that RTL uses a discrete model of time. As a result, integers are used for times in RTL formulas, and not real or rational numbers. There are several reasons for this. Since digital systems are by definition discrete, modeling them using discrete time seems appropriate. We can therefore specify the timing constraints of interest using discrete time. Also, there are philosophical and physics questions that become involved as the difference between distinct times gets smaller.

2.1. Overview

Real Time Logic is a first order predicate logic invented primarily for reasoning about timing properties of real-time systems. It provides a uniform way for the specification of both relative and absolute timing of events. In RTL, we reason about individual occurrences of events where an event occurrence marks a point in time which is of significance to the behavior of the system. There is an important distinction between an action and an event in RTL. An action is an operation which requires a non-zero but bounded amount of system resources. However, events serve only as temporal markers. An occurrence of an event defines a time value, namely its time of occurrence, and imposes no requirement on system resources. The execution of an action is represented by two events: one denoting its initiation and the other denoting its completion.

Events have unique names. Two classes of events are of particular interest: (1) start/stop events marking the initiation and completion of an action, and (2) transition events denoting a change in a state variable.

Start and Stop Events: We use the notation ↑A to represent the event marking the initiation of action A, and ↓A to denote the event marking the completion of action A. For instance, ↑SAMPLE and ↓SAMPLE represent the events corresponding to the start and the stop of action SAMPLE, respectively.

Transition Events: A state variable may describe a physical aspect or a certain property of a system, e.g., an autopilot switch which is either ON or OFF. The execution of an action may cause the value of one or more state variables to change. A state attribute, S is a predicate which asserts that a state variable takes on a certain value in its domain. For example, S may denote the predicate: the autopilot is ON. The corresponding state variable transition events, represented syntactically by (S:=T) and (S:=F), denote respectively the events that mark the turning on and off of the autopilot switch. As in [Heninger 80], a transition event occurs when the value of state attribute changes from true to false or vice versa. Therefore, the events (S:=T) and (S:=F) will alternate. This makes it possible to assign times to the occurrences of theses events. If the events did not alternate, it would be necessary to assign a time to a non-event, i.e., the state attribute staying the same, and there would be no reason to prefer one such time to another, making such an assignment difficult. Also this might require the assignment of a time to something that was not observable, such as turning on a switch that was already on. Accordingly, alternating transition events seems the most reasonable approach to take. Whenever it is unambiguous, we shall use state variable and state attribute interchangeably.

In addition to the above two classes, other classes of events can be specified when modeling a real time system. For instance, in [Jahanian & Mok 86a], the class of external events was introduced to denote the events that cannot be caused to happen by the computer system but can impact a system behavior. We use the notation of any name in capital letters prefixed by the special letter Ω (Omega) to denote an external event. For example, ΩBUTTON1 represents the external event associated with pressing button 1.

All of the events of a system are considered to constitute the system's set of events. For example, if a system consisted exactly of those events described in the three preceding paragraphs, its set of events would be {↑SAMPLE, ↓SAMPLE, (S:=T), (S:=F), ΩBUTTON1 }.

The *occurrence relation*, denoted by the letter R (Theta), is introduced to capture the notion of real time. The event constants introduced earlier represent the things that can happen in a system. The occurrence relation assigns a time value to each occurrence of an event which happens. Informally, $R(e,i,t)$ denotes that the ith occurrence of an event e happens at time t, where e is an event constant, i is a positive integer term, and t is non-negative integer term. For instance, $R(\downarrow\text{SAMPLE},1,x)$ denotes that the first occurrence of the event marking the completion of action SAMPLE happens at time x.

The notion of an occurrence relation is central to RTL. In particular, a specification of a system and the timing requirements on its behavior are restrictions on the occurrence relation and its arguments. Observe that the occurrence relation does not require that all occurrence of an event must happen since an event may occur only a finite number of times or even not at all.

RTL predicates are formed from the occurrence relation, or from the mathematical relations (=, <, ≤, >, ≥) and algebraic expressions allowing integer constants, variables, and addition. Multiplication by constants is used as an abbreviation for addition. One further restriction is that time values cannot be added to or compared with occurrence values. RTL formulas are constructed using the occurrence relation, the equality/inequality predicates, universal and existential quantifiers, and the first-order logical connectives $(\neg,\wedge,\vee,\rightarrow)$†. Before presenting a formal treatment of Real time Logic in the next section, a simple example will be shown to illustrate how the specification of a system can be expressed in RTL.

Two of the choices indicated in the preceding paragraph may require further explanation. The separation of occurrence values and time values seems reasonable from a unit analysis perspective, and makes the logic more amenable to analysis. Also a review of real-time systems does not seem to indicate a need for doing so. The decision not to permit multiplication seems reasonable since there does not seem to be a need to multiply either occurrence values or time values.

Example:

Consider the English description of a simple system which samples and displays information on demand from an external stimulus. Upon pressing button #1, action SAMPLE is executed within 30 time units. During each execution of this action, the information is

† The standard precedence order is assumed for these connectives. $\neg$ has the highest precedence, $\wedge$ and $\vee$ the next highest precedence, and $\rightarrow$ the lowest precedence.

sampled and subsequently transmitted to the display panel. The computation time of action SAMPLE is 20 time units. The following set of formulas is a partial description of the system in RTL:

$$\forall i\ \forall t\ \ R(\Omega\mathrm{BUTTON1},i,t) \ \rightarrow\ [\ \exists x,y\ \ R(\uparrow\mathrm{SAMPLE},i,x)\ \wedge\ R(\downarrow\mathrm{SAMPLE},i,y)\ \wedge\ t \le x\ \wedge\ y \le t+30\ ]$$

$$\forall i\ \forall x,y\ \ [\ R(\uparrow\mathrm{SAMPLE},i,x)\ \wedge\ R(\downarrow\mathrm{SAMPLE},i,y)\ ] \ \rightarrow\ x+20 \le y \qquad \square$$

2.2. The Language of Real Time Logic

We begin by introducing the language of Real Time Logic. The formulas of RTL are made up of the following symbols:

- The truth symbols *true* and *false*
- A set of time variable symbols **A**.
- A set of occurrence variable symbols **B**.
- A set of constant symbols **C** including the natural numerals
- A set of event constant symbols **D**
- The function symbol +
- The predicate symbols <, ≤, >, ≥, =
- The occurrence relation symbol *R*
- The logical connectives $\wedge$, $\vee$, ¬ and →.
- Existential and universal quantifier symbols ∃, ∀.

The *time terms* of RTL are expressions built up according to the following rules:

- The constant symbols in **C** are time terms.
- The variable symbols in **A** are time terms.
- If t_1 and t_2 are time terms, then the function application

  $$t_1 + t_2$$

  is a time term.

The *occurrence terms* of RTL are expressions built up according to the following rules:

- The constant symbols in **C** are occurrence terms.
- The variable symbols in **B** are occurrence terms.
- If i and j are occurrence terms, then the function application

  $$i + j$$

  is an occurrence term.

The *propositions* of RTL are constructed according to the following rules:

- The truth symbols *true* and *false* are propositions.
- If t_1 and t_2 are time terms and ρ is an inequality/equality predicate symbol, then

  $$t_1 \,\rho\, t_2$$

  is a proposition.
- If i and j are occurrence terms and ρ is an inequality/equality predicate symbol, then

  $$i \,\rho\, j$$

  is a proposition.
- If i is an occurrence term, t is a time term and e is an event constant, then

  $$R(e, i, t)$$

  is a proposition.*

The *formulas* of RTL are constructed from the propositions, logical connectives and quantifiers in the usual fashion.

