The Transition Assertions Specification Method

Victor A. Carreño
vac@air12.larc.nasa.gov

NASA Langley RC
MS 130
Hampton, VA 23681-0001
U.S.A.

Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
U.K.

Abstract

A modeling and specification method for real-time, reactive systems is described. Modeling is performed by constructing time dependent relations of the system parameters. A textual formal notation using higher order logic and a graphical notation are presented. The formal notation allows the use of rigorous mathematical methods on the specification, one of the primary sources of design errors. A cruise control case example is included in the paper and the HOL mechanized theorem prover is used to show that the specification complies with some top level requirements.

1 Introduction

This paper consists of two parts. One, a generic model to represent time dependent systems. Two, a method to specify real-time systems using a graphical and textual notation. The generic model is described in section 2 and forms the basis for the graphical and textual specification method. Reading section 2 is not necessary for the understanding of the specification method. Section 2 includes the rationale for developing such a model. The generic model presented is called the Transition Assertions model. Its main characteristics are:

1. Transitions are not instantaneous; they require at least one time step.
2. The model uses assertions over variables to describe behaviour rather than the more common method of assertions over states [1, 4, 5, 7, 9, 11]
3. Variables may or may not be specified at a given time, thus the state of the system may not always be completely defined.
4. Time is discrete and is mapped to the natural numbers including zero. The size of the time increments is arbitrarily chosen. For example t= 0,1,2, ... could correspond to $0\mu s, 1\mu s, 2\mu s$, ... or 0ns,1ns,2ns, ...

Section 3 describes the specification method. The objective is to have a method that:

1. Describes time dependent systems or properties of time dependent system in which the timing characteristic are explicitly present.
2. Has a formal mathematical notation to support verification through theorem proving.
3. Is expressive enough to support the specification of relevant systems.
4. Is of manageable complexity.

Section 4 is an example of the specification of a cruise control system using the transition assertion specification method. The specification was shown to comply with three top level requirements. The HOL mechanized theorem prover was used to perform the proof of compliance. A discussion of the proof effort and the findings is in section 5.

2 The Transition Assertions Generic Model

The Transition Assertion model is a generic model to describe a dynamic system over time. Two approaches can be taken to model system behaviour with respect to time. The first one, and the most widely used [5, 6, 9, 10], assumes that the system is in a given state, and that it transitions from one state to another instantaneously. This approach is generally suitable to model a wide range of systems, but it can also bring mathematical or conceptual contradiction when formalizing a specification. This is due to the fact that when a variable is transitioning from one value to a different value, no time elapses, and a variable can have two distinct values at the same time.

The second approach, and the one used in this model, assumes that the system is constantly transitioning, and in order to distinguish between two different states, an interval of time, greater than zero, must have elapsed. A functional relation is defined between time and values in which if $t_1 = t_2$ then $v(t_1) = v(t_2)$. This approach is also used in [4, 8].

Variables (or system parameters) are functions from time to values. Time is a natural number used as the argument to the time dependent functions. Using the natural numbers to represent time significantly simplifies the transition assertions model. This does not result in a penalty on accuracy since the time steps can be made arbitrarily small. The variables, values, assertions, and the absolute time t are defined as follows:

V the set of system variables, including inputs, outputs and control variables. Each variable v is a function from time to value. v:N $\rightarrow$ {values}.

Val the set of sets of possible values. The set Val is infinite and can include the sets N, {colors}, {names}, and others. ({values} is used to range over sets of values.)

T the set of system assertions. Each assertion is a first order or higher order formula τ :B.

t the system time $t \in$ N.

A system execution is defined as a sequence of observations of the system variables over time, starting at time zero or at a reference time t_0:

		t		
	0	1	2	...
w	35^oF	36^oF	36^oF	
x	1	5	237	
y	0			
z	red	$blue$	$blue$	
.				
.				
.				

The value of a variable can be undefined or unobservable at a given time and is represented by a blank space in the example above for $y(1)$ and $y(2)$. The set of all possible system executions describes the system behaviour. The definition of system execution and behaviour is similar to the definition of computation and behaviour, respectively, in [9] for linear temporal logic.

