Equivalence Preserving Transformations for Timed Transition Models

Mark Lawford and W. M. Wonham

Abstract— **The increasing use of computer control systems in safety-critical real-time systems has led to a need for methods to ensure the correct operation of real-time control systems. Through an example, this paper introduces the use of algebraic equivalence to verify the correct operation of such systems. A controller is considered verified if its implementation is proven to be equivalent to its specification. Real-time systems are modeled using a modified version of Ostroff's Timed Transition Models (TTMs), which is introduced along with our adaptation of Milner's observation equivalence to TTMs. A set of "behavior preserving" transformations is then developed, shown to be consistent for proving observation equivalence, and then applied to solve an industrial real-time controller software verification problem. Finally the incompleteness of a given set of transformations is briefly discussed.**

I. Introduction

The goal of this paper is to provide a visual framework for the development and verification of real-time systems that is simple to use yet is based on a sound logical foundation. The software failures catalogued by Lee [1] clearly point to the need for methods to ensure the correct operation of safety critical systems, while the effectiveness of a visual approach to software verification has been argued by Harel [2]. In response to the need for formal methods with visual appeal, Ostroff et al. have introduced Timed Transition Models (TTMs) [3],[4]; but the Real-Time Temporal Logic on which the proof system is based can be difficult for control and software practitioners to master and tends to lack a visual component. Also, no method was provided for moving between levels of abstraction of real-time models to allow behavioral comparison of two TTMs: such flexibility would enable one to project out extraneous behavior to obtain high-level TTM models or, conversely, to refine high level TTM specifications into workable implementations.

Formal software verification has been studied by computer specialists for over a decade. For instance, process algebras [5], [6], [7] provide equational methods of modeling concurrent systems along with algebraic laws for equational transformation. While incorporating abstraction, the cited methods do not explicitly express time or provide visual representations. Real-time extensions of [7] such as [8],[9] lack the abstracting power of a congruence relation, like weak observation congruence [7], which permits interchange of congruent subsystems and therefore the combination of abstraction with modularity. A version of weak observation congruence is developed in [10] and proven correct for a real-time setting where each event is assigned an exact (real-valued) time of occurrence. However, the adoption of $\mathbf{R}^+$ as the time set significantly complicates the algebraic laws used to establish system equivalence, owing to the need for transition structures with uncountable numbers of transitions. By contrast, the TTM setting uses a discrete time model (in which the clock 'ticks'), and assigns to events a range of possible occurrence times within the clock resolution. The advantage is that weak observation equivalence can be retained, and applied in a straightforward way.

A visual method was introduced in [11] for checking the reachability of a class of extended timed Petri nets. The net approach and the provision of net transformations led to a graph-based method of verification. As in [3] the problem of constructing equivalent abstract real-time systems from a given system model was not considered. In the same spirit as [11], we develop in this paper a set of easily applicable, and demonstrably correct, transformations that preserve system equivalence, and lend themselves to abstraction, in the setting of TTMs. The proof that the transformations preserve observation equivalence relies on the properties of observation equivalence as described in [7] but many of the TTM transformations have no direct analog in Milner's process algebra setting. Also, a software engineer need not be familiar with observation equivalence or process algebra to be able to use the simple set of visual transformations to prove that a system correctly implements its specification in a well defined way.

Section II introduces a version of TTMs, while Section III defines TTM equivalence. A set of behavior-preserving transformations is developed in Section IV and shown to be consistent for proving TTM observation equivalence. An application to a small real-world example is presented in Section V; and the issue of incompleteness of given sets of transformations briefly addressed in Section VI.

II. Timed Transition Models (TTMs)

In this section we introduce a modified version of the Timed Transition Models (TTMs) employed in [4]. We drop the Real Time Temporal Logic (RTTL) assertion language, though we still use the infinite string semantics it required. Also, we do not consider systems explicitly composed of subsystems that interact via communication channels or synchronized transitions. As a result, our definition of TTMs will not include the communication channels or the parallel composition operator of Ostroff's original definition. To simplify the problem of equivalence verification, the initial condition is limited to specifying a unique initial

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

The authors are with the Department of Electrical & Computer Engineering, University of Toronto, Toronto, Ontario, Canada M5S 1A4. E-mail: lawford or wonham@control.utoronto.ca .

state instead of (possibly) multiple initial states.

A *Timed Transition Model* (TTM) M is a triple given by

$$M := (\mathcal{V}, \Theta, \mathcal{T})$$

where $\mathcal{V}$ is a set of variables, Θ is an initial condition (a boolean-valued expression in the variables), and $\mathcal{T}$ is a finite set of transitions.

$\mathcal{V}$ always includes two special variables: the global time variable t and an activity variable which we will usually denote by x. For $v \in \mathcal{V}$ the range space of v is $Range(v)$ (eg. $Range(t) = \mathbf{N}$ where $\mathbf{N} := \{0, 1, 2, \ldots\}$). We define $\mathcal{Q}$, the set of *state assignments of* M, to be the product of the ranges of the variables in $\mathcal{V}$. That is

$$\mathcal{Q} := \times_{v_i \in \mathcal{V}} Range(v_i)$$

For a state assignment $q \in \mathcal{Q}$ and a variable $v \in \mathcal{V}$, we will denote the value of v in state assignment q by $q(v)$ where $q(v) \in Range(v)$.

$\mathcal{T}$ is the transition set. A transition α is a 4-tuple

$$\alpha := (e_\alpha, h_\alpha, l_\alpha, u_\alpha)$$

where e_α is the transition's enablement condition (a boolean valued expression in the variables of $\mathcal{V}$), h_α is the operation function, and $l_\alpha \in Range(t) = \mathbf{N}$ and $u_\alpha \in \mathbf{N} \cup \{\infty\}$ are the lower and upper time bounds respectively with $l_\alpha \leq u_\alpha$. We say that α is *enabled* when $q(e_\alpha) = true$. The operation function $h_\alpha : \mathcal{Q} \rightarrow \mathcal{Q}$ is a partial function, defined when $q(e_\alpha) = true$, that maps the current state assignment to the new state assignment when the transition occurs. $\mathcal{T}$ always contains the special transition *tick*,

$$tick := (true, [t : t+1], -, -)$$

which represents the passage of time on the global clock. *tick* is the only transition that affects the time variable t and also has no lower or upper time bound. All other transition time bounds are given relative to numbers of occurrences of *tick*.

Θ is the initial condition, a boolean valued expression in the variables of $\mathcal{V}$ that is used to identify a unique initial state of the system.

A. TTM Semantics

A *trajectory* of a TTM is any infinite string of the TTM state assignments connected by transitions, of the form $q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \xrightarrow{\alpha_2} \ldots$. The interpretation is that q_i goes to q_{i+1} via the transition α_i. A state trajectory $\sigma := q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \xrightarrow{\alpha_2} \ldots$ is a *legal* trajectory of a TTM M if it meets the following four requirements:

1. **Initialization:** The initial state assignment satisfies the initial condition ($q_0(\Theta) = true$ - ie. q_0 satisfies Θ and hence is the unique initial state assignment).
2. **Succession:** For all i, q_{i+1} is obtained from q_i by applying the operation function of α_i ($q_{i+1} = h_{\alpha_i}(q_i)$) and α_i is enabled in state assignment q_i (ie. $q_i(e_{\alpha_i}) = true$).
3. **Ticking:** The clock must *tick* infinitely often. That is, there are an infinite number of transitions $\alpha_i = tick$. This eliminates the possibility of "clock stoppers" in the trajectory where an infinite number of non-*tick* transitions occur consecutively without being interleaved with any *tick*s. This would imply that the TTM is performing an infinite number of actions in a finite time.
4. **Time Bounds:** To determine if the trajectory σ satisfies the time bound requirements of the TTM M, we associate with each non-*tick* transition α, a counter variable c_α with $Range(c_\alpha) = \mathbf{N}$. Each α transition's counter is initially set to zero and is reset to zero after an α transition or a transition that enters a new state assignment where α is disabled (ie. $e_\alpha = false$). The counter is only incremented by the occurrence of a *tick* transition when α is enabled ($e_\alpha = true$). Any non-*tick* transition α can legally occur only when when its counter is in the region specified by the transition's time bounds (ie. $l_\alpha \leq c_\alpha \leq u_\alpha$). The upper time bounds on transitions represent hard time bounds by which time the transitions are guaranteed to occur. Thus if α's counter reaches its upper time bound, then it is forced to occur before the next tick of the clock unless it is preempted by another non-*tick* transition that disables α (and hence resets α's counter). Hence for a *tick* transition to legally occur, every enabled transition α must have a counter value less than its upper time bound ($c_\alpha < u_\alpha$). We now formalize the above description.

   For the TTM $M := (\mathcal{V}, \Theta, \mathcal{T})$, we will denote the set of transition counters by $C := \{c_\alpha : \alpha \in \mathcal{T} - \{tick\}\}$. From the trajectory σ we derive the *full trajectory* $\bar{\sigma} := \bar{q}_0 \xrightarrow{\alpha_0} \bar{q}_1 \xrightarrow{\alpha_1} \bar{q}_2 \xrightarrow{\alpha_2} \ldots$, where each $\bar{q}_i \in \mathcal{Q} \times \mathbf{N}^C$ is obtained from σ as follows:

   For all $v \in \mathcal{V}$, $\bar{q}_i(v) = q_i(v)$.