2.3. The Meaning of an RTL Formula

An interpretation for an RTL formula must assign a meaning to each of the free symbols in the formula. It will assign elements from a domain to the constants, functions (over the domain) to the function symbols, and relations (over the domain) to the predicate symbols.

Let **N** be the set of natural numbers, and **E** be a set of events. Usually **D** and **E** are identical, although this is not necessary. An interpretation $s+2Is-2$ over the domain **N**∪**E** assigns values to each of a set of constant, function, and predicate symbols, as follows:

Each element in the set of constant symbols **C** is assigned an element in **N**, with the numerals being assigned to the corresponding natural numbers. Each element in the set of event constant symbols **D** is assigned a distinct element in **E**. The function symbol '+' is integer addition. The predicate symbols $<$, $\leq$, $>$, $\geq$, and $=$ are assigned the usual equality/inequality binary relations. The predicate symbol R is assigned an *occurrence relation*.

Definition: An *occurrence relation* is any relation on the set

$$\mathbf{E} \times \mathbf{Z}^{+} \times \mathbf{N}$$

* An alternative would be to have a family of functions indexed by events, but the authors feel this notation better retains the relationships between events, and has a higher aesthetic value.

where **E** is a set of events, $\mathbf{Z}^+$ is the set of positive integers, and **N** is the set of natural numbers, such that the following axioms hold:

Monotonicity Axioms: For each event e in the set **D**,

$$\forall i\ \forall t\ \forall t'\ [\ R(e,i,t)\ \wedge\ R(e,i,t')\ ]\ \rightarrow\ t = t'$$
$$\forall i\ \forall t\ [\ R(e,i,t)\ \wedge\ i > 1\ ]\ \rightarrow\ [\ \exists t'\ R(e,i-1,t')\ \wedge\ t' < t\ ]$$

The first axiom requires that at most one time value can be associated with each occurrence i of an event e, i.e., the same occurrence of an event cannot happen at two distinct times. The second axiom expresses the requirement that if the ith occurrence of an event e happens, then the previous occurrences of e must have happened earlier. This axiom also requires that two distinct occurrences of the same event must happen at different times.

Start/Stop Event Axioms: For each pair of start/stop events in the set **D**,

$$\forall i\ \forall t\ R(\downarrow A,i,t)\ \rightarrow\ [\ \exists t'\ R(\uparrow A,i,t')\ \wedge\ t' < t\ ]$$

where $\uparrow$A and $\downarrow$A denote the events marking the start and stop of an action A, respectively. The above axiom requires every occurrence of a stop event to be preceded by a corresponding start event.

Transition Event Axioms: For the transition events in the set **D** corresponding to a state variable S,

$$\begin{aligned}&R((S{:=}T),1,0)\ \rightarrow\\&(\ \forall i\ \forall t\ R((S{:=}F),i,t) \rightarrow [\ \exists t'\ R((S{:=}T),i,t')\ \wedge\ t' < t\ ]\ \wedge\\&\forall i\ \forall t\ R((S{:=}T),i+1,t) \rightarrow [\ \exists t'\ R((S{:=}F),i,t')\ \wedge\ t' < t\ ]\ )\end{aligned}$$

$$\begin{aligned}&R((S{:=}F),1,0)\ \rightarrow\\&(\ \forall i\ \forall t\ R((S{:=}T),i,t) \rightarrow [\ \exists t'\ R((S{:=}F),i,t')\ \wedge\ t' < t\ ]\ \wedge\\&\forall i\ \forall t\ R((S{:=}F),i+1,t) \rightarrow [\ \exists t'\ R((S{:=}T),i,t')\ \wedge\ t' < t\ ]\ )\end{aligned}$$

The preceding transition event axioms define the order in which two complementary transition events can occur depending on whether S is initially true or false.

2.4. State Predicates

A specification of a real-time system often refers to properties of the system over time. One way to express these assertions is to introduce predicates which are time-dependent, as is done in temporal logic. In RTL, these assertions are expressed as formulas involving the occurrence relation on the appropriate transition events and inequality relations on the time of occurrences of these events. For convenience, RTL uses a notational device, called a state predicate, for asserting the truth value of a state variable (e.g., the autopilot switch of an airplane being in the ON position) during an interval.

Suppose S is a state variable whose truth value remains unchanged over an interval. RTL provides nine different forms of state predicates to assert the value of S over an interval, depending on the boundary conditions:

S[x,y], S(x,y), S<x,y>, S[x,y), S[x,y>, S(x,y], S(x,y>, S<x,y], and S<x,y).

Each state predicate qualifies the timing of two events, one marking the transition event that changes the value of the state variable to true and the other marking the transition event that changes the value of S to false. The two arguments, x and y, in the state predicates are used in conjunction with the symbols "[", "]", "(", ")", "<" and ">" to denote an interval over which the state variable remains true.

The convention we use for arriving at this syntax requires some explanation. Suppose E_t and E_f denote the transition events making S true and false, respectively. Informally,

"[x " denotes that E_t occurs at time x,
"(x " denotes that E_t occurs before or at time x,
"<x " denotes that E_t occurs before time x,
" y]" denotes that E_f occurs at time y,
" y)" denotes that E_f does not occur before time y,
" y>" denotes that E_f does not occur before or at time y,

For example, the state predicate S[x,y] indicates that a state variable S is true exactly during the interval between x and y. That is, a transition event making S true occurs at time x; S remains true between x and y; and the transition event making S false occurs at time y. Consider another example, the state predicate S(x,y) denotes that a state variable is true during the interval between x and y, but it says nothing about the value of S before time x or after time y. Observe that we use this notation only when $x \leq y$ and that we do not require the second event to occur in all cases. The formal definitions of these state predicates follow.

Definition:

If a state variable S is initially true, i.e., $R((S{:=}T),1,0)$,

$$S[x,y] \equiv \exists i\ R((S{:=}T),i,x) \wedge R((S{:=}F),i,y)$$
$$S[x,y) \equiv \exists i\ R((S{:=}T),i,x) \wedge [\ \forall t\ R((S{:=}F),i,t) \rightarrow y \leq t\ ]$$
$$S[x,y> \equiv \exists i\ R((S{:=}T),i,x) \wedge [\ \forall t\ R((S{:=}F),i,t) \rightarrow y < t\ ]$$

The other five forms are defined similarly.

If a state variable S is initially false, i.e., $R((S{:=}F),1,0)$,

$$S[x,y] \equiv \exists i\ R((S{:=}T),i,x) \wedge R((S{:=}F),i+1,y)$$

The other eight forms are defined similarly.

According to the above definition, if S is initially true, then the predicate S[x,y> means that the state variable S becomes true exactly at time x, and that if it ever becomes false again (the i^{th} occurrence of the event (S:=F) does happen), the time it does so is after y. Note that S[x,y> does require that some occurrence of the event (S:=T) happens, but does not require an occurrence of the event (S:=F). In other words, S[x,y> would be satisfied if S became true at time x, and never became false afterwards.

Two very useful predicates are defined if both arguments of a state predicate are the same. Specifically, the state predicate S(x,x) says that a system attribute S is true at an interval around time x. Similarly, S<x,x) denotes that S is true prior to time x, i.e., S becomes true sometime before time x and it remains true at least up to time x.

Similar state predicates can be defined for the case when a state variable is false during an interval: $\bar{S}[x,y]$, $\bar{S}(x,y)$, $\bar{S}<x,y>$, etc. The formal definitions for these predicates are straightforward: replace each (S:=T) with (S:=F) and vice versa in the above definitions.