System assertions restrict the set of behaviours and are used to model the system. The assertion $\forall t.Vcc(t) = 5$ represents a system in which the value of Vcc is always five (for t greater or equal to zero). The assertion $x(t) < y(t)$ means that the value of x is less than the value of y at all times. This is assuming that an ordering exists amongst the values of x and y. The assertion $x(t) < x(t + t')$, for $t' \neq 0$, means that at a future time the value of x is going to be larger than the value of x now. That is, x is monotonically increasing. The following are other examples of system assertions:

$x(t+1) = x(t) + 1$
$(t = 0) \Rightarrow x(t) = 0$
$y(t) < 10 \Rightarrow ((x(t+1) = x(t) + z(t)) \wedge (y(t+1) = y(t) + 1))$

A system S is defined by the conjunction of all the system assertions; $S = \forall t. \tau_1 \wedge \tau_2 \wedge \ldots \wedge \tau_n$. The following system will increment indefinitely the variable x from zero, at time equals zero: V $= \{x\}$; T $= \{x(0) = 0,\ x(t+1) = x(t) + 1\}$. It is possible to define a system with contradictory predicates. Consider for example V $= \{x\}$; T $= \{x(0) = 0,\ x(t+1) = x(t) + 1,\ x(20) = 10\}$. This definition will result in S = F and it is impossible to implement.

3 The Graphical and Textual Notation

The specification method uses 8 transition models. The transition models assert that if a condition P holds (is true) at time t, then a condition Q will hold at a time $t + dt$ later. The increment time dt is bounded by a lower and upper bound d and D, where d and D are natural numbers. dt is therefore always greater than zero.

Each transition model has a graphical representation associated with it. A specification can be formulated by generating the graphical model of the transition or by directly writing the textual model. There are 3 basic transition models and 5 compound or derived models.

3.1 The Three Basic Models

The first transition model is the Multiple Transition. This model is defined by the following textual and graphical representation:

1. Multiple Transition

$MT\ d\ D\ P\ Q = \exists dt.\ \forall t.\ (d < dt) \wedge (dt < D) \wedge (P\ t \Rightarrow Q\ t\ (t + dt))$

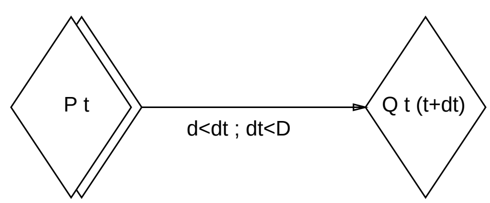

Figure 1. Multiple Transition Model

In transition model number 1, a transition does not have to be finished for a new one to begin (hence multiple transition). Consider for example a conveyor belt:

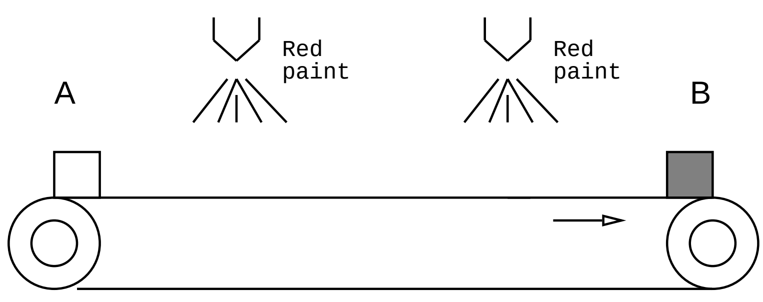

Figure 2. Conveyor Belt

Condition P is *part_at_A* and condition Q is *part_at_B_painted_red*. A new part can be put at A before the first one appears at B. The number of transitions that can be taking place simultaneously depends on how often condition P can be satisfied and the transition time dt.