   For all $c_\alpha \in C$, $\bar{q}_0(c_\alpha) = 0$ and for $i = 0, 1, 2, \ldots$

$$\bar{q}_{i+1}(c_\alpha) = \begin{cases} \bar{q}_i(c_\alpha) + 1, \text{ if } q_i(e_\alpha) = true \ \&\ \alpha_i = tick \\ 0, \text{ if } q_{i+1}(e_\alpha) = false \text{ or } \alpha_i = \alpha \\ \bar{q}_i(c_\alpha), \text{ otherwise} \end{cases}$$

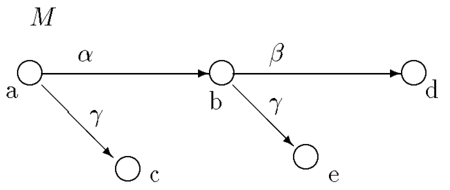
   The trajectory σ satisfies the time bounds of M iff the following two conditions hold in $\bar{\sigma}$ for all $i = 0, 1, \ldots$:

   (i) $\alpha_i = tick$ iff for all $\alpha \in \mathcal{T} - \{tick\}$, $q_i(e_\alpha) = true$ implies $\bar{q}_i(c_\alpha) < u_\alpha$.

   (ii) $\alpha_i = \alpha$, $\alpha \in \mathcal{T} - \{tick\}$ iff $l_\alpha \leq \bar{q}_i(c_\alpha) \leq u_\alpha$.

A condition equivalent to (i) is that for all $c_\alpha \in C$, $\bar{q}_i(c_\alpha) \leq u_\alpha$. Note that any loop of transitions in a TTM (a sequence of transitions starting and ending in the same activity) must have at least one transition with a non-zero upper time bound. Otherwise, once the first transition of the loop is enabled, our transition rules could possibly force an infinite number of non-*tick* transitions to occur without being interleaved by an infinite number of *tick*s.

As a small example, consider the TTM $M := (\mathcal{V}, \Theta, \mathcal{T})$ shown in Figure 1. The full enablement conditions for the



$$
\begin{aligned}
\mathcal{V} &:= \{u, v, t, x\} \\
\Theta &:= u = 0 \wedge v = 1 \wedge x = a \\
\mathcal{T} &:= \{\alpha, \beta, \gamma, tick\}
\end{aligned}
$$

where

$$
\begin{aligned}
\alpha &:= (u \geq 0, [u : u + v], 0, 2) \\
\beta &:= (true, [u : u + 1, v : v - 1], 2, \infty) \\
\gamma &:= (v \geq 0, [\,], 2, 2)
\end{aligned}
$$

Fig. 1. An example of a simple TTM

transitions should also include conditions that enable the transitions only when the TTM is in activities that they exit in the transition diagram. For instance in the case of γ, the full enablement condition is $e_\gamma := u \geq 0 \wedge (x = a \vee x = b)$. When describing TTM transitions we will usually omit these activity variable conditions since they are obvious from the transition diagram. From the above discussion it is apparent that the definition of a transition such as $\gamma \in \mathcal{T}$ can result in several arrows with the same label in a TTM transition graph. To allow us to distinguish between a transition and the arrows that it defines in a transition diagram, we will call the arrows in the transition diagram *instances* of the transitions they are labeled by. In the example TTM M, there is an instance of transition γ exiting activity a and another instance exiting activity b. Finally, the special transition $tick$ is declared to be in $\mathcal{T}$.

In writing out the operation functions of the transitions of M we employ Ostroff's assignment format. When a transition occurs, the new value of the activity variable x is obtained from the transition diagram. The other variables that are affected by the transition are listed in the form $[v_1 : expr_1, v_2 : expr_2, \ldots, v_n : expr_n]$ with the interpretation that variables v_1 to v_n are assigned the new values given by the simultaneous evaluations of expressions $expr_1$ to $expr_n$ respectively. The operation function acts as the identity on variables not listed in the assignment statement. For instance $h_\alpha := [u : u + v] = [u : u + v, v : v]$ for M above.

If we let the current state assignment be represented by a 4-tuple of the form (u, v, x, t), then a legal trajectory of M would be

$$q_0 \xrightarrow{tick} q_1 \xrightarrow{\alpha} q_2 \xrightarrow{tick} q_3 \xrightarrow{\gamma} q_4 \xrightarrow{tick} \ldots$$

$$(0, 1, a, 0) \xrightarrow{tick} (0, 1, a, 1) \xrightarrow{\alpha} (1, 1, b, 1) \xrightarrow{tick} (1, 1, b, 2) \xrightarrow{\gamma} (1, 1, e, 2)$$

where from q_4 onward the trajectory is continued by an infinite string of $tick$s. Note that after the second occurrence of $tick$, γ is forced to occur. A $tick$ could not take place from q_3 since γ has $u_\gamma = 2$ and, upon reaching q_3, e_γ has been true for two $tick$s already.

If the initial condition for M is $\Theta := (u = 0 \wedge v = -1 \wedge x = a)$, then a trajectory that by the above definition is "legal" is

$$(0, -1, a, 0) \xrightarrow{\alpha} (-1, -1, b, 0) \xrightarrow{tick} (-1, -1, b, 1) \xrightarrow{tick} (-1, -1, b, 2)$$

where again this trajectory is continued by an infinite number of $tick$ transitions. This trajectory illustrates our interpretation of $u_\beta = \infty$. We do not insist on "fairness", allowing trajectories such as the one above where β is a possible next transition for an infinitely long time, although it does not occur. Thus an upper time bound of ∞ means that a transition is possible but is not forced to occur in a legal trajectory.

Occasionally we will use the transition graph representation of a TTM, where each instance of a transition in the TTM is represented as shown in Figure 2. This can be

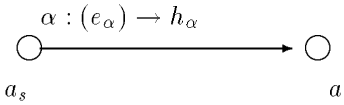


Fig. 2. The transition graph format of a TTM

informally interpreted as follows: "if the TTM is currently in activity a_s and if e_α evaluates to $true$, then the edge labeled by α may be traversed while doing operation h_α, after which the TTM is in activity a_d." We will usually use this style of displaying TTMs when the time bounds are understood or not of particular importance to the discussion.

III. Equivalence of TTMs

In this section Labeled Transition Systems (LTS) are introduced to describe the behavior of TTMs. LTS allow us to develop a notion of equivalence for TTMs.

A. Labeled Transition Systems and TTMs

LTS have been used by De Nicola [12] to compare different notions of equivalence proposed for concurrent systems. A notion similar to LTS forms the basis of TTMs and many other models of concurrency. We now borrow some of the definitions and notation of [12].

Definition 1: A **Labeled Transition System** is a 4-tuple $(A, \Sigma, \xrightarrow{\mu}, a_0)$ where A is an at most countable set of states, Σ is an at most countable set of elementary actions, $\xrightarrow{\mu}$ with $\mu \in \Sigma \cup \{\tau\}$, is a set of binary relations on A and $a_0 \in Q$ is the initial state.

In the above definition if $\alpha \in \Sigma$ and $a, a' \in A$, then $a \xrightarrow{\alpha} a'$ means that the LTS can move from state a to a' by executing elementary action α. Following the notation of [6] and [7], the special symbol τ, $\tau \notin \Sigma$, is used to denote *internal* (unobservable) actions. Hence $a \xrightarrow{\tau} a'$ means that the system can move from a to a' via an unobservable or *silent* transition. In the graphical representation of an LTS, the binary relations $\xrightarrow{\mu}$ are represented by arrows connecting related states. The arrows represent the possible "transitions" the system could make when actions occur, and are labeled by the corresponding actions. For example an LTS with $a \xrightarrow{\alpha} a'$ would have two states a and a' connected by an arrow from a to a' labeled "α". The set of elementary actions $\Sigma \cup \{\tau\}$ forms the set of transition labels in the graphical representation.

The following notation is also helpful:

Σ denotes the set of visible actions.

Σ^* denotes the set of finite strings of visible actions.

$\Sigma_\tau := \Sigma \cup \{\tau\}$ and similarly Σ_τ^* denotes the set of finite strings over Σ_τ.

$a \xrightarrow{s} a'$ where $s = \mu_1\mu_2\dots\mu_k \in \Sigma_\tau^*$ denotes $(\exists a_1,\dots,a_{k-1} \in A)\ a \xrightarrow{\mu_1} a_1 \xrightarrow{\mu_2} \dots a_{k-1} \xrightarrow{\mu_k} a'$ and $a \xrightarrow{s}$ will mean $(\exists a' \in A) a \xrightarrow{s} a'$.

$a \overset{s}{\Rightarrow} a'$ where $s = \mu_1\mu_2\dots\mu_k \in \Sigma_\tau^*$ means: $(\exists a_1,\dots,a_k \in A)$

$$a \xrightarrow{\tau^{n_1}\mu_1} a_1 \xrightarrow{\tau^{n_2}\mu_2} a_2 \xrightarrow{\tau^{n_3}\mu_3} \dots a_{k-1} \xrightarrow{\tau^{n_k}\mu_k} a_k \xrightarrow{\tau^{n_{k+1}}} a'$$

where $n_i \geq 0,\ 1 \leq i \leq k+1$. The idea behind the relation $\overset{s}{\Rightarrow}$ is that the system can move from a to a' while executing the actions $\mu_1, \mu_2, \dots, \mu_k$ interleaved with internal τ actions. As before we will write $a \overset{s}{\Rightarrow}$ as a short form for $(\exists a' \in A) a \overset{s}{\Rightarrow} a'$.

One of the operations of [7] that we will find useful is that of relabeling an LTS. In this operation the structure of an LTS is left unaltered while the transition labels are changed in a consistent way. That is, if one instance of a label is changed to a new label, then all instances of the label must be changed to the same new label in the relabeled LTS.