3. Two Examples

This section contains two examples of specification and verification using RTL. The first example uses a simplified nuclear reactor to illustrate a natural deduction approach for verifying a safety property with respect to the specification. The second example applies a resolution-style derivation to a real-time producer-consumer problem.

3.1. Example 1: Moving Control Rods in a Reactor

This example involves a system used to monitor and control a reactor. Specifically, it concerns the part of the system responsible for moving the control rods in a reactor. (Lowering a control rod slows down a nuclear reaction.) For simplicity, we assume that the system is currently configured with two primary control rods. As shown in Figure 1, three asynchronous components are involved in moving the control rods: Subsystem 1, Subsystem 2, and Manager.

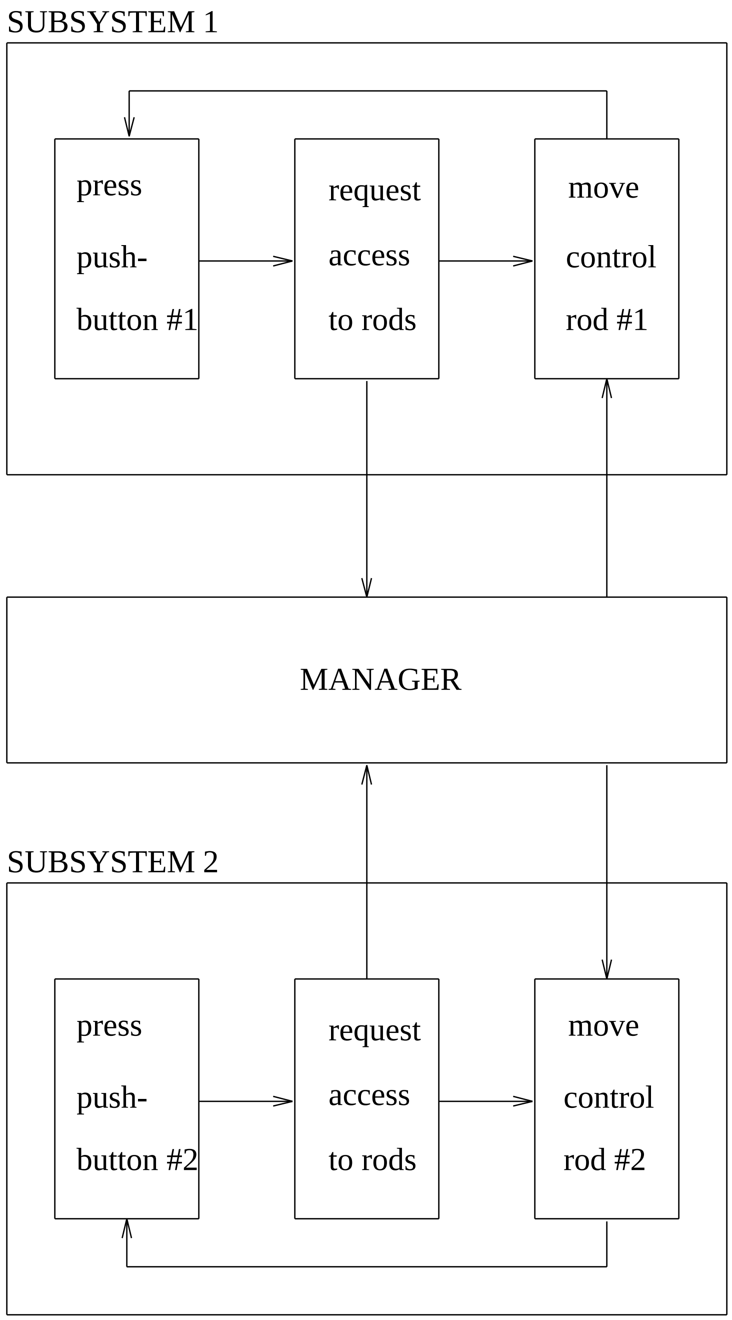

FIGURE 1.

Upon pressing pushbutton #1, Subsystem 1 executes an action to ensure the conditions are satisfied for moving control rod #1, then it requests permission from the manager to move a rod. After the manager grants the request, Subsystem 1 issues the appropriate commands to the reactor to move rod #1. The specification imposes two timing constraints on Subsystem 1:

- When the request is granted by the manager, the action to move the rod must be started within 5 time units, and
- at most 20 additional time units is permitted to complete the action to move the rod.

Subsystem 2 performs similar tasks for moving control rod #2 with identical timing constraints. The details about the behavior of Manager are hidden at this level of specification. However, the following timing constraint is imposed on Manager:

- After granting a request, Manager must wait at least 30 time units before granting another request.

There is no signal from Subsystem 1 or Subsystem 2 to inform Manager after a rod movement has been completed. Hence, Manager can only rely on the enforcement of the preceding timing constraint to decide when a granted request has been released, and then grant permission to move a rod to a waiting subsystem.

For ease of understanding, we list below the physical interpretation of the events which will appear in the RTL specification of the above system.

ΩBUTTON1	: external event denoting pushbutton #1 is pressed
↑RA1	: start of action to request access by Subsystem 1
↓RA1	: stop of action to request access by Subsystem 1
GRANT1:=T	: event denoting request by Subsystem 1 is granted
GRANT1:=F	: manager assumes the granted request is released
↑MOVE1	: start of action to move control rod #1
↓MOVE1	: stop of action to move control rod #1

The remaining events ↑RA2, ↓RA2, GRANT2:=T, GRANT2:=F, ↑MOVE2, and ↓MOVE2 have similar meanings.

The preceding English description of the system can be specified formally in RTL, as shown below.†

#

specification of subsystem 1

#

$$\forall i\ \forall t\ \ R(\Omega\mathrm{BUTTON1}, i, t) \rightarrow [\ \exists t'\ \ R(\uparrow\mathrm{RA1}, i, t') \wedge t \le t'\ ] \qquad \text{(SP1)}$$

$$\forall i\ \forall t\ \ R(\uparrow\mathrm{RA1}, i, t) \rightarrow [\ \exists t'\ \ R(\downarrow\mathrm{RA1}, i, t') \wedge t \le t'\ ] \qquad \text{(SP2)}$$

$$\forall i\ \forall t\ \ R(\downarrow\mathrm{RA1}, i, t) \rightarrow [\ \exists t'\ \ R((\mathrm{GRANT1{:=}T}), i, t') \wedge t \le t'\ ] \qquad \text{(SP3)}$$

† Certain RTL axioms, as described in [Jahanian & Mok 86a], are omitted here since they do not play a role in the safety analysis of this example.