The specification for the conveyor belt using model 1 could be:

$$MT\ 30\ 35\ (\lambda\ t.part\_at\_A\ t)\ (\lambda\ t\ t'.part\_at\_B\_painted\_red\ t')$$

or expanding with the definition of MT:

$$\exists\ dt.\forall\ t.30 < dt \wedge dt < 35 \wedge (part\_at\_A\ t \Rightarrow part\_at\_B\_painted\_red\ (t + dt))$$

By modelling the conveyor belt with transition model 1, a condition exists in which painted parts may appear at B without having been put at A. This is because when P is false at time t and Q is true at time $t + dt$ the transition assertion will be satisfied. If it must be guaranteed that no parts appear at B if no parts have been put at A then transition model 2 is used:

2. Multiple Transition with Negation

$$MTN\ d\ D\ P\ Q = \exists dt.\ \forall t.\ (d < dt) \wedge (dt < D) \wedge (P\ t =\ Q\ t\ (t + dt))$$

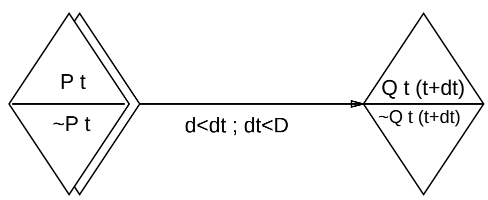

Figure 3. Multiple Transition with Negation Model

In this model, Q is true at time $t + dt$ if and only if P is true at t. A specification for the conveyor belt using model 2 will the be:

$$\exists\ dt.\forall\ t.30 < dt \wedge dt < 35 \wedge (part\_at\_A\ t = part\_at\_B\_painted\_red\ (t + dt))$$

Model 2 can not be used in cases were two transitions update the same variable, unless an additional assertion is used. Consider the following example in figure 4a.

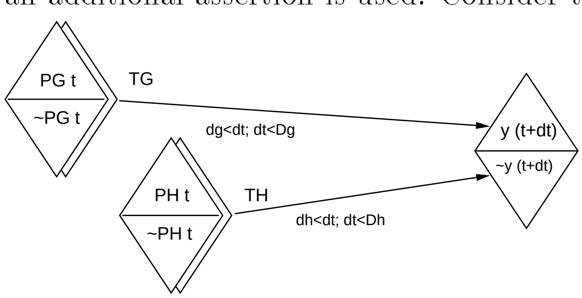

Figure 4a. Logical Disjunction for MTN

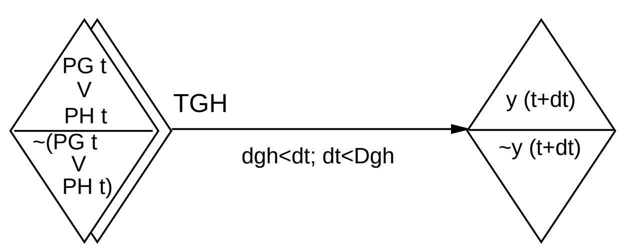

Figure 4b. Logical Disjunction for MTN; common delay

Transitions TG and TH update variable y. Let PG be always false which will force y to be always false. If PH is true at t then y is true at $t + dt$ which clearly contradicts the first statement. The specification can only be satisfied if PG and PH are always false. When transition model 2 is used as in figure 4a, the specification is redefined as follows:

$TG = MTN\ dg\ Dg\ PG\ update\_g$
$TH = MTN\ dh\ Dh\ PH\ update\_h$
$\forall t.update\_g\ t \vee update\_h\ t = y\ t$

Each transition updates a variable *update*. The third statement makes y true if one or both of the *update* variables is true, and false otherwise. The third statement is the logical disjunction of the Multiple Transition with Negation model. If the transition times for TG and TH are equivalent, the definition can be simplified to one transition as in figure 4b and one statement:

$TGH = MTN\ dgh\ Dgh\ (PG \vee PH)\ y$

A specific example of this feature is shown in section 4.2, figure 21. Transitions T1 and T4 will both cause a capture_current_speed if maintain speed or increase speed are activated. Capture_current_speed will be false if neither of these conditions take place.