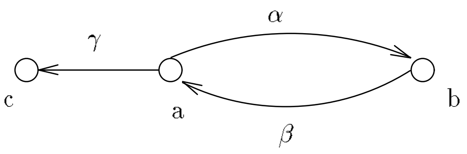
Definition 2: Let r be a function from transition labels to transition labels and $T := (A, \Sigma, \xrightarrow{\mu}, a_0)$ be an LTS. Then the r **relabeling of** T is given by:

$$r(T) := (A, \{r(\alpha) : \alpha \in \Sigma\}, \xrightarrow{r(\mu)}, a_0)$$

We now consider T_M, the Labeled Transition System generated by a TTM $M := (\mathcal{V}, \Theta, \mathcal{T})$. There are many possible LTS that represent the legal trajectories of a given TTM but for simplicity we adopt tree structures with all possible next transitions exiting the current LTS state to new LTS states. It is often the case that the transition names are unimportant. What is important is the effect the transitions have upon the variables of interest and how the latter affect the ordering of transitions. Accordingly the event labels of T_M are the actual operation functions of the TTM transitions. We will see in the example below that h_α (where $h_\alpha = [w : w+1, y : y+z]$) is written in T_M when transition α occurs in the legal trajectory of M. A convenient state set for T_M is the set of all finite strings of transitions $\mathcal{T}^*$. We then let the initial state of T_M be ϵ, the empty string. The transition relations follow naturally by defining for any $s \in \mathcal{T}^*$, $s \xrightarrow{h_\alpha} s\alpha$ if, starting from its unique initial state assignment, M can perform the transitions $s\alpha$ as the initial sequence of transitions of a legal trajectory. More formally, for a TTM $M := (\mathcal{V}, \Theta, \mathcal{T})$ we have $T_M := (\mathcal{T}^*, \{h_\alpha : \alpha \in \mathcal{T}\}, \xrightarrow{h_\alpha}, \epsilon)$.

As we have defined them, each TTM has a unique initial state and the operation functions of the TTM's transitions are deterministic. Thus the effect on a TTM's variables can be completely determined by knowing the sequence of transitions that has taken place. This is what will allow us to compare the behavior of TTMs by comparing forms of the LTS that they generate. From now on the LTS representing the behavior of a TTM will be the LTS T_M as described above.

Consider M, the simple TTM of Figure 3. The LTS



$$\begin{aligned}
\Theta &:= x = \mathrm{a} \wedge v = w = y = z = 0 \\
\alpha &:= (w = 0, [w : w+1, y : y+z], 0, 1) \\
\beta &:= (true, [w : w-1, z : z-1], 0, 0) \\
\gamma &:= (w = 0 \wedge y \leq 0, [w : -1, v : v+1], 1, 2)
\end{aligned}$$

Fig. 3. Simple TTM $M := (\mathcal{V}, \Theta, \mathcal{T})$

representing the behavior of M, which we denote by T_M, is shown in Figure 4. Note that the *tick* transitions of the clock have been included in T_M and that at each state all legal continuations of the trajectory are possible. The self-looped h_{tick} transition at the end of some paths is for display purposes only and helps indicate that the path can only be continued by an infinite string of *tick*s.

We now consider the restriction of an LTS (representing the behavior of a TTM) to a subset of variables of interest. We need some preliminary definitions.

Definition 3: For a TTM M with variable set $\mathcal{V}$ and a subset of variables $\mathcal{U} \subset \mathcal{V}$, we define the **state assignments over** $\mathcal{U}$, denoted by $\mathcal{Q}_\mathcal{U}$, to be the product of the ranges of the variables in $\mathcal{U}$. Hence

$$\mathcal{Q}_\mathcal{U} := \times_{v_i \in \mathcal{U}} Range(v_i)$$

The **natural projection** $P_\mathcal{U} : \mathcal{Q} \to \mathcal{Q}_\mathcal{U}$ maps a state assignment to its corresponding state assignment over $\mathcal{U}$.

Definition 4: Suppose $M := (\mathcal{V}, \Theta, \mathcal{T})$ is a TTM, $\mathcal{U} \subset \mathcal{V}$ is a set of variables, and $\alpha \in \mathcal{T}$ is a transition. Let $h_\alpha : \mathcal{Q} \to \mathcal{Q}$ be the operation function of α and $P_\mathcal{U} : \mathcal{Q} \to \mathcal{Q}_\mathcal{U}$ be the natural projection from the state assignments $\mathcal{Q}$ to $\mathcal{Q}_\mathcal{U}$, the state assignments over $\mathcal{U}$. Then the **map induced in** $\mathcal{Q}_\mathcal{U}$ **by** h_α, when it exists, is the map $\overline{h_\alpha} : \mathcal{Q}_\mathcal{U} \to \mathcal{Q}_\mathcal{U}$ such that $P_\mathcal{U} \circ h_\alpha = \overline{h_\alpha} \circ P_\mathcal{U}$.

The relationship between h_α and $\overline{h_\alpha}$ is illustrated in the commutative diagram, Figure 6.

For a given $\mathcal{U}$, $\overline{h_\alpha}$ will exist if the operations of h_α upon the elements of $\mathcal{U}$ are independent of the values of the variables in $\mathcal{V} - \mathcal{U}$. For instance with $h_\alpha := [w : w+1, y : y+z] = [w : w+1, y : y+z, z : z]$ and $\mathcal{U} = \{y, z\}$ we have $\overline{h_\alpha} = [y : y+z]$. Note that $\overline{h_\alpha}$ is not defined for $\mathcal{U} = \{w, y\}$ since the new value of y depends upon the current value of z. The existence condition for $\overline{h_\alpha}$ can be formally stated as the mapping kernel condition $\ker(P_\mathcal{U}) \subseteq \ker(P_\mathcal{U} \circ h_\alpha)$.

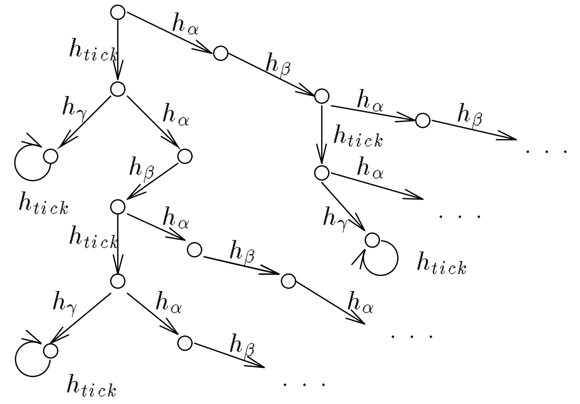


Fig. 4. T_M - the LTS reachability tree for M

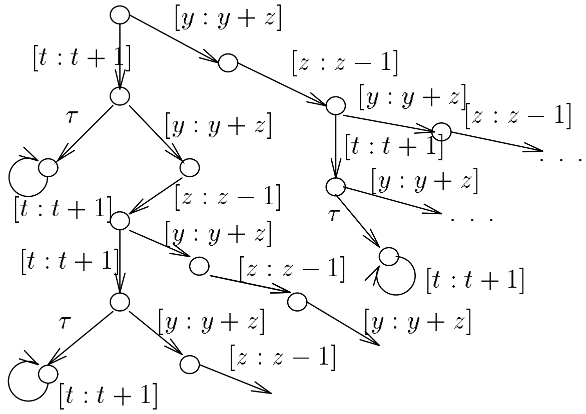


Fig. 5. $T_M|\{y,z\} = r(T_M)$ the restricted LTS for M

$$\begin{tikzcd} \mathcal{Q} \arrow[r, "h_\alpha"] \arrow[d, "P_{\mathcal{U}}"'] & \mathcal{Q} \arrow[d, "P_{\mathcal{U}}"] \\ \mathcal{Q}_{\mathcal{U}} \arrow[r, dashed, "\overline{h_\alpha}"] & \mathcal{Q}_{\mathcal{U}} \end{tikzcd}$$

Fig. 6. Commutative Diagram for Induced Operation Function

We now have the machinery to define the timed behavior of a TTM M restricted to a subset of its variables.

Definition 5: For $M := (\mathcal{V}, \Theta, \mathcal{T})$, $\mathcal{U} \subset \mathcal{V}$ and $T_M := (\mathcal{T}^*, \{h_\alpha : \alpha \in \mathcal{T}\}, \xrightarrow{h_\alpha}, \epsilon)$ we define the **restriction of T_M to $\mathcal{U}$** as follows. Let r be the LTS relabeling function such that $r(h_\alpha) = \overline{h_\alpha}$ where $\overline{h_\alpha}$ is the map induced in $\mathcal{Q}_{\mathcal{U}'}$ by h_α when $\mathcal{U}' := \mathcal{U} \cup \{t\}$. Then

$$T_M|\mathcal{U} := r(T_M) = (\mathcal{T}^*, \{r(h_\alpha) : \alpha \in \mathcal{T}\}, \xrightarrow{r(h_\alpha)}, \epsilon)$$

We then denote the **timed behavior of M restricted to $\mathcal{U}$** by $M|\mathcal{U} := T_M|\mathcal{U}$.
Note that $T_M|\mathcal{U}$ is defined iff $(\forall \alpha \in \mathcal{T})$ $\overline{h_\alpha}$ exists. When $T_M|\mathcal{U}$ is defined we say that $\mathcal{U}$ is *restrictable* for M.

If the variables of interest for the TTM M of Figure 1 are $\mathcal{U} = \{y, z\}$ (and implicitly t to guarantee the timing) then the LTS of the behavior of M over these variables can be obtained by replacing the transitions' operation functions with their induced maps. For example we replace h_α in T_M with $\overline{h_\alpha} := [y : y+z]$. In the case of the transition γ, $\overline{h_\gamma} := [\,]$ the identity or 'silent' function for $\{y, z, t\}$. $T_M|\{y,z\}$, the restriction of T_M as described above, is shown in Figure 5. Here we replace h_γ with the silent transition τ to help it stand out in the graph. Starting from the initial state of $T_M|\{y,z\}$, if the first transition is a clock tick, the next event may be y changing to $y+z$ or the system moving unobservably via τ to a state where no further changes can be made to $\{y, z\}$.