$\forall i\ \forall t\ \ R((GRANT1:=T), i, t) \rightarrow [\ \exists t'\ \ R(\uparrow MOVE1, i, t') \wedge t \le t' \wedge t' \le t + 5\ ]$ (SP4)

$\forall i\ \forall t\ \ R(\uparrow MOVE1, i, t) \rightarrow [\ \exists t'\ \ R((GRANT1:=T), i, t') \wedge t' \le t \wedge t \le t' + 5\ ]$ (SP4')

$\forall i\ \forall t\ \ R(\uparrow MOVE1, i, t) \rightarrow [\ \exists t'\ \ R(\downarrow MOVE1, i, t') \wedge t < t' \wedge t' \le t + 20$ (SP5)

#

specification of subsystem 2

#

$\forall i\ \forall t\ \ R(\Omega BUTTON2, i, t) \rightarrow [\ \exists t'\ \ R(\uparrow RA2, i, t') \wedge t \le t'\ ]$ (SP6)

$\forall i\ \forall t\ \ R(\uparrow RA2, i, t) \rightarrow [\ \exists t'\ \ R(\downarrow RA2, i, t') \wedge t \le t'\ ]$ (SP7)

$\forall i\ \forall t\ \ R(\downarrow RA2, i, t) \rightarrow [\ \exists t'\ \ R((GRANT2:=T), i, t') \wedge t \le t'\ ]$ (SP8)

$\forall i\ \forall t\ \ R((GRANT2:=T), i, t) \rightarrow [\ \exists t'\ \ R(\uparrow MOVE2, i, t') \wedge t \le t' \wedge t' \le t + 5\ ]$ (SP9)

$\forall i\ \forall t\ \ R(\uparrow MOVE2, i, t) \rightarrow [\ \exists t'\ \ R((GRANT2:=T), i, t') \wedge t' \le t \wedge t \le t' + 5\ ]$ (SP9')

$\forall i\ \forall t\ \ R(\uparrow MOVE2, i, t) \rightarrow [\ \exists t'\ \ R(\downarrow MOVE2, i, t') \wedge t < t' \wedge t' \le t + 20$ (SP10)

#

specification of manager

#

$\forall i\ \forall t\ \ R((GRANT1:=T), i, t) \rightarrow [\ \exists t'\ \ R((GRANT1:=F), i+1, t') \wedge t + 30 \le t'\ ]$ (SP11)

$\forall i\ \forall t\ \ R((GRANT2:=T), i, t) \rightarrow [\ \exists t'\ \ R((GRANT2:=F), i+1, t') \wedge t + 30 \le t'\ ]$ (SP12)

$$\begin{aligned}\forall i,j\ \ \forall t_1,t_2,t_3,t_4\ \ [\ & R((GRANT1:=T), i, t_1) \wedge R((GRANT1:=F), i+1, t_2) \wedge \\ & R((GRANT2:=T), j, t_3) \wedge R((GRANT2:=F), j+1, t_4)\ ] \\ & \rightarrow t_4 < t_1 \vee t_2 < t_3 \end{aligned} \quad \text{(SP13)}$$

$R((GRANT1:=F),1,0)$ (SP14)

$R((GRANT2:=F),1,0)$ (SP15)

3.1.1. Safety Assertion

Since Subsystem 1 and Subsystem 2 are asynchronous, the requests to move the rods are made at arbitrary times. The desired safety property is that the reactor rods should be moved one at a time. In other words, execution of the action to move rod #1 may not overlap with the execution of the action to move rod #2.

Safety Assertion in RTL:

$$\begin{aligned}\forall i,j\ \forall t_1,t_2,t_3,t_4\ \ & R(\uparrow MOVE1, i, t_1) \wedge R(\downarrow MOVE1, i, t_2) \wedge \\ & R(\uparrow MOVE2, j, t_3) \wedge R(\downarrow MOVE2, j, t_4) \\ & \rightarrow t_4 < t_1 \vee t_2 < t_3 \end{aligned}$$

3.1.2. Verification

The RTL formulas describing the system specification and the safety assertion were presented earlier. The proof of the safety assertion from the system specification follows. Consider an arbitrary instance of MOVE1 action

1. a. $R(\uparrow\text{MOVE1}, \text{I}, T_1)$
   b. $R(\downarrow\text{MOVE1}, \text{I}, T_2)$

and an instance of MOVE2 action

1. c. $R(\uparrow\text{MOVE2}, \text{J}, T_3)$
   d. $R(\downarrow\text{MOVE2}, \text{J}, T_4)$.

The safety assertion requires that the execution of the two actions must not overlap. The executions of actions MOVE1 and MOVE2 define the intervals $[T_1,T_2]$ and $[T_3,T_4]$, repectively. We need to show that $T_4 < T_1 \vee T_2 < T_3$.

Lemma: $\exists T_5 \exists T_6$ GRANT1$[T_5,T_6] \wedge T_5 \le T_1 \wedge T_2 \le T_6$.
The lemma states that the interval $[T_1,T_2]$ corresponding to the execution of MOVE1 is enclosed within an interval $[T_5,T_6]$ during which state variable GRANT1 is true. The proof of the lemma is as follows:

2.	a. $\exists T_5\ R((\text{GRANT1:=T}),\text{I},T_5) \wedge T_5 \le T_1 \le T_5 + 5$	1.a and SP4’
	b. $\exists T_6\ R((\text{GRANT1:=F}),\text{I+1},T_6) \wedge T_5 + 30 \le T_6$	2.a and SP11
	c. GRANT1$[T_5,T_6]$	def of a state pred, 2.a, 2.b, SP14
	d. $T_1 + 25 \le T_6$	2.a and 2.b
3.	$T_2 \le T_1 + 20$	1.a, 1.b and SP5
4.	$T_2 + 5 \le T_6$	2.d and 3.

Hence, $\exists T_5 \exists T_6$ GRANT1$[T_5,T_6] \wedge T_5 \le T_1 \wedge T_2 + 5 \le T_6$. □

A similar lemma holds for the interval $[T_3,T_4]$ corresponding to the execution of MOVE2 action. Its proof is similar.

Lemma: $\exists T_8 \exists T_9$ GRANT2$[T_8,T_9] \wedge T_8 \le T_3 \wedge T_4 \le T_9$. □

Given the two intervals GRANT1$[T_5,T_6]$ and GRANT2$[T_8,T_9]$, from SP13 we conclude that the two *grant* intervals cannot overlap. Hence, $T_9 < T_5 \vee T_6 < T_8$. Since, by the above lemmas, intervals $[T_1,T_2]$ and $[T_3,T_4]$ are contained inside the corresponding grant intervals, we conclude that the two intervals $[T_1,T_2]$ and $[T_3,T_4]$ cannot overlap:

$$T_3 < T_1 \vee T_4 < T_3.$$

3.2. Example 2: A Real-Time Producer-Consumer

This example presents a system consisting of a producer and a consumer. the producer puts a message in memory at the beginning of every p time units interval. The consumer removes a message from memory and and processes it at any time during each period p. Furthermore, the consumer takes c time units to process a message. Since the producer and the consumer are on different processors, the consumer's clock is α time units behind the producer's clock. The RTL specification of producer and consumer are shown below. (The event P denotes the insertion of a message in memeory by the producer; the event $\uparrow$Q denotes the removal and start of processing a message by the consumer; and the event $\downarrow$Q denotes the completion of processing a message by the consumer.)

Producer: $\forall i \;\; @(P,i) = i*p$

Consumer: $\forall i \;\; i*p + \alpha \leq @(\uparrow Q,i) \wedge @(\downarrow Q,i) \leq (i+1)*p + \alpha$

$\forall i \;\; @(\uparrow Q,i) + c \leq @(\downarrow Q,i)$

Clock Skew: $\alpha < p$

The safety assertion for this system requires that the consumer removes each message from memeory before the producer puts another one in memory. The following RTL formula expresses the safety assertion:

Safety Assertion: $\forall i \;\; @(\uparrow Q,i) \leq @(P,i+1)$

We use a resolution-style proof technique to drive the condition under which the specification satisfies the safety assertion, i.e., the condition under which the formula SP $\rightarrow$ SA is valid where SP is the specification and SA denotes the safety assertion. This is equivalent to finding the condition which makes the formula SP $\wedge$ $\neg$SA unsatisfiable. Removing the quantifiers and skolemizing SP $\wedge$ $\neg$SA result in the following set of RTL clauses:

SP:

(1) $@(P,i) = i*p$

(2) $i*p + \alpha \leq @(\uparrow Q,i)$

(3) $@(\downarrow Q,i) \leq (i+1)*p + \alpha$

(4) $@(\uparrow Q,i) + c \leq @(\downarrow Q,i)$

(5) $\alpha < p$

$\neg$SA:

(6) $@(P,I+1) < @(\uparrow Q,I)$

where i is a variable, p is a constant denoting the period, c is a constant denoting the computation time of action Q, and I is a skolem constant. The condition which makes the above inequalities unsatisfiable is obtained by repeated application of the transitivity axiom:

1. $(I+1)*p < @(\uparrow Q,I)$	(6) and (1)
2. $@(\downarrow Q,I) < @(\uparrow Q,I) + \alpha$	step 1 and (3)
3. $@(\uparrow Q,I) + c < @(\uparrow Q,I) + \alpha$	step 2 and (4)
4. $c < \alpha$	simplifying step 3

The above set of ineqaulities, (1) through (6), admits a solution if $c < \alpha$. Hence, if $\alpha \leq c$, SP $\rightarrow$ SA is valid.