The third basic model is the single transition model. In model 3 only one transition can be occurring at the same time. Once a transition has started, making condition P true will not cause a new transition. The definition and graphical representation of single transition model 3 is:

3. Single Transition

$$
\begin{array}{l}
ST\ d\ D\ P\ Q\ trans = \\
(\forall t. \\
\quad (P\ t) \wedge (\neg trans\ t) \Rightarrow \\
\quad \exists dt. \\
\qquad (d < dt) \wedge \\
\qquad (dt < D) \wedge \\
\qquad (Q\ t(t + dt)) \wedge \\
\qquad (\forall k.(0 < k) \wedge (k < dt) \Rightarrow (trans(t + k))) \wedge \\
\qquad (\neg trans(t + dt))) \wedge \\
(\forall t.(\neg P\ t) \wedge (\neg trans\ t) \Rightarrow (\neg trans(t + 1))) \wedge \\
(\neg trans\ 0)
\end{array}
$$

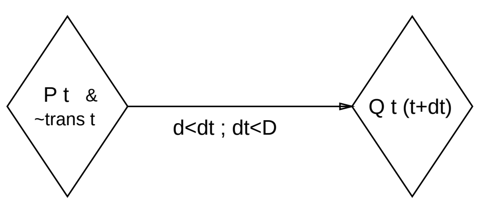

Figure 5. Single Transition Model

The variable *trans* represents whether a transition is occurring or not and it is true if a transition is taking place and false if it is not. As shown on the definition of model 3 above, a transition will take place if P is true and *trans* is false.

The following example illustrates the single transition model. System S, figure 6, has a data input N, a control input *start*, and a data output SQRN.

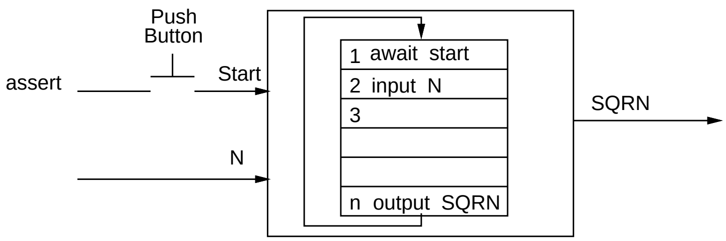

Figure 6. Single Transition Example

When the control line *start* is asserted by the push button, the system latches the value of data line N, computes the square root of the value, and puts the result on line SQRN. Activating the push button at any time during computation will have no effect. The system will effectively ignore this signal during transition. The behaviour of system S, and model 3, is illustrated further by the timing diagram of figure 7.

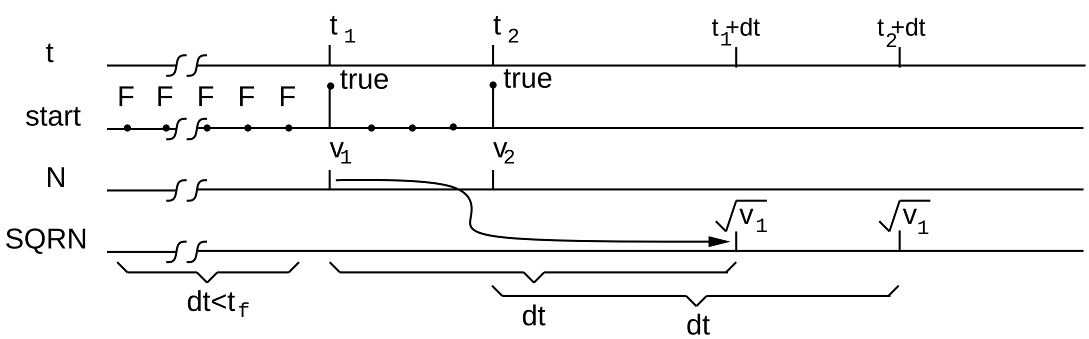

Figure 7. Single Transition Timing Diagram

At time t_1, with the value v_1 on input N, the line *start* is asserted after being false for a time longer that dt. At time t_2, line start is asserted in the middle of the transition. The transition continues without interruption and at time $t_1 + dt$ the value $\sqrt{v_1}$ appears on output SQRN. Asserting *start* at time t_2 has been ignored and therefore the value on SQRN is not updated to $\sqrt{v_2}$ at time $t_2 + dt$. The graphical representation and formal definition of system S is:

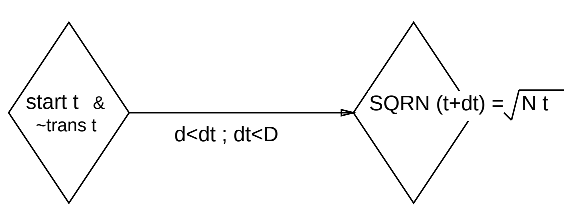

Figure 8. System S Diagram

$$ST\ d\ D\ (\lambda\ t.start\ t)\ (\lambda\ t\ t'.SQRN(t') = \sqrt{N(t)})\ trans\_sqr$$

3.2 The Five Derived Transition Models

Transitions 4 through 8 are derived from the basic transition models and include inertia, and preemption.

3.2.1 Inertia

Inertia refers to the attribute a variable, parameter or condition has to retain its present value indefinitely unless a process or transition occurs and changes its value. The Transition Assertion generic model does not inherently have inertia. If an assertion states that the value of variable x is 7 at time 1732, $x(1732) = 7$, then the value of x at 1733 is undefined unless specifically defined by another assertion.

Inertia is implemented in the transition models by an auxiliary inertia assertion. The auxiliary inertia assertion states that the value of a variable does not change:
$\forall\ t.x(t) = x(t-1)$
Variable x is then of little use since it always has the same value and can not be changed by any process. The auxiliary assertion is then modified to state that the value of the variable does not change unless a transition is updating its value. A variable *update* is included in the transition model, and the transition model makes *update* equal to true when it is changing variable values and false any other time. The auxiliary inertia assertion is then, not updating, stay the same:
$\forall\ t.\neg update\ t \Rightarrow x(t) = x(t-1)$

If two or more transitions can change the value of a variable *var* then the auxiliary assertion is extended to be:
$\forall\ t.\neg update1\ t \wedge \neg update2\ t \wedge ... \Rightarrow var(t) = var(t-1)$
The assertion states that if none of the transitions that can update *var* are updating it at time t then the value of *var* stays the same.

Transition models with inertia are models 4 and 6. Model 4 is defined next.

4. Multiple Transition with Inertia

$$
\begin{array}{l}
MTi\ d\ D\ P\ Q\ update = \\
\exists dt. \\
\quad (\forall j.j < dt \Rightarrow \neg update\ j) \wedge \\
\quad (\forall t. \\
\quad\quad ((d < dt) \wedge (dt < D) \wedge (P\ t \Rightarrow (Q\ t\ (t + dt)))) \wedge \\
\quad\quad (P\ t = update(t + dt))
\end{array}
$$

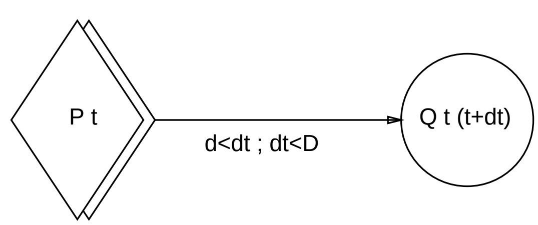

Figure 9. Multiple Transition with Inertia Model

Model 4 is based on model 1 with the variable *update* added to implement inertia. Variable *update* is true at the end of the transition, when condition Q must be true, and false at all other times.

To illustrate model 4, two transitions (figure 10) from the cruise control example of section 4.2 are used[1]. Inertia is implemented on the maintain speed request variable m_req.