The example of Figure 5 illustrates how restriction can create systems that can move unobservably to a deadlocking state - a state with only strings of *tick*s as possible legal continuations. In the example of Section V we shall use a notion of equivalence that can distinguish between a deadlocking and a non-deadlocking system.

The main purpose of looking at the LTS generated by a TTM is to develop a notion of equivalence for TTMs. We will consider two TTMs to be equivalent over a set of variables $\mathcal{U}$ if their initial states agree on all variables in $\mathcal{U}$ and their respective LTS are equivalent when restricted to the variables of interest. More formally:

Definition 6: Given two TTMs $M_1 := (\mathcal{V}_1, \Theta_1, \mathcal{T}_1)$ and $M_2 := (\mathcal{V}_2, \Theta_2, \mathcal{T}_2)$ and EQ, an equivalence relation over the set of all LTS. Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be the sets of state assignments for M_1 and M_2 and $P_1 : \mathcal{Q}_1 \to \mathcal{Q}_{\mathcal{U}'}$ and $P_2 : \mathcal{Q}_2 \to \mathcal{Q}_{\mathcal{U}'}$ be their respective natural projections, for some $\mathcal{U}$, a set of variables. We say that M_1 **is EQ equivalent over $\mathcal{U}$ to** M_2, written $M_1\ EQ/\mathcal{U}\ M_2$, if and only if

(i) If $q_1 \in \mathcal{Q}_1$ and $q_2 \in \mathcal{Q}_2$ then $q_1(\Theta_1) = true$ and $q_2(\Theta_2) = true$ implies $P_1(q_1) = P_2(q_2)$

(ii) $T_{M_1}|\mathcal{U}\ EQ\ T_{M_2}|\mathcal{U}$

where T_{M_1} and T_{M_2} are the LTS generated by M_1 and M_2 respectively.
In practice usually $\mathcal{U} \subset \mathcal{V}_1 \cap \mathcal{V}_2$ though this need not be the case in general. The first condition in the definition guarantees that the systems start out in state assignments that are identical when restricted to $\mathcal{U}$ while the second condition guarantees that observed changes to variables in $\mathcal{U}$ will be equivalent.

B. Observation Equivalence

Reducing the problem of TTM equivalence to one of LTS equivalence allows us to choose from the multitude of LTS equivalence relations in [12]. For deadlock avoidance and other control properties described in [13], we will use Milner's observation equivalence (see [7]). To properly define observation equivalence we need an operator on Σ_τ^* that projects out all occurrences of τ. We denote this projection operator by$\hat{}: \Sigma_\tau^* \to \Sigma^*$. For example if $s = \alpha\tau\beta$ then $\hat{s} = \alpha\beta$. Recalling that $a \overset{\mu}{\Rightarrow} a'$ denotes $a \xrightarrow{\tau^i \mu \tau^j} a'$ for some $i, j \in \mathbf{Z}^+$, we now give the definition of Milner's observation equivalence.

Definition 7: Let $T_1 := (A, \Sigma, \xrightarrow{\mu}, a_0)$ and $T_2 := (B, \Sigma, \xrightarrow{\mu}, b_0)$ be LTS. A relation $S \subseteq A \times B$ is a **weak bisimulation** if $(a, b) \in S$ implies for all $\mu \in \Sigma_\tau$,

(i) Whenever $a \xrightarrow{\mu} a'$ then $(\exists b' \in B)\ b \stackrel{\widehat{\mu}}{\Rightarrow} b'$ and $(a', b') \in S$.

(ii) Whenever $b \xrightarrow{\mu} b'$ then $(\exists a' \in A)\ a \stackrel{\widehat{\mu}}{\Rightarrow} a'$ and $(a', b') \in S$.

In other words, two states $a \in A$, $b \in B$, are weakly bisimilar if any move from a to a new state a' can be matched by a finite sequence of moves from b, that produces the same observation and leads to a state b' that is weakly bisimilar to a'. Also, any move from b must be matched in a similar fashion. From [7] we know that the set of weak bisimulation relations over $A \times B$ is closed under union and thus there is always a largest weak bisimulation relation $\approx$, relating the states of T_1 and T_2. That is

$$\approx := \cup\{S | S \text{ is a weak bisimulation over } A \times B\}$$

We write $a \approx b$ when $(a, b) \in \approx$.

We can now formally define *observation equivalence* for LTS. We will use $\approx$ to denote both this binary relation over LTS and the largest weak bisimulation relation over the state sets of a pair of LTS.

Definition 8: **Observation Equivalence** $\approx$: Let $T_1 := (A, \Sigma, \xrightarrow{\mu}, a_0)$ and $T_2 := (B, \Sigma, \xrightarrow{\mu}, b_0)$ be LTS. Then $T_1 \approx T_2$ iff there exists a weak bisimulation S over $A \times B$ such that $(a_0, b_0) \in S$

Thus $T_1 \approx T_2$ iff $a_0 \approx b_0$.

The relation $\approx$ is an equivalence relation over the set of LTS; the reader is referred to [7] for the details in the setting of Milner's process algebra.

IV. Equivalence Preserving Transformations

The purpose of this section is to explain transformations and their use. After demonstrating an intuitive notion of transformation with a simple example, we define a set of behavior preserving transformations and conclude by proving that these preserve the formal observation equivalence of TTMs.

A. Introduction to Transformations

A transformation is *behavior preserving* if it changes a TTM in such a way that the timed behavior of the transformed TTM restricted to the variables of interest, is equivalent (for a specified LTS equivalence relation) to the restricted timed behavior of the original TTM. Consider the two TTMs M_1 and M_2 of Figure 7. Suppose we are only

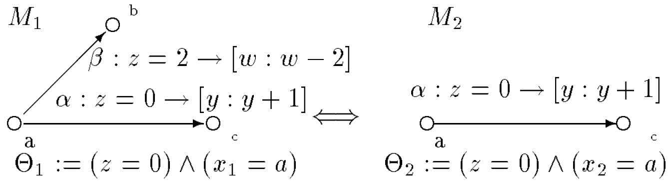


Fig. 7. An example of Transition Addition/Transition Deletion

interested in the timed behavior of the variables y and z. The initial condition Θ_1 prevents β from ever being enabled. If α has the same time bounds in both systems then it is apparent that M_1 and M_2 allow the same timed trajectories over y and z. In fact, since β is never enabled we could delete this transition from M_1 to *transform* M_1 into M_2. Similarly we could add a β transition to M_2 without changing its set of legal trajectories as the initial condition Θ_2 would prevent the new transition from ever occurring. Thus M_2 can also be transformed into M_1.

This is the idea behind the transformational technique of equivalence verification. Given a set of variables of interest $\mathcal{U}$, if it is possible to change one TTM into another by a set of behavior preserving transformations, then the two TTMs' timed behavior restricted to $\mathcal{U}$ will be equivalent (ie. $T_{M_1}|\mathcal{U} \approx T_{M_2}|\mathcal{U}$) and hence the TTMs will behave equivalently in a well defined sense. Clearly if our transformational method is correct, the transformations must abstract away unimportant details in such a way that the key features of the structure of $T_M|\mathcal{U}$ are preserved.

B. A Partial Set of Transformations

The addition of transition β to M_2 to form M_1 is an example of the Transition Addition transformation (**TA**). Going from M_1 to M_2 is an application of the dual of **TA**, the Transition Deletion transformation (**TD**). Below we describe these and the other transformation pairs needed to solve the verification problem of Section V. Throughout the section the transformations refer to the "set of variables of interest" $\mathcal{U}$. These are the variables we wish to "observe" so the transformations are designed to produce TTMs that generate equivalent timed behaviors when restricted to the variables in $\mathcal{U}$.

TA/TD Transition Addition/Transition Deletion: As demonstrated above one may add an instance of a transition to a TTM without changing its timed behavior if the transition's enablement condition is never satisfied in the new source activity. More formally, consider a TTM M with the transition $\alpha := (e, h, l, u)$, where α's full enablement condition from the transition graph of M is $e_\alpha := e \wedge (x = a_1 \vee x = a_2 \vee \ldots \vee x = a_n)$ implying that there are instances of α exiting activities $a_1, \ldots, a_n$ in the transition graph. One may add an instance of α exiting activity $a \notin \{a_1, \ldots, a_n\}$, with any other activity as its destination, provided that in any reachable state assignment q of M it is the case that $q(x) = a$ implies $q(e) = false$. The new full enablement condition for α after the transformation is $e_\alpha := e \wedge (x = a_1 \vee x = a_2 \vee \ldots \vee x = a_n \vee x = a)$
Similarly one can change the full enablement condition of the transition α from $e_\alpha := e \wedge (x = a_1 \vee x = a_2 \vee \ldots \vee x = a_n \vee x = a)$ to $e_\alpha := e \wedge (x = a_1 \vee x = a_2 \vee \ldots \vee x = a_n)$, thereby removing the instance of α exiting activity a in the transition graph of M if in all the reachable state assignments q of M it is the case that $q(x) = a$ implies $q(e) = false$. That is, one may remove an instance of a transition from a TTM if the transition's enablement condition is always false in the source activity from which the instance of the

transition will be deleted.