4. Undecidability of Real Time Logic

Since RTL is made up of integer arithmetic without multiplication, together with a single non-algebraic predicate, RTL formulas can be converted into formulas of extended Presburger arithmetic, which is exactly the language of integer arithmetic without multiplication, including integer function symbols, by replacing each occurrence of R by the inequality $f_R > 0$. Although the decision procedure for Presburger arithmetic is double exponential [Cooper 72], this does not have a major impact on RTL, because the techniques used do not apply when function symbols are added. Presburger arithmetic with a single monadic predicate was shown to be undecidable in [Downey 72]. Since the occurrence relation of RTL is constrained by the axioms in section 2.3, RTL is a proper subset of extended Presburger arithmetic and so the undecidability of extended Presburger arithmetic does not imply the the undecidability of RTL. However, we can show that RTL is undecidable by adapting Downey's proof. An outline of the proof follows, but the proof itself can be found in Appendix A.

The central idea of the proof is to encode the configurations of a two counter machine into the occurrence relation, and to write formulas that simulate the finite control of the machine. One event keeps a record of the state of the machine, and two more record the values of the counters. The only complication is that the history of the value must be used instead of the value itself, in order to preserve monotonicity. There are other simpler proofs that can be applied to adding monotonic functions to full Presburger arithmetic, but the restrictions imposed on RTL formulas, notably the separation into occurrence terms and time terms, prevent these from being used here.

5. Decidable Subsets

The subclass of quantifier-free Presburger arithmetic with uninterpreted integer functions has been shown to be decidable in [Shostak 79]. If a Presburger formula containing uninterpreted functions (or predicates) is universally quantified, Shostak shows that F can be reduced to an *equivalid* formula F' free of uninterpreted functions. The correctness of the reduction is straightforward; given a model for ¬F, one can construct a model for ¬F', and conversely. Since F' is a Presburger formula, its validity can be checked by a decision procedure for Presburger arithmetic.

The class of RTL formulas in [Jahanian & Mok 87] also has a subset for which a decision procedure for determining the satisfiability (or unsatisfiability) of a formula in the subclass can be shown. Suppose an RTL formula is of the form

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n$$

such that each C_i is of the form

$$(Q_i \ \ L_1 \vee L_2 \vee \cdots \vee L_m)$$

where Q_i is the prefix (quantifiers) for the disjunction and each L_j is an arithmetical relation. Furthermore, suppose that each inequality L_j is of the form

$$\text{occurrence function} \pm \text{integer constant} \ \leq \ \text{occurrence function}$$

where the occurrence functions contain integer variables quantified by the corresponding matrix. Note that the negation of an inequality can be replaced by an equivalent inequality without the negation.

Theorem: The above subclass is decidable if the quantifiers in each prefix Q_i are either all ∀ or all ∃.

Proof: The proof of this theorem follows directly from Herbrand's Theorem. Suppose F is a formula in the above subclass. From the hypothesis, skolemizing F produces a set of clauses S such that each skolem function in S is a 0-place function, i.e., a constant. Furthermore, the integer variables in S (denoting occurrences) appear only inside occurrence functions. Since the "@" function does not take an instance of itself as an argument, the Herbrand Universe is finite. By Herbrand's Theorem, the set of clauses in S is unsatisfiable iff there is a finite unsatisfiable set S' of ground instances of S. Since Herbrand's universe is finite, its powerset is finite and so there is a finite number of ground instances of S which must be checked. Checking a finite set of ground clauses can be done using any decision procedure for propositional logic.

□

6. RTL and Temporal Logic

In addition to the differences in the models of computation and the introduction of the notion of an event occurrence, there is one further distinction between RTL and propositional linear temporal logics, which are the languages generally used to describe transition system based models. A similar construction to that in section 4.3 can be used to show that any finite state control can be simulated by an RTL formula, by encoding the configurations of the machine as events, and using formulas to reflect the transitions allowed by the finite state control. Similarly the various acceptance criteria for finite automata on infinite strings can also be expressed in RTL. As an example, the requirement that a certain state appears infinitely often could be represented as

$$\forall i \, \exists j \;\; @(st,j) - @(st,j\text{-}1) = desiredstate \;\wedge\; j > i$$

Accordingly, RTL is at least as expressive as finite state machines over infinite strings. Propositional temporal logics are equivalent in expressive power to counter-free automata on infinite strings, and hence cannot express the proposition that the number of times an event happens is congruent to n modulo m for some $n,m \in \mathbf{N}$ [Pnueli et al 85], RTL can express such formulas quite easily. For example, if n and m are 2 and the given event is E, then the desired formula is

$$\exists k \;\; (\exists t \;\; R(E,2*k,t) \;\wedge\; \forall j>2*k \, \neg\exists u R(E,j,u)) \;\vee\; \neg\exists k \exists t R(E,k,t).$$

Combining these two observations, it can be seen that RTL is strictly more expressive than propositional linear temporal logic.

7. Concluding Remarks

This paper presents the definition of RTL, a logic for reasoning about the absolute timing properties of real-time systems. Real Time Logic provides a uniform way for the specification of both relative and absolute timing of events. The ability of RTL to describe real-time systems has been demonstrated by means of an example specification. A safety property for this system was verified using natural deduction in predicate calculus.

The goal of our research is to investigate whether formal methods can be applied to real-time systems. An important direction for future research is determining practical methods for automating the analysis of systems described by RTL formulas. The undecidability proof of RTL given in the appendix applies to any subset where arithmetic relations contain at least three time-valued variables, there are at least three events, and where multiplication by two is allowed. No decision procedure will exist for a superset of this sublogic. However, it can be shown that the graph algorithm of [Jahanian & Mok 87] is a decision procedure for a subset of its domain. There are likely to be expressive subsets of RTL which are amenable to mechanical analysis, either by appropriate syntactic restriction or by exploiting semantic information. Both approaches are under investigation.

Appendix A -- Undecidability of RTL

A.1 Background

Before considering the decidability of RTL, it is helpful to define an abbreviation as was done in [Jahanian & Mok 86a]. If for each event e in some subset $\mathbf{E}'$ of $\mathbf{E}$ $\forall i\ \exists t\ \ R(e,i,t)$ is satisfied, we will define the function

$$@ :\ \ \mathbf{E}' \times \mathbf{Z}^{+} \rightarrow \mathbf{N}$$

such that $@(e,i)=t$ if and only if $R(e,i,t)$. @ is known as the *occurrence function*. Informally, $@(e,i)$ is the time of the i^{th} occurrence of event e. Since for each choice of $e \in \mathbf{E'}$ and i such a t exists, @ is a total function. The monotonicity axioms for R imply that

$$(R(e,i,t)\ \wedge\ R(e,j,t')\ \wedge\ i<j)\ \rightarrow\ t<t',$$

so @ is monotonic in its second argument. RTL formulas which contain only references to this function, and no explicit references to R, form precisely the language described in [Jahanian & Mok 86a], and are a subset of the language of Presburger arithmetic with uninterpreted function symbols. This abbreviation in no way changes the validity of the following proof, but does make it notationally more convenient, since the function @ will indeed be total for each of the events used in the proof.