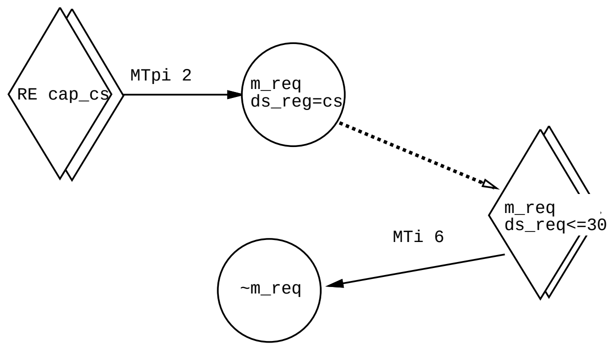

Figure 10. Multiple Transition with Inertia Example

Variable m_req is set to true by transition T2 when the variable cap_cs, capture current speed, changes from false to true[2]. The request is canceled, by resetting m_req to false in transition T6, if the desired speed is less than or equal to 30 miles per hour. The textual representation including the auxiliary assertion is:

$$T2 = MTi\ \ d\ D2\ (\lambda\ t.RE\ cap\_cs\ t)\ (\lambda\ t\ t'.(m\_req\ t') \wedge ((ds\_reg\ t') = (cs\ t)))\ update2$$
$$T6 = MTi\ \ d\ D6\ (\lambda\ t.(m\_req\ t) \wedge ((ds\_reg\ t) <= 30))\ (\lambda\ t\ t'.\neg m\_req\ t')\ update6$$
$$m\_req\_i = \forall\ t.((\neg update2\ t) \wedge (\neg update6\ t)) \Rightarrow (m\_req\ t) = (m\_req(t-1))$$

The value of m_req will remain unchanged if both transitions T2 and T6 are inactive.

3.2.2 Preemption

Preemption is implemented in the next model. Preemption is the capability of interrupting a transition while the transition is taking place. Transition models with preemption assert that if P is true then Q is true at a later time unless a condition W is true during some interval before the transition is over. Transition models with preemption are 5, 6, 7 and 8.

The definition and graphical representation of transition model 5 is:

5. Multiple Transition with Preemption

$MTp\ d\ D\ P\ Q\ W =$
$(\exists dt.$
$\quad \forall t.$
$\quad (d < dt) \wedge (dt < D) \wedge (\forall k.(0 < k) \wedge (k < dt) \Rightarrow \neg W(t+k)) \Rightarrow$

[1]Preemption was eliminated in transition T2 to simplify the example. Preemption is discussed in subsection 3.2.2.

[2]The predicate RE, raising edge, returns true at time t when its argument changes from false to true from $t-1$ to t. It is defined by $RE\ sig\ t = (((sig(t-1)) = F) \wedge ((sig\ t) = T))$.

$P\ t \Rightarrow$
$(Q\ t\ (t + dt)))$

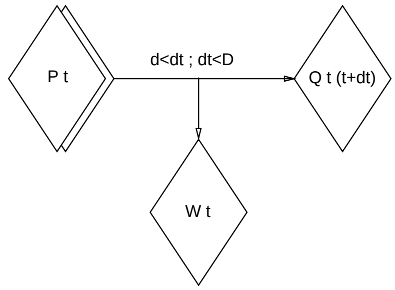

Figure 11. Multiple Transition with Preemption Model

An example of model 5 is transition T5 in the cruise control example of section 4.2. In transition T5 (figure 12), if there exists a maintain speed request, desired speed is greater than 30 miles per hour and the brake and the increase button are not activated, then the maintain speed actuator will be enabled and the maintain speed request cleared. If, however, the brake is activated during transition T5, then the transition will be interrupted and the maintain speed actuator will not be enabled.

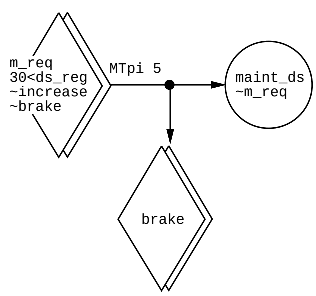

Figure 12. Multiple Transition with Preemption Example

Model 6 includes both preemption and inertia and is defined next.