CA/CD Control Addition/Control Deletion: This transformation lets one add or remove a condition from a transition's enablement condition under certain conditions. Consider a transition α with $e_\alpha := e$ and let p be some first order predicate over the variables in $\mathcal{V}$. If whenever a source activity for α is entered, p is true (p is false), then $e_\alpha^{new} := e \wedge p$ ($e_\alpha^{new} := e \vee p$).
Conversely if $e_\alpha := e \wedge p$ ($e_\alpha := e \vee p$) and, in every activity that α exits, p is guaranteed to be true (false), then $e_\alpha^{new} := e$.

AM/AS Activity Merge/Activity Split: This transformation is defined only when the activity variable x is not in the set of variables of interest (ie. $x \notin \mathcal{U}$). The basic idea of this transformation is that two activities can be merged if they have the same future. Hence, two activities may be merged if they have the same exiting transitions going to the same destination activities. In the example of Figure 8, the activity merge transformation changes δ's full enablement condition from $e_\delta := e \wedge (x = a_1 \vee x = a_2 \vee \ldots)$ to $e_\delta := e \wedge (x = a \vee \ldots)$. For the merged activity one must be careful to choose a name that differs from the remaining TTM activities.

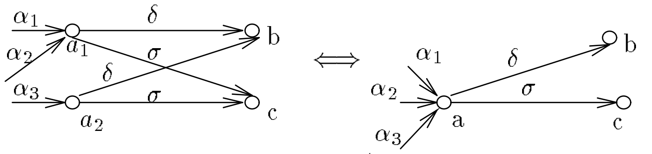


Fig. 8. Activity Merge/Activity Split

For activity splitting, if activity a is the destination activity of transitions $\alpha_1, \ldots, \alpha_k, \alpha_{k+1}, \ldots, \alpha_n$ then split a into a_1 and a_2. $\alpha_1, \ldots, \alpha_k$ will have destination activity a_1 and $\alpha_{k+1}, \ldots, \alpha_n$ will have destination activity a_2. a_1 and a_2 will be the source activities for the same transitions to the same destination activities as in the case of activity a.

RT Rename Transition: This transformation is its own dual. It renames one or more instances of a transition with a new name provided the latter does not conflict with another name or change the structure of $T_M|\mathcal{U}$. Consider Figure 9, where one instance of

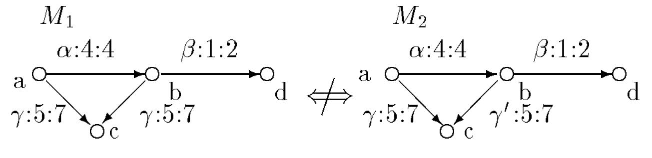
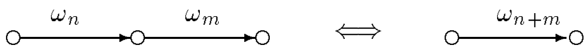


Fig. 9. A Problem with the Rename Transition Transformation

the transition γ is renamed γ', altering the behavior of the system. If the numbers immediately following the transition labels denote the lower and upper time bounds respectively, it is apparent that transition γ is enabled across activities a and b in M_1 so although α happens before γ, γ has been enabled long enough that it can occur before β. On the other hand in M_2 γ is always preempted by α and γ' is always preempted by β.
In general, when it is possible for a transition to remain enabled when moving from one activity to another, then it is not possible to rename the two instances independently (ie. in any application of **RT** the two instances must be given the same name).

OM Operation Modification: If a variable does not occur in the enablement condition of any transition or the operations affecting any other variables and is not in $\mathcal{U}$, the set of variables of interest, then any operations affecting the variable can be added or deleted from any transition.
Let $v \in \mathcal{V} - \mathcal{U}$ and $P_{\mathcal{V}-\{v\}} : \mathcal{Q} \to \mathcal{Q}_{\mathcal{V}-\{v\}}$ be the natural projection from the state assignments to state assignments over $\mathcal{Q}_{\mathcal{V}-\{v\}}$. Then the OM transformation is defined if for all $\alpha \in \mathcal{T}$, the enablement condition e_α of α is independent of v, and $\ker(P_{\mathcal{V}-\{v\}}) \subseteq \ker(P_{\mathcal{V}-\{v\}} \circ h_\alpha)$ (ie. there exists an induced operation function $\overline{h_\alpha} : \mathcal{Q}_{\mathcal{V}-\{v\}} \to \mathcal{Q}_{\mathcal{V}-\{v\}}$ such that $\overline{h_\alpha} \circ P_{\mathcal{V}-\{v\}} = P_{\mathcal{V}-\{v\}} \circ h_\alpha$). If v satisfies these conditions then for any $\alpha \in \mathcal{T}$, h_α can be replaced with the $\overline{h_\alpha}$ induced by $P_{\mathcal{V}-\{v\}}$.
The rationale behind this transformation is that the value of v has no effect upon how the TTM operates on the variables in $\mathcal{U}$, hence we can set v to any value we wish, or ignore it altogether.

WM/WS (Wait Merge/Wait Split): A commonly occurring transition is the "wait" transition that serves the purpose of marking the passing of a fixed number of clock ticks. This transformation is a statement of the intuitive notion that waiting for n ticks and then waiting for m ticks is equivalent to waiting for n+m ticks. For technical reasons we require that $n \geq 1$.



where $\omega_i := (True, [\,], i, i)$

Fig. 10. Wait Merge/Wait Split

C. Proof of Transformation Consistency

In the following proof that the transformations preserve observation equivalence, most of the work is accomplished by the choice of an appropriate equivalence relation on TTMs.

Theorem 1: Let $M := (\mathcal{V}, \Theta, \mathcal{T})$ be a TTM, $\mathcal{U} \subset \mathcal{V}$ and $\mathbf{T}$ be one of the TTM transformations of subsection IV-B. If $\mathbf{T}(M)$ is defined, then $M|\mathcal{U} \approx \mathbf{T}(M)|\mathcal{U}$.

Proof: Consider the TA (transition addition) transformation. An instance of a transition is added only if the transition's enablement condition will never be satisfied in the source activity the new instance exits, thereby forming $\mathbf{T}(M)$. The newly added instance of the transition never occurs in $\mathbf{T}(M)$ since it is never enabled. Therefore $T_M = T_{\mathbf{T}(M)}$ and hence $T_M|\mathcal{U} \approx T_{\mathbf{T}(M)}|\mathcal{U}$.

All the other transformations, with the exceptions of the OM and WM/WS transformations, are similarly only defined when their application leaves T_M unaltered.

Suppose $v \in \mathcal{V}$ and M's effect upon v is being modified. The OM transformation does not change the tree structure of T_M since $\mathbf{T}(M)$ is defined only if v does not occur in the enabling conditions of any TTM transitions and also v does not affect any other variables. By changing a transition's operation on v we are in effect merely relabeling the transition of T_M. Suppose β is the transition in M being modified, that is h_β becomes h_β^{new} by changing the value h_β assigns to v. Then $T_{\mathbf{T}(M)}$ is obtained from T_M by replacing all occurrences of h_β with h_β^{new}. But $v \notin \mathcal{U}$, the set of variables of interest (otherwise OM is not defined). Thus the maps induced in $\mathcal{U} \cup \{t\}$ by h_β and h_β^{new} are identical (ie. $\overline{h_\beta} = \overline{h_\beta^{new}}$) and so $T_M|\mathcal{U} \approx T_{\mathbf{T}(M)}|\mathcal{U}$ (in fact they are equal).

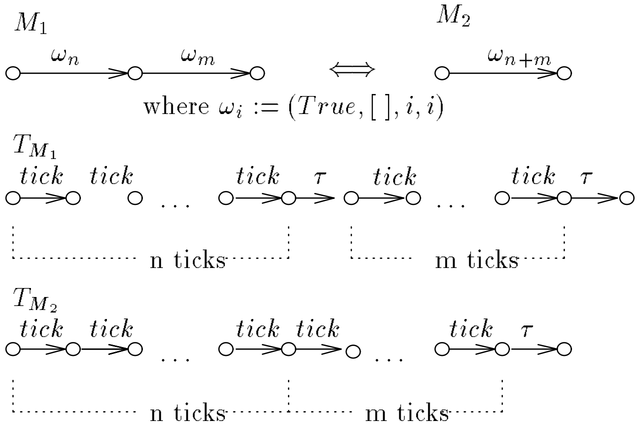


Fig. 11. Illustrating the Observation Congruence of WM/WS TTMs

Finally we turn our attention to the WM/WS transformations. These transformations actually alter the structure of T_M. Consider the transitions labeled by M_1 and M_2 of Figure 11. The transition labelled by M_2 is obtained from the transitions of M_1 by an application of WM (M_1 from M_2 via WS). It is easy to show that $T_{M_1} \approx T_{M_2}$. But if the transitions of M_1 occur in a TTM M that we transform to obtain $\mathbf{T}(M)$, by replacing the transitions M_1 with the transition M_2, we must ensure that M and $\mathbf{T}(M)$ are observationally equivalent. In both cases $\mathbf{T}(M)$ is only defined for $n \geq 1$ so in both T_{M_1} and T_{M_2}, the first event must be a *tick*. Thus both M and $\mathbf{T}(M)$ may only enter a region in which their structures differ by taking the initial *tick* transition, after which they both produce the same future observations. Under these circumstances it is easily verified that $M \approx \mathbf{T}(M)$.

We conclude that for any transformation $\mathbf{T}$ and TTM M we have $M \approx \mathbf{T}(M)$. ∎

More transformations can be added to those listed in subsection IV-B. One has to verify that each new transformation preserves observation equivalence. Theorem 1 implies that any TTM derived from another TTM via a finite sequence of the transformations of subsection IV-B is observationally equivalent to the original TTM.