The proof in [Downey 72] reduces the acceptance problem for deterministic two counter machines to the satisfiability problem in extended Presburger arithmetic. The proof proceeds by constructing a predicate that is true if and only if its argument is a configuration in the computation of the corresponding two counter machine on the given input. This is done by representing the configuration of the machine by a product of prime powers, similar to a Goedel encoding. As a result, any decidable subset of RTL will have to be restricted in the amount of multiplication and addition of occurrence variables that is allowed. Since there is little need for multiplication in the arguments of the occurrence relation in RTL, this does not appear to be a particularly severe restriction.

Even though the occurrence relation (and @ function) of RTL (when in Presburger form) is uninterpreted, the result in [Downey 72] is not sufficient to show that RTL is undecidable, because the standard way of converting a predicate into a function need not result in a monotonic function. One way of doing so is to replace the unary predicate S of [Downey 72], by the event S, which is defined so that $R(S,x,2*x+1)$ if S(x) is true, and $R(S,x,2*x)$ if S(x) is false. The relation $R(S,x,t)$ satisfies the monotonicity axioms, and still meets all of the conditions imposed on S in [Downey 72], so with this modification, that proof can now be used to show that RTL is undecidable.

Since this proof of the undecidability of RTL relies on the Goedel-like encoding, preventing addition of occurrence variables, as well as multiplication by three mutually relatively prime constants in any given set of formulas is one possible way of obtaining a decidable subset of RTL. This restriction is not enough to make RTL decidable, however. A representation for machine configurations can be constructed that meets this new restriction. There is no addition or multiplication of occurrence variables, and the occurrence function is monotonic. Other than the change in representation, the proof that follows closely follows the proof in [Downey 72].

A.2 Two Counter Machines

A two counter machine (2CM) [Hopcroft & Ullman 79] is a quadruple,(Q,q_1,q_m,δ). Q is a finite set of states, with q_1 a distinguished start state and q_m a distinguished final state. δ is a finite transition function from $Q\times\{0,+\}\times\{0,+\}$ to $Q\times\{+,-,0\}\times\{+,-,0\}$. The machine can test each of its counters to see if it is positive or zero, and can then decrement or increment it by one, or leave it unchanged, n a single transition. Without loss of generality it will be assumed that each transition falls into one of four categories, and that a counter is tested to see if an opperation is legal before it is performed (i.e., a ptest preceedes each dec). A ptest checks a single counter to see if it is positive, a ztest does the same for zero, a dec decrements a single counter, and an inc increments a single counter. Notice that unlike in a programming language, both tests are necessary, since the ‘enabling condition’ of a transition must be true for it to be taken, and there is no ‘default’ transition. Lemma 1 of [Downey 72] shows that a 2CM with only these elementary transitions can be constructed that is equivalent to an arbitrary 2CM, so this restriction does not affect the expressiveness of the 2CM, and is done only to simplify the construction and proof. Similarly, it will be assumed that the machine makes at least two transitions before entering state q_m,if it enters q_m then it stays in q_m (and does not halt), and that $q_1 \neq q_m$.

A configuration of a 2CM is a triple (s,c1,c2), where $s \in Q$, and c1 and c2 are natural numbers. A computation of a 2CM M on input x is a sequence of configurations $\Sigma=\sigma_1,\sigma_2,\sigma_3,...$, where $\sigma_i=(s_i,c1_i,c2_i)$, and $s_1=q_1$, $c1_1=x$, $c2_1=0$, and for all $i \geq 1$,

$\delta^*(\sigma_i)=\sigma_{i+1}$, where δ^* is the natural extension of δ to configurations. M is said to accept x if for any configuration σ in the computation of M on x, $s=q_m$. For the purpose of this construction, each state will also be identified with some unique positive integer.

A.3 Proof

The configurations of the computation of M on input x will be encoded in the occurrence function by the addition of three new events, st,ctr1, and ctr2, which will be used to represent the state of the two counter machine, and the value stored in each counter. The encoding to accomplish this will be as follows:

$$\forall i>1 \;\; s_i = @(st,i) - @(st,i-1)$$
$$\forall i>1 \;\; c1_i = @(ctr1,i) - @(ctr1,i-1)-1$$
$$\forall i>1 \;\; c2_i = @(ctr2,i) - @(ctr2,i-1)-1$$

and $s_1=@(st,1)$, $c1_1=@(ctr1,1)-1$, and $c2_1=@(ctr2,1)-1$. As a result,

$$\forall i\geq 1 \;\; @(st,i)=\sum_{j=1}^{i} s_j$$
$$\forall i\geq 1 \;\; @(ctr1,i)=\sum_{j=1}^{i} (c1_j+1)$$
$$\forall i\geq 1 \;\; @(ctr2,i)=\sum_{j=1}^{i} (c2_j+1)$$

Since all of the s_j's are positive, and all of the $c1_j$'s and $c2_j$'s are non-negative, the occurrence function above is clearly monotonically increasing in its second argument. Lemmas 1 and 2 will show that the formula constructed in the proof of Theorem 1 does result in the above encoding.

Theorem 1: *For any 2CM M and input x, an RTL formula can be constructed that is unsatisfiable if and only if M accepts x.*

Proof: The configurations of the computation of M on input x will be encoded in the occurrence function as above. With this encoding in mind, the formula F is constructed of the following conjuncts, where $i \in \{1,2\}$ and i' is $3-i$:

(1) $@(st,1)=s_1 \;\wedge\; @(ctr1,1)=x+1 \;\wedge\; @(ctr2,1)=1 \;\wedge$
$@(st,2)=s_1+s_2 \;\wedge\; @(ctr1,2)=x+1+c1_2+1 \;\wedge\; @(ctr2,2)=1+c2_2+1$

(2) For each transition from state q_s to state q_r there is a subformula depending on the transition:

(a) ptest of counter i

$\forall t\ (@(st,t+1) - @(st,t) = q_s \wedge @(ctri,t+1) - @(ctri,t) > 1) \rightarrow$
$(@(st,t+2) = @(st,t+1) + q_r \wedge @(ctri,t+2) = 2*@(ctri,t+1) - @(ctri,t)$
$\wedge @(ctri',t+2) = 2*@(ctri',t+1) - @(ctri',t))$

fsp

(b) ztest of counter i

$\forall t\ (@(st,t+1) - @(st,t) = q_s \wedge @(ctri,t+1) - @(ctri,t) = 1) \rightarrow$
$(@(st,t+2) = @(st,t+1) + q_r \wedge @(ctri,t+2) = 2*@(ctri,t+1) - @(ctri,t)$
$\wedge @(ctri',t+2) = 2*@(ctri',t+1) - @(ctri',t))$

(c) dec of counter i

$\forall t (@(st,t+1) - @(st,t) = q_s) \rightarrow$
$(@(st,t+2) = @(st,t+1) + q_r \wedge @(ctri,t+2) = 2*@(ctri,t+1) - @(ctri,t) - 1$
$\wedge @(ctri',t+2) = 2*@(ctri',t+1) - @(ctri',t))$

(d) inc of counter i

$\forall t\ (@(st,t+1) - @(st,t) = q_s) \rightarrow$
$(@(st,t+2) = @(st,t+1) + q_r \wedge @(ctri,t+2) = 2*@(ctri,t+1) - @(ctri,t)+1$
$\wedge @(ctri',t+2) = 2*@(ctri',t+1) - @(ctri',t))$

(3) $\forall t\ @(st,t+1) - @(st,t) \neq q_m$.