6. Multiple Transition with Preemption and Inertia

$$
\begin{array}{l}
MTpi\ d\ D\ P\ Q\ W\ update = \\
(\exists dt. \\
\quad (\forall j. j < dt \Rightarrow \neg update j) \wedge \\
\quad (\forall t. \\
\qquad (d < dt) \wedge \\
\qquad (dt < D) \wedge \\
\qquad ((\forall k. (0 < k) \wedge (k < dt) \Rightarrow \neg W(t + k)) \Rightarrow \\
\qquad\quad (P\ t \Rightarrow (Q\ t\ (t + dt))) \wedge (P\ t = update(t + dt))) \wedge \\
\qquad (\neg(\forall k. (0 < k) \wedge (k < dt) \Rightarrow \neg W(t + k)) \Rightarrow \neg update(t + dt)))
\end{array}
$$

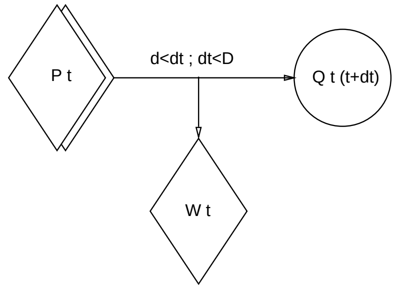

Figure 13. Multiple Transition with Preemption and Inertia Model

As an example of model 6, transition T2 from the cruise control is used again, this time with preemtion (figure 14).

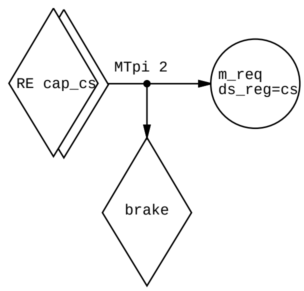

Figure 14. Multiple Transition with Preemption Example

Inertia is implemented on variable m_req. m_req is set to true by transition T2 unless condition $brake$ becomes true at any time during T2, at which point the transition will terminate and the variable m_req will not be changed. Since variable m_req has inertia, the value will remain constant after the transition terminates unless updated by a new transition.

Model 7 is defined next.

7. Mult. Trans. with Negation and Preemption

$$
\begin{array}{l}
MTNp \;\; d \; D \; P \; Q \; W = \\
(\exists dt. \;\; (\forall j. j < dt \Rightarrow \neg Q \; 0 \; j) \wedge \\
\quad \forall t. \\
\qquad (d < dt) \wedge \\
\qquad (dt < D) \wedge \\
\qquad ((\forall k. (0 < k) \wedge (k < dt) \Rightarrow \neg W(t + k)) \Rightarrow \\
\qquad\quad (P \; t = (Q \; t \; (t + dt)))) \wedge \\
\qquad (\neg(\forall k. (0 < k) \wedge (k < dt) \Rightarrow \neg W(t + k)) \Rightarrow (\neg Q \; t \; (t + dt))))
\end{array}
$$

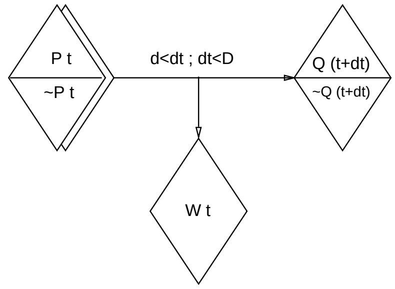

Figure 15. Mult. Trans. with Negation and Preemption Model

Model 7 is based on model 2 with preemption added. For model 7, Q is true at time $t + dt$ if and only if P is true at t and W is false at all times between t and $t + dt$. Going back to the conveyor belt example, P is part_at_A, Q is part_at_B_painted_red and the condition W could represent emergency_power_off. That is, a part will appear at point B painted red at time $t + dt$ if and only if a part was put at A at time t and the power was not shut off between t and $t + dt$. The transition assertions graphical representation is in figure 16.

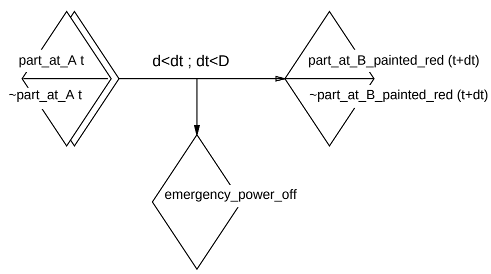

Figure 16. Mult. Trans. with Negation and Preemption Example

The last transition model is model 8 defined below.