V. The Delayed Trip System

This section introduces the Delayed Trip System (DTS), a real-time example from industry. The problem is then solved in the TTM framework.

Currently in industry many of the control systems that were previously implemented using discrete and analog components are being replaced by microprocessor-based implementations in order to realize cost savings and greater flexibility. A question that now arises is whether the new system behaves the same as the old. That is, are the two implementations equivalent?

A. Setting and Assumptions

The DTS is typical of many real-time systems from industry. When a certain set of circumstances arises, we want the system to provide the correct response in a timely fashion. In this case when pressure and power measurements exceed acceptable safety limits in a particular way, we want the DTS controller to trip a relay causing the system to shut down. The result of failure to shut down could be

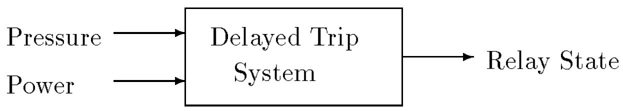


Fig. 12. Block Diagram for the Delayed Trip System

catastrophic. Conversely, each time the system is improperly shut down, significant financial loss could result (eg. stopping a sensitive chemical process in mid-reaction could ruin the product and possibly damage the plant). Clearly it is important that the DTS behave in a very specific manner.

The desired input/output relationship for the DTS block diagram has the following informal description: if the power exceeds power threshold PT and the pressure exceeds delayed set point DSP, then wait for 3 seconds. If after 3 seconds the power is again greater than PT, then open the relay for 2 seconds.

The new DTS is to be implemented on a microprocessor system with a cycle time of 100ms. That is, the system samples the inputs and passes through a block of control code every 0.1 seconds. We assume that the input signals have been properly filtered and that the sampling rate is high enough to ensure proper control. Figure 13 displays the pseudocode for a proposed control program for the microprocessor. The code uses integer counter variables c_1 and c_2 to time the 3 second and 2 second delays respectively. Also, the program makes use of the variables *Pressure*, *Power* and *Relay* for the sampled DTS inputs and output respectively.

The question of whether a microprocessor implementing the algorithm of Figure 13 satisfies the informal requirements above is somewhat problematic. To answer it we first pose the DTS problem in the TTM framework.

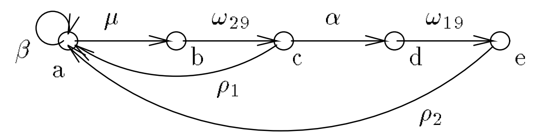
```
If Power ≥ PT then
   If counter c1 is reset then
     If counter c2 is reset then
       If Pressure ≥ DSP then
         increment c1          ]μ1
       Endif
     ElseIf ¬(counter c2 timed out) then
         increment c2          ]μ2
         open Relay            ]
       Endif
     Endif
   ElseIf counter c1 timed out then
       open Relay              ]
       reset c1                |α
       increment c2            ]
     Else
       increment c1            ]μ1
     Endif
   Endif
Else If counter c1 is reset then
     If counter c2 is reset then
       close Relay             ]β
     ElseIf counter c2 timed out then
         close Relay           ]ρ2
         reset c2              ]
       Else
         increment c2          ]μ2
         open Relay            ]
       Endif
     Endif
   ElseIf counter c1 timed out then
       reset c1                ]ρ1
     Else
       increment c1            ]μ1
     Endif
  Endif
Endif
```

Fig. 13. Pseudocode for Proposed DTS Control Program

B. Modeling the Delayed Trip System Specification

By modeling the DTS specification as a TTM we can remove any ambiguities from the informal specification and ensure that the input/output behavior of the microprocessor system is completely determined. When the DTS is implemented in the actual system there are three identical DTSs running in parallel, with the final decision on when to shut down the system implemented by majority rule. As a result it is important that an individual system be able to recover when it is in disagreement with the other two systems. Also a system should never deadlock. For instance, after the power and pressure have exceeded their critical values and the system has waited 3 seconds to check the power level again, if the power is below its threshold value PT, then the system should reset and revert to monitoring both inputs. This is *implicit* in the informal specification.

In order to facilitate the verification process, the TTM representation of the desired I/O characteristics for the DTS is put in a form that closely resembles the microprocessor behavior. A *tick* of the global TTM clock



SPEC Transition Table

$$
\begin{aligned}
\Theta &:= x = a \wedge Relay = \text{CLOSED} \\
&\quad\ \wedge Power < \text{PT} \wedge Pressure < \text{DSP} \\
\mu &:= (e_\mu, [\,], 1, 1) \\
\alpha &:= (Power \geq \text{PT}, [Relay : \text{OPEN}], 1, 1) \\
\beta &:= (Power < \text{PT}, [Relay : \text{CLOSED}], 1, 1) \\
\omega_{29} &:= (true, [\,], 29, 29) \\
\omega_{19} &:= (true, [\,], 19, 19) \\
\rho_1 &:= (Power < \text{PT}, [\,], 1, 1) \\
\rho_2 &:= (Power < \text{PT}, [Relay : \text{CLOSED}], 1, 1) \\
\text{where} & \\
e_\mu &:= Power \geq \text{PT} \wedge Pressure \geq \text{DSP}
\end{aligned}
$$

Fig. 14. SPEC: TTM Representation of DTS Specification

is assumed to represent 100ms, the cycle time of the microprocessor. In the TTM specification SPEC of Figure 14, the enablement conditions of a transition must be satisfied for at least one clock tick before the transition can occur. The earlier assumption that the input signals are filtered to ensure proper control guarantees that any change will persist at least one sampling period and hence will be detected.

After transition μ occurs, SPEC waits in activity b for 29 clock ticks (2.9 seconds) before proceeding to activity c. At activity c the power level is checked again. If the power is too high then the system opens the relay via transition α, or else the system resets via ρ_1 to continue monitoring both inputs in activity a. After α the system waits in activity d for 19 clock ticks (1.9 seconds) and then moves to e. As an added safety feature, the system remains at e as long as $Power \geq$ PT. Otherwise the system resets to a via ρ_2 while closing the relay. Once back in activity a, β ensures that the relay is closed once the power returns to an acceptable level.

From the above discussion it is apparent that the TTM SPEC gives a more thorough description of what is required of the DTS, expanding upon the previous informal specification. It now remains to model the microprocessor system in the TTM framework before formalizing the verification problem.

C. Modeling the Microprocessor DTS Implementation

On the right hand side of Figure 13 is a list of transition names. Each time the microprocessor passes through the code it performs one of the groups of operations identified by a transition name. Identical groups of operations on the program variables are identified by the same transition name. A group of program operations then determines the operation function of the transition. The enablement conditions for these transitions are formed by taking the conjunction of the conditions specified by the 'If' statements for each occurrence of a given transition's program operations. As an example consider e_{μ_2}, the enablement condition for μ_2. The first occurrence of μ_2 happens if $Power \geq \mathrm{PT}$, c_1 is reset, not (c_2 is reset) and not (c_2 has timed out). The second occurrence is executed if not ($Power \geq \mathrm{PT}$), c_1 is reset, not (c_2 is reset) and not (c_2 has timed out). Counting off 20 consecutive cycles through the code translates to an elapsed time of 2 seconds, the minimum time the relay is to remain open. If we consider the counter variables to be reset when they are equal to zero and counter c_2 as timed out when $c_2 \geq 20$, μ_2's enablement condition becomes:

$$
\begin{aligned}
e_{\mu_2} &:= (Power \geq \mathrm{PT} \wedge c_1 = 0 \wedge c_2 \neq 0 \wedge c_2 < 20) \\
&\quad \vee (Power < \mathrm{PT} \wedge c_1 = 0 \wedge c_2 \neq 0 \wedge c_2 < 20) \\
&\text{or} \\
e_{\mu_2} &:= c_1 = 0 \wedge 0 < c_2 < 20
\end{aligned}
$$

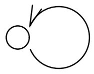
In the final step we use the fact that c_2 can never be negative since it starts at $c_2 = 0$ and all transitions reset c_2 to zero or increment it.

Similarly we can obtain the enabling conditions for the other transitions. As mentioned earlier, with each pass through the code, the microprocessor picks out one of the labeled blocks of code. The block chosen is the one whose enabling conditions are satisfied. The program then loops back to the start and re-evaluates all the enabling conditions in the next cycle. Hence each transition has a lower and upper time bound of one.

All of the above information is used to construct the simple TTM PROG (see Figure 15). The single activity is representative of the fact that the program is basically a large case statement implemented using If statements, the appropriate case being selected out of all possible cases on each pass through the code.