Lemma 1: *If M does not accept x, then F is satisfiable*.

Proof: Let *I* be the standard model of Presburger arithmetic together with an assignment of a function to @ such that $\forall t @(st,t)=\sum_{j=1}^{t} s_j$, $@(ctr1,t)=\sum_{j=1}^{t}(c1_j+1)$, and $@(ctr2,t)=\sum_{j=1}^{t}(c2_j+1)$, and a consistent assignment to R. Since *I* is a model for Presburger arithmetic, it is only necessary to show that $I|=F$, which is done for each type of subformula of F.

(1) $I|=@(st,1)=s_1 \wedge @(ctr1,1)=x+1 \wedge @(ctr2,1)=1$, since $s_1=\sum_{j=1}^{1} s_j$, $x+1=\sum_{j=1}^{1}(c1_j+1)$, and $1=\sum_{j=1}^{1}(c2_j+1)$. Similarly, *I* satisfies the subformulas with t=2.

(2a) ptest for counter 1

Suppose the antecedent is satisfied by *I*, then $I|=@(st,t+1) - @(st,t)=q_s \wedge @(ctr1,t+1) - @(ctr1,t)>1$. By construction of @, $s_{t+1}=q_s$ and $c1_{t+1}>0$. Since M is deterministic, only one transition can be enabled in a given configuration, so this transition is the one that is to be taken, and because the transition is a ptest for counter 1, $s_{t+2}=q_r$, $c1_{t+2}=c1_{t+1}$, and $c2_{t+2}=c2_{t+1}$. Therefore, $\sum_{j=1}^{t+2} s_j=(\sum_{j=1}^{t+1} s_j)+q_r$, and @(st,t+2)=@(st,t+1)+q_r, again by construction of @. Likewise,

$$\sum_{j=1}^{t+2}(c1_j+1) = (\sum_{j=1}^{t+1}(c1_j+1)) + (c1_{t+1}+1)$$

$$=\sum_{j=1}^{t+1}(c1_j+1) + \sum_{j=1}^{t+1}(c1_j+1) - \sum_{j=1}^{t}(c1_j+1),$$

so @(ctr1,t+2)=2*@(ctr1,t+1)–@(ctr1,t). Therefore, *I* satisfies subformulas of this type.

A similar argument shows that *I* satisfies the other seven types of subformulas of type 2.

(3) Since *I* does not accept x, $\neg\exists t\ s_t=q_m$, therefore $I|=\forall t$@(st,t+1)–@(st,t)$\neq q_m$.
Since *I* satisfies all of the conjuncts of F, *I*|=F. □

Lemma 2: *If I|=F, then for all t,* $@(st,t)=\sum_{j=1}^{t} s_j$, $@(ctr1,t)=\sum_{j=1}^{t}(c1_j+1)$, *and* $@(ctr2,t)=\sum_{j=1}^{t}(c2_j+1)$.

Proof: By induction on t.

Base case (t=1 or t=2): Since *I*|=F, $@(st,1)=s_1=\sum_{j=1}^{1} s_j$, $@(ctr1,1)=x+1=\sum_{j=1}^{1}(c1_j+1)$, and $@(ctr2,1) = 1 = \sum_{j=1}^{1}(c2_j+1)$, because of subformula 1. The case where t=2 is similar.

Induction step (t= k+2): Assume the lemma holds for all t<k+2. Since $\sigma_{k+1}|\!\!-\!\!-_M\sigma_{k+2}$ is some transition that takes M from σ_{k+1} to σ_{k+2}, and it must be of one of the eight elementary types of transitions.

If σ_{k+2} results from σ_{k+1} by applying a ptest of counter 1, with states q_s and q_r,then $c1_{k+1}>0$, so $c1_{k+1}+1>1$. Therefore, $\sum_{j=1}^{k+1}(c1_j+1) - \sum_{j=1}^{k}(c1_j+1)>1$, and by the induction hypothesis, @(ctr1,k+1)–@(ctr1,k)>1. Likewise, since $s_{k+1}=q_s$, $\sum_{j=1}^{k+1}s_j-\sum_{j=1}^{k}s_j=q_s$, and @(st,k+1)–@(st,k)=$q_s$. Therefore, by the corresponding conjunct of type 2a of F, $@(st,k+2)=@(st,k+1)+q_r=\sum_{j=1}^{k+1}s_j+q_r=\sum_{j=1}^{k+2}s_j$, since $s_{k+2}=q_r$. Also,

$$
\begin{aligned}
@(\text{ctr1},k+2) &= 2*(\text{ctr1},k+1) - @(\text{ctr1},k) \\
&= 2*\sum_{j=1}^{k+1}(c1_j+1) - \sum_{j=1}^{k}(c1_j+1) \\
&= \sum_{j=1}^{k+1}(c1_j+1) + c1_{k+1}+1 \\
&= \sum_{j=1}^{k+1}(c1_j+1) + c1_{k+2}+1 \\
&= \sum_{j=1}^{k+2}(c1_j+1),
\end{aligned}
$$

since $c1_{k+1}=c1_{k+2}$. Similarly, $@(\text{ctr2},k+2)=\sum_{j=1}^{k+2}(c2_j+1)$. The same argument applies to the other seven basic transition types, and the lemma holds by induction.□

Lemma 3: *If F is satisfiable, then M does not accept x.*

Proof: Let *I* be a model for RTL, and *I*|=F. Suppose M accepts x. Then ∃t>2 such that $s_t=q_m$. From lemma 2, $@(st,t)–@(st,t–1)=q_m$, which is inconsistent with conjunct 3 of F. Therefore the assumption that M accepts x leads to a contradiction, and the lemma holds.□

Lemmas 1 and 3 establish the following theorem.

Theorem 2: *RTL is undecidable.*

Proof: This follows from theorem 1 and the result that the acceptance problem for two counter machines is undecidable. □

Appendix B -- Detailed Reactor Proof

The RTL formulas describing the system specification and the safety assertion for the reactor control rod example were presented in section 3.1. A detailed natural deduction proof of the safety assertion from the system specification is shown below. In the following proof, upper case letters denote constants and lower case letters denote variables.