D. The Verification Problem in Terms of TTMs

Having modeled the specification and pseudocode in the two preceding subsections we are now able to consider what the verification problem means in terms of the TTM models SPEC and PROG. The original question was: 'Does the program do what we want?' In the TTM setting this becomes the question of whether the timed trajectories of the inputs and outputs of PROG are "equivalent" to the timed input/output trajectories of SPEC. That is, ignoring the events that do not affect the state of *Relay*, do SPEC and PROG permit the same interleavings of changes to



selfloop$(\mu_1, \mu_2, \alpha, \beta, \rho_1, \rho_2)$

PROG Transition Table

$$
\begin{aligned}
\Theta &:= c_1 = c_2 = 0 \wedge Relay = \mathrm{CLOSED} \\
&\quad\ \wedge Power < \mathrm{PT} \wedge Pressure < \mathrm{DSP} \\
\mu_1 &:= (e_{\mu_1}, [c_1 : c_1 + 1], 1, 1) \\
\mu_2 &:= (c_1 = 0 \wedge 1 \leq c_2 \leq 19, \\
&\quad\ [c_2 : c_2 + 1, Relay : \mathrm{OPEN}], 1, 1) \\
\alpha &:= (Power \geq \mathrm{PT} \wedge c_1 \geq 30, \\
&\quad\ [c_1 : 0, c_2 : c_2 + 1, Relay : \mathrm{OPEN}], 1, 1) \\
\beta &:= (Power < \mathrm{PT} \wedge c_1 = c_2 = 0, \\
&\quad\ [Relay : \mathrm{CLOSED}], 1, 1) \\
\rho_1 &:= (Power < \mathrm{PT} \wedge c_1 \geq 30, [c_1 : 0], 1, 1) \\
\rho_2 &:= (Power < \mathrm{PT} \wedge c_1 = 0 \wedge c_2 \geq 20, \\
&\quad\ [c_2 : 0, Relay : \mathrm{CLOSED}], 1, 1)
\end{aligned}
$$

where

$$
\begin{aligned}
e_{\mu_1} &:= (Power \geq \mathrm{PT} \wedge Pressure \geq \mathrm{DSP} \\
&\quad\ \wedge c_1 = c_2 = 0) \vee (1 \leq c_1 \leq 29)
\end{aligned}
$$

Fig. 15. PROG: TTM Representation of Pseudocode for DTS

the input variables *Power* and *Pressure*, output variable *Relay* and clock variable t? In this case the *set of variables of interest* is given by:

$$\mathcal{U} := \{Power, Pressure, Relay\}$$

In the next subsection we answer the question using the transformations of Section IV.

E. Solving the DTS Verification Problem

We now solve the DTS verification problem by applying the transformations described in the previous section to check if PROG $\approx/\mathcal{U}$ SPEC, where $\mathcal{U} := \{Power, Pressure, Relay\}$. While the presented example has a finite state representation, we would like to stress that the technique employed is applicable to TTMs, which in general, may not have a finite state space.

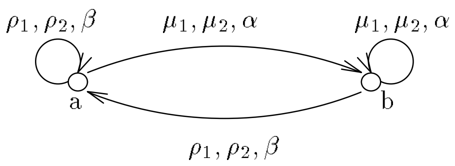
Starting from PROG with SPEC as the final goal, we try to make PROG look progressively more like SPEC until we are left with a copy of SPEC at the end. At each step we check that the desired transformation is applicable and describe its effect. Starting with $PROG_0$:=PROG, at each step i we apply a transformation to $PROG_{i-1}$ to obtain $PROG_i$.

Claim 1: PROG is behaviorally equivalent to SPEC over $\mathcal{U}$.

Proof: (PROG $\longrightarrow$ SPEC Over $\mathcal{U}$)

0. The original PROG TTM is shown in Figure 15.

1. AS To apply AS we only have to make sure that instances of every transition exit both of the new activities. Since all the transitions are self-looped to



$\Theta_{new} := \Theta_{PROG} \wedge x = a$

Fig. 16. TTM for $PROG_1$

the one activity in the original system, we have some choice over how we distribute their destination activities. The reason for the choice shown in Figure 16 will become apparent in the next few steps.

2. TD TD is applied next to remove the instance of transition β exiting activity b and the instances of transitions α and ρ_1 exiting activity a. We are justified in these actions since:

i. All transitions entering b increment either c_1 or c_2.

ii. All transitions either increment or reset c_i to 0 so in any activity $c_i \geq 0$.

iii. All transitions entering a either set $c_1 = 0$ or leave c_1 unaffected while requiring $c_1 = 0$ in their enablement conditions. Therefore in activity a $c_1 = 0$.

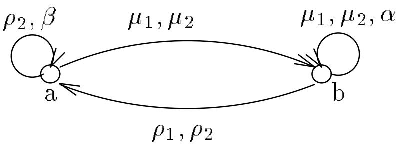
Hence by (i), (ii) and (iii) we know that:

$$(x = b) \Rightarrow (c_1 > 0 \vee c_2 > 0) \Rightarrow (e_\beta = false)$$

and

$$(x = a) \Rightarrow (e_\alpha = e_{\rho_1} = false)$$

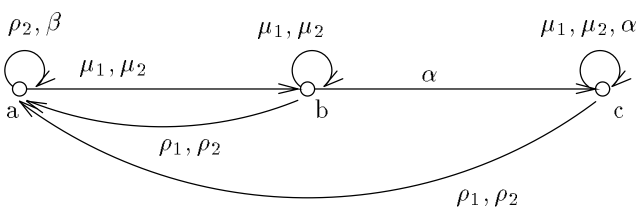
Thus we are justified in deleting the β exiting activity b and the α and ρ_1 exiting activity a.



$\Theta_{new} := \Theta_{PROG} \wedge x = a$

Fig. 17. TTM for $PROG_2$

3. AS This time we split activity b with α exiting b to the newly formed c activity (see Figure 18). Notice that in splitting activity b into b and c we have not altered the timed behavior. Both b and c have the same possible futures as activity b in $PROG_2$.



$\Theta_{new} := \Theta_{PROG} \wedge x = a$

Fig. 18. TTM for $PROG_3$

4. TD Upon entering activity c we know that $c_1 = 0 \wedge c_2 > 0$ since $h_\alpha := [c_1 : 0, c_2 : c_2 + 1, Relay : \text{OPEN}]$ and by 2(ii) we know that $c_i \geq 0$ in activity b. But e_{μ_1} requires that either $c_1 = c_2 = 0$ or $1 \leq c_1 \leq 29$, so e_{μ_1} is initially false in activity c. Also the other transitions entering c (α and μ_2) leave c_1 unaltered and only increment c_2. Hence

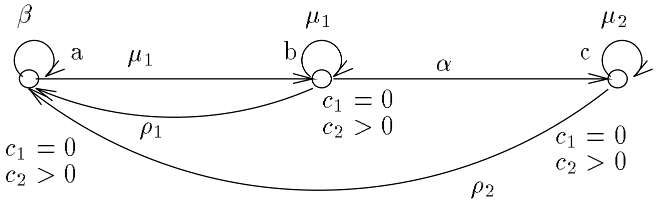
$$\begin{aligned} x = c &\implies c_1 = 0 \wedge c_2 > 0 \\ &\implies e_{\mu_1} = e_{\rho_1} = e_\alpha = false \end{aligned}$$

Conclusion: delete the instances of μ_1, ρ_1, and α with source activity c.

5. TD The initial condition Θ_{new} starts $PROG_4$ in activity a with $c_2 = 0$. The only transitions that affect c_2 are μ_2 and α. Transition μ_2 requires $c_2 > 0$ to occur. Hence, starting from the initial state, α must precede μ_2. Once α has occurred we have $x = c$ with only ρ_2 exiting c. This transition sets $c_2 = 0$ so (i)$c_2 > 0$ iff $x = c$. Thus

$$\begin{aligned} (i) &\implies (e_{\mu_2} = true \implies x = c) \\ &\text{also} \\ &\qquad e_{\rho_2} = true \implies x = c \end{aligned}$$

Conclusion: delete all instances of μ_2 and ρ_2 except



$\Theta_{new} := \Theta_{PROG} \wedge x = a$

Fig. 19. TTM for $PROG_5$

those with source activity c. This leaves us with $PROG_5$ as shown in Figure 19. The range of values that c_1 and c_2 take on in each activity can be easily deduced from 5(i) and 2(iii) so we include this information in Figure 19 as well.

6. RT Referring to Figure 19, we can rename the instance of μ_1 exiting activity a without affecting the dynamics of the variables of interest because μ_1 is now the only transition entering activity b and after a transition occurs, if it remains enabled, its time bounds are reset. This means that a problem like that illustrated in **RT** Figure 9 cannot occur as a result of renaming only one of the instances of μ_1. The new transition exiting a will be called μ.

7. CD Considering the enablement conditions in $PROG_6$ we have:

$$\begin{aligned} e_\mu &:= e_{\mu_1} \\ &= \underbrace{(Power \geq \text{PT} \wedge Pres. \geq \text{DSP} \wedge c_1 = c_2 = 0)}_{p} \\ &\quad \vee \underbrace{(1 \leq c_1 \leq 29)}_{q} \end{aligned}$$

When $x = a$ by 2(iii) and 5(i) we know that $c_1 = c_2 = 0$ and when $x = b$ by 2(ii) we know that $c_1 > 0$. This gives:

$$(i) x = a \implies q = false$$
$$(ii) x = b \implies p = false$$

Using the Control Deletion transformation with (i) as justification, we can change e_μ to $e_\mu^{new} := p$. Similarly using (ii) and Control Deletion again we obtain $e_{\mu_1}^{new} := q$.
Now consider e_μ^{new} and e_β:

$$\begin{aligned} e_\mu^{new} &:= Power \geq \mathrm{PT} \wedge Pressure \geq \mathrm{DSP} \\ &\quad \wedge c_1 = c_2 = 0 \\ e_\beta &:= Power < \mathrm{PT} \wedge c_1 = c_2 = 0 \end{aligned}$$

but

$$x = a \implies c_1 = c_2 = 0$$

Applying **CD** yet again to simplify further we now have:

$$\begin{aligned} e_\mu^{new} &:= Power \geq \mathrm{PT} \wedge Pressure \geq \mathrm{DSP} \\ e_\beta^{new} &:= Power < \mathrm{PT} \end{aligned}$$

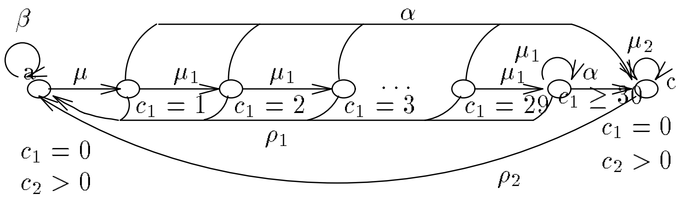
In a similar fashion, again applying **CD**:

$$x = c \implies c_1 = 0$$

so

$$e_{\rho_2}^{new} := Power < \mathrm{PT} \wedge c_2 \geq 20$$

8. *AS* The activity b is now split into thirty different activities with μ_1 taking the TTM from one new b activity to the next as c_1 is incremented. After μ occurs we are in an activity where $c_1 = 1$. μ_1 takes us to the next activity where $c_1 = 2$ and so on until we reach an activity where $c_1 \geq 30$ and μ_1 is self-looped. For each value of c_1 between 1 and 29, b has been split into a new activity, with an additional activity for $c_1 \geq 30$. We project out the TTM's dependence on c_1 by systematically adding new activities to the TTM to a point where for each value of c_1 between 1 and 30 there is an individual activity. Again note that we are in no way changing the dynamics of the system over $\mathcal{U}$ as the same transitions exit each of the new activities.



$$\Theta_{new} := \Theta_{PROG} \wedge x = a$$

Fig. 20. TTM for $PROG_8$

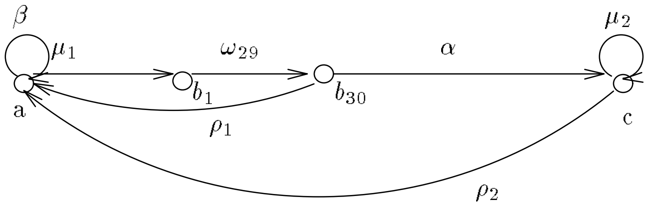
9. *TD* Knowing the value of c_1 in each of the newly added activities allows us to delete all instances of α and ρ_1 except for the activity where $c_1 \geq 30$ since both transitions' enablement conditions require $c_1 \geq 30$. Also, the μ_1 transition self-looped at the activity $c_1 \geq 30$ may be removed because $e_{\mu_1} := (1 \leq c_1 \leq 29)$ in $PROG_8$.
10. *CD* Now that transition μ_1 has as source activities only those activities for which $1 \leq c_1 \leq 29$, e_{μ_1} is always true in any of μ_1's source activities. Thus we can remove the c_1 dependence from e_{μ_1}. The same can be done for α and ρ_1 giving:

$$\begin{aligned} e_{\mu_1}^{new} &:= true \\ e_\alpha^{new} &:= Power \geq \mathrm{PT} \\ e_{\rho_1}^{new} &:= Power < \mathrm{PT} \end{aligned}$$

11. *OM* Variable c_1 no longer occurs in any transition's enabling conditions or operation functions that affect other variables. Hence we can drop the variable from all transition operations. The modified transitions are

$$\begin{aligned} \mu^{new} &:= (Power \geq \mathrm{PT} \wedge Pressure \geq \mathrm{DSP}, [\,], 1, 1) \\ \mu_1^{new} &:= (true, [\,], 1, 1) = \omega_1 \end{aligned}$$

12. *WM* All instances of μ_1 are now merged into one ω_{29} by the Wait Merge transformation (See Figure 21).



$$\Theta_{new} := Relay = \mathrm{CLOSED} \wedge c_2 = 0 \wedge x = a$$

Fig. 21. TTM for $PROG_{12}$

13-17. Now repeat steps 8-12 for activity c and transition μ_2 to project out the dynamics of variable c_2 and we have the desired result, a TTM identical to SPEC. ■

By transforming PROG into SPEC above we have shown that the pseudocode implements an algorithm that satisfies the behavioral requirements expressed by SPEC.

VI. Incompleteness of Transformations

In [13] the set of transformations of Section IV is shown to be incomplete for proving observation equivalence of TTMs and it is further demonstrated that no finite set of transformations is complete for proving observation equivalence of general TTMs. The proof closely follows a similar proof in Milner's process algebra [7]. As in Milner's setting, the incompleteness property does not prevent the theory from being potentially useful in many practical applications. Indeed the exponential state explosion that occurs with the addition of new variables makes exhaustive verification routines impractical for even finite state TTMs. Thus heuristic methods such as the transformational technique introduced in this paper provide a useful method of real-time system verification. Also, the transformations may be used to synthesize an implementation from a specification that is correct by construction. That is, the implementation resulting from the transformations will be guaranteed to be observationally equivalent to the specification, thereby eliminating the need to perform an exhaustive equivalence verification.

In this paper we have demonstrated that the transformational proof technique employed above can be used to guarantee the formal observational equivalence (in the sense of [7]) of TTMs' legal trajectories. The same technique can be applied to other problems to formally prove the observation equivalence of two TTMs.

In conclusion, equivalence preserving transformations of TTMs were introduced as a method of verifying the equivalence of two TTMs. The Delayed Trip System (DTS) example has been introduced as a practical application of the equivalence of TTMs over a set of variables of interest. Finally, a set of transformations was developed and shown to be sufficiently expressive to solve the DTS verification problem.

References

[1] L. Lee, *The Day the Phones Stopped*, Donald I. Fine Inc., New York, 1991.

[2] D. Harel, "Biting the silver bullet: Toward a brighter future for software development", *IEEE Computer Magazine*, vol. 25, no. 1, pp. 8–20, 1992.

[3] J. S. Ostroff, *Temporal Logic for Real-Time Systems*, Advanced Software Development. Research Studies Press, Somerset England, 1989.

[4] J. S. Ostroff and W. M. Wonham, "A framework for real-time discrete event control", *IEEE Transactions on Automatic Control*, vol. 35, no. 4, pp. 386–397, 1990.

[5] C.A.R. Hoare, *Communicating Sequential Processes*, International Series in Computer Science. Prentice-Hall International, Englewood Cliffs, NJ, 1985.

[6] R. Milner, *A Calculus of Communicating Systems*, vol. 92 of *Lecture notes on computer science*, Springer-Verlag, New York, 1980.

[7] R. Milner, *Communication and Concurrency*, Prentice Hall, New York, 1989.

[8] F. Moller and C. Tofts, *A Temporal Calculus of Communicating Systems*, vol. 458 of *LNCS*, pp. 401–415, Springer–Verlag, 1990.

[9] Y. Wang, *CCS + Time = an Interleaving Model for Real Time Systems*, vol. 510 of *LNCS*, pp. 217–228, Springer–Verlag, 1991.

[10] A. Klusener, *Abstraction in Real Time Process Algebra*, vol. 600 of *LNCS*, pp. 325–352, Springer–Verlag, 1991.

[11] M. K. Franklin and A. Gabrielian, "A transformational method for verifying safety properties in real-time systems", in *Proceedings 10th IEEE Real-Time Syst. Symp*, December 1989, pp. 112–123.

[12] R. DeNicola, "Extensional equivalences for transition systems", *Acta Informatica*, vol. 24, pp. 211–237, 1987.

[13] M. Lawford, "Transformational equivalence of timed transition models", M.A.Sc. thesis, Department of Electrical Engineering, University of Toronto, Toronto, ON, 1992, Also appears as Systems Control Group Report #9202, Dept. of Elec. Eng, Univ. of Toronto, 1992.

[14] M. Lawford and W.M. Wonham, "An application of real-time transformational equivalence", in *Proc. of 26th Conf. on Information Sciences and Systems*, Princeton, NJ, Mar. 1992, vol. 1, pp. 233–238.

[15] M. Lawford and W.M. Wonham, "Equivalence preserving transformations for timed transition models", in *Proc. of 31st Conf. Decision and Control*, Tucson, AZ, USA, Dec. 1992, pp. 3350–3356.

Mark Lawford (S'88) received the B.Sc. degree in Engineering Mathematics from Queen's University, Kingston, Ontario, Canada in 1989 and the M.A.Sc. degree from the University of Toronto, Ontario, Canada, in 1992. He is currently enrolled in the Ph.D. program of the Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto.

He recently completed a contract position as a real-time simulator consultant with AlliedSignal Aerospace Canada Ltd. and presently acts as an internet consultant for QL Systems Ltd. His research interests include discrete event systems, formal methods for real-time systems and equivalence verification techniques.

W. M. Wonham (M'64-SM'76-F'77) received the B.Eng. degree in engineering physics from McGill University, Montreal, P.Q., Canada, in 1956, and the Ph.D. degree in control engineering from the University of Cambridge, England, in 1961.

From 1961-1969, he was associated with the Control and Information Systems Laboratory at Purdue University, Lafayette, IN, the Research Institute for Advanced Studies (RIAS) of Martin Marietta Co., the Division of Applied Mathematics at Brown University, Providence, RI, and (as a National Academy of Sciences Research Fellow) with the Office of Control Theory and Application of NASA's Electronics Research Center. In 1970, he joined the Systems Control Goup of the Department of Electrical Engineering at the University of Toronto, Ontario, Canada. He currently holds the J. Roy Cockburn Chair. In addition, he has held visiting academic appointments with the Department of Electrical Engineering at Massachusettes Institute of Technology, Cambridge, MA, the Department of Systems Science and Mathematics at Washington University, St. Louis, MO, the Department of Mathematics of the University of Bremen, the Mathematics Institute of the Academia Sinica, Beijing, the Indian Institute of Technology, Kanpur, and other institutions. His research interests have lain in the areas of stochastic control and filtering, the geometric method of linear multivariable control, and more recently in the discrete event systems from the point of formal logic and language. He has coauthored about sixty research papers as well as the book *Linear Multivariable Control: A Geometric Approach.*

Dr. Wonham is a Fellow of the Royal Society of Canada. In 1987, he was the recipient of the IEEE Control Systems Science and Engineering Award, and in 1990, was Brouwer Medalist of the Netherlands Mathematical Society