1.	Show Safety Assertion	
2.	Show $R(\uparrow\text{MOVE1}, \text{I}, T_1) \wedge R(\downarrow\text{MOVE1}, \text{I}, T_2) \wedge$	
	$R(\uparrow\text{MOVE2}, \text{J}, T_3) \wedge R(\downarrow\text{MOVE2}, \text{J}, T_4)$	
	$\rightarrow T_4 < T_1 \vee T_2 < T_3$	
3.	a. $R(\uparrow\text{MOVE1}, \text{I}, T_1)$	ACP†
	b. $R(\downarrow\text{MOVE1}, \text{I}, T_2)$	ACP
	c. $R(\uparrow\text{MOVE2}, \text{J}, T_3)$	ACP
	d. $R(\downarrow\text{MOVE2}, \text{J}, T_4)$	ACP
	e. Show $T_4 < T_1 \vee T_2 < T_3$	
4.	a. $\exists t'\ R((\text{GRANT1:=T}), \text{I}, t') \wedge t' \le T_1 \wedge T_1 \le t' + 5$	$\forall$ Instantiation, 3.a, SP4', MP‡
	b. $R((\text{GRANT1:=T}), \text{I}, T_5) \wedge T_5 \le T_1 \wedge T_1 \le T_5 + 5$	$\exists$ Elimination, 4.a
5.	a. $\exists t'\ R((\text{GRANT1:=F}), \text{I+1}, t') \wedge T_5 + 30 \le t'$	$\forall$ Instantiation, 4.b, SP11, MP
	b. $R((\text{GRANT1:=F}), \text{I+1}, T_6) \wedge T_5 + 30 \le T_6$	$\exists$ Elimination, 5.a
6.	$T_1 + 25 \le T_6$	Transitivity Axiom, 4.b, 5.b
7.	a. $\exists t'\ R(\downarrow\text{MOVE1}, \text{I}, t') \wedge t' \le T_1 + 20$	$\forall$ Instantiation, 3.a, SP5, MP
	b. $R(\downarrow\text{MOVE1}, \text{I}, T_7) \wedge T_7 \le T_1 + 20$	$\exists$ Elimination, 7.a
	c. $T_2 \le T_1 + 20$	Occurrence Relation Ax, 3.b, 7.b
8.	$T_2 + 5 \le T_6$	Transitivity Axiom, 6, 7.c
9.	a. $R((\text{GRANT2:=T}), \text{I}, T_8) \wedge T_8 \le T_3 \wedge T_3 \le T_8 + 5$	similar to step 4.b
	b. $R((\text{GRANT2:=F}), \text{I+1}, T_9)$	similar to step 5.b
	c. $T_4 + 5 \le T_9$	similar to step 8
10.	$T_9 < T_5 \vee T_6 < T_8$	$\forall$ Inst, SP13, 4.b, 5.b, 9.a, 9.b, MP
11.	Show $T_6 < T_8 \rightarrow T_2 < T_3$	
	a. $T_6 < T_8$	ACP
	b. $T_6 < T_3$	Transitivity Axiom, 9.a, 11.a
	c. $T_2 + 5 < T_3$	Transitivity Axiom 11.b, 8
12.	$T_9 < T_5 \rightarrow T_4 < T_1$	Similar to proof in step 11
13.	$T_4 < T_1 \vee T_2 < T_3$	Disj Exploitation 10, 11, 12
14.	$\forall \text{i,j}\ \forall t_1, t_2, t_3, t_4\ R(\uparrow\text{MOVE1}, \text{i}, t_1) \wedge R(\downarrow\text{MOVE1}, \text{i}, t_2) \wedge$	
	$R(\uparrow\text{MOVE2}, \text{j}, t_3) \wedge R(\downarrow\text{MOVE2}, \text{j}, t_4)$	
	$\rightarrow t_4 < t_1 \vee t_2 < t_3$	Universal Generalization, 2

† Assumption for Conditional Proof

‡ Modus Ponens

[Barbacci & Wing 87]. M. Barbacci and J. Wing, ‘‘Specifying Functional and Timing Behavior for Real-Time Applications,’’ pp. 124-140 in *Lecture Notes in Computer Science 259*, Springer Verlag (1987).

Bibliography

[Bernstein & Harter 81]. A. Bernstein and P.K. Harter, ''Proving Real-Time Properties of Programs with Temporal Logic,'' *Proc. 8th Symposium on Operating System Principles, ACM SIGOPS,* pp. 1-11 (1981).

[Cooper 72]. D.C. Cooper, ''Theorem Proving in Arithmetic without Multiplication,'' in *Machine Intelligence 7,* ed. D. Michie,Edinburgh Univ. Press, Edinburgh (1972).

[Downey 72]. Peter J. Downey, ''Undecidability of Presburger Arith. with a Single Monadic Predicate Letter,'' Center for Research in Computing Technology 18-72, Harvard University (1972).

[Heninger 80]. K.L. Heninger, ''Specifying Software Requirements for Complex Systems: New Techniques and their Application,'' *IEEE Transactions on Software Engineering,* **SE-6**(1) pp. 2-13 (Jan. 1980).

[Hopcroft & Ullman 79]. J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, Reading, Mass (1979).

[Jahanian & Mok 86a]. F. Jahanian and A.K. Mok, ''Safety Analysis of Timing Properties in Real-Time Systems,'' *IEEE Transactions on Software Engineering* **SE-12**(9) pp. 890-904 (Sept. 1986).

[Jahanian & Mok 87]. F. Jahanian and A.K. Mok, ''A Graph-Theoretic Approach for Timing Analysis and its Implementation,'' *IEEE Transactions on Computers* **C-36**(8) pp. 961-975 (August 1987).

[Jahanian et al 88b]. F. Jahanian, R.S. Lee, and A.K. Mok, ''Semantics of Modechart in Real Time Logic,'' *Proc. 21st Hawaii International Conference on System Sciences*, (Jan. 1988.).

[Lamport 83]. L. Lamport, ''What Good is Temporal Logic?,'' pp. 657-668 in *Proceeding of IFIP, Information Processing*, ed. R.E.A. Mason,,North-Holland (1983).

[Leveson & Harvey 83]. N.G. Leveson and P.R. Harvey, ''Analyzing Software Safety,'' *IEEE Transactions on Software Engineering* **SE-9**(5) pp. 569-579 (Sept. 1983).

[Leveson 86]. N. G. Leveson, ''Software Safety: What, Why, and How,'' *Computing Surveys* **18**(2) pp. 125-163 (June 1986).

[MacEwen 87]. G.H. MacEwen, ''Using Higher-Order Logic for Modular Specification of Real-Time Distributed Systems,'' CS Lab, SRI International, Menlo Park, CA (1987).

[Merlin & Farber 76]. P.M. Merlin and D.J. Farbar, ''Recoverability of Communication Protocols-Implications of a Theoretical Study,'' *IEEE Trans. on Communications*, pp. 1036-43 (Sept. 1976).

[Mok 85]. A.K. Mok, ''SARTOR-a Design Environment for Real-Time Systems,'' *Proc. 9th IEEE COMPSAC*, pp. 174-181 (Oct. 1985).

[Ostroff & Wonham 87]. J. S. Ostroff and W. M. Wonham, ''Modelling, Specifying and Verifying Real-time Embedded Computer Systems,'' *Proceedings of the Real Time Systems Symposium*, pp. 124-32 (December, 1987).

[Pnueli et al 85]. Amir Pnueli, Orna Lichtenstein, and Lenore Zuck, ''The Glory of the

Bibliography

Past,'' pp. 196-218 in *Lecture Notes on CS: Workshop on Logics of Programs*, Springer-Verlag (1985).

[Pnueli 81]. A. Pnueli, ''The Temporal Semantics of Concurrent Programs,'' *Theoretical Computer Science* **13** pp. 45-60 (1981).

[Ramchandani 74]. C. Ramchandani, ''Analysis of Asynchronous Concurrent Systems by Timed Petri Nets,'' TR 120, Project MAC, MIT (Feb. 1974).

[Razouk & Phelps 84]. R.R. Razouk and C.V. Phelps, ''Performance Analysis of Timed Petri Nets,'' *Proc. of 4th International Workshop on Protocol Spec, Verf and Testing*, (June 1984).

[Shankar & Lam 87]. A.U. Shankar and S.S. Lam, ''Time-Dependent Distributed Systems: Proving Safety, Liveness and Real-Time Properties,'' *Distributed Computing*, (2) pp. 61-79 Springer-Verlag, (1987).

[Shostak 79]. R.E. Shostak, ''A Practical Decision Procedure for Arithmetic with Function Symbols,'' *JACM,* **26**(2) pp. 350-361 (April 1979).

[Zuberek 1980]. W.M. Zuberek, ''Timed Petri Net and Preliminary Performance Evaluation,'' *7th Annual Symposium on Computer Architecture*, pp. 88-96 (1980).