8. Single Transition with Preemption

$$
\begin{array}{l}
STp\ d\ D\ P\ Q\ W\ trans = \\
(\forall t. \\
\quad P\ t \wedge\ trans\ t \Rightarrow \\
\quad (\exists dt. \\
\qquad d < dt \wedge \\
\qquad dt < D \wedge \\
\qquad ((\forall k. 0 < k \wedge k < dt \Rightarrow\ W(t + k)) \Rightarrow \\
\qquad\quad Q\ t\ (t + dt) \wedge \\
\qquad\quad (\forall k. 0 < k \wedge k < dt \Rightarrow trans(t + k)) \wedge \\
\qquad\quad trans(t + dt)) \wedge \\
\qquad (\exists t'. \\
\qquad\quad t < t' \wedge t' <= (t + dt) \wedge W\ t' \Rightarrow \\
\qquad\quad (\forall i. 0 < i \wedge i <= t' \Rightarrow trans(t + i)) \wedge\ trans(t' + 1)))) \wedge \\
(\forall ti.\ Pti \wedge\ transti \Rightarrow\ trans(ti + 1))
\end{array}
$$

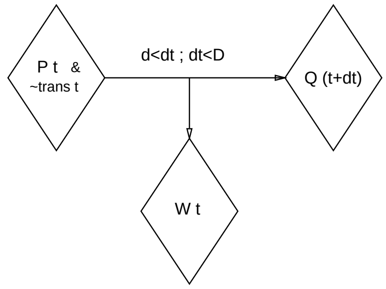

Figure 17. Single Transition with Preemption Model

Model 8 is similar to model 3 with preemption added. As an example of model 8, the example shown in figure 6 is reproduced in figure 18 with a reset input added.

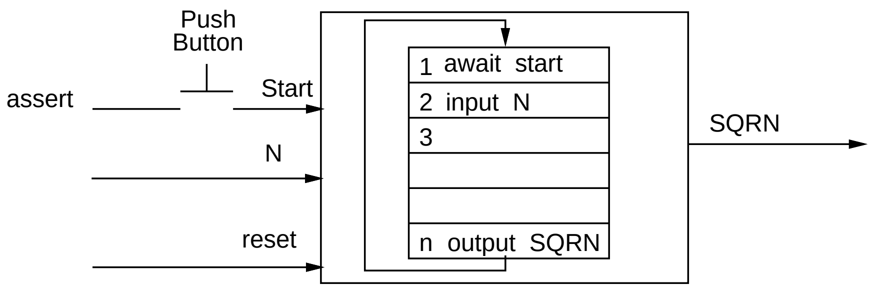

Figure 18. Single Transition with Preemption Example

If start is activated, the system will calculate the square root of the value N and put the calculated value on SQRN. Activating start at any time during the transition will have no effect. If line reset is asserted during the transition, the transition will terminate enabling a new transition to begin by activating start. When a transition is terminated, the value of output SQRN will be undifined if the variable SQRN does not have inertia. If inertia is implemented on SQRN, the value at termination will be the value the variable had before the transition started.

4 Cruise Control Example

4.1 Description

The example is a motor car cruise control used to maintain constant the speed of the car during extended motoring. This example was chosen to evaluate the transition assertion method against the objectives listed in the introduction. The cruise control example was extracted from reference [10]. In the reference, a method called Extended System Modeling Language (ESML), based on data flow diagrams, is used for the description of the cruise control.

For this paper, only the *control speed* module of the cruise control was used. The *control speed* module is represented by a state diagram in the ESML model and is reproduced in this paper in figure 19. The arcs of the state diagrams represent the transitions between states. The enabling condition for the transition to occur is shown above the line on the arc label. The action that will take place when the transition occurs is shown under the line. For example, if the system is in state IDLE and the condition RESUME;(DS>30) becomes true, the system will transition to state MAINTAINING with the action *enable maintain speed* taking place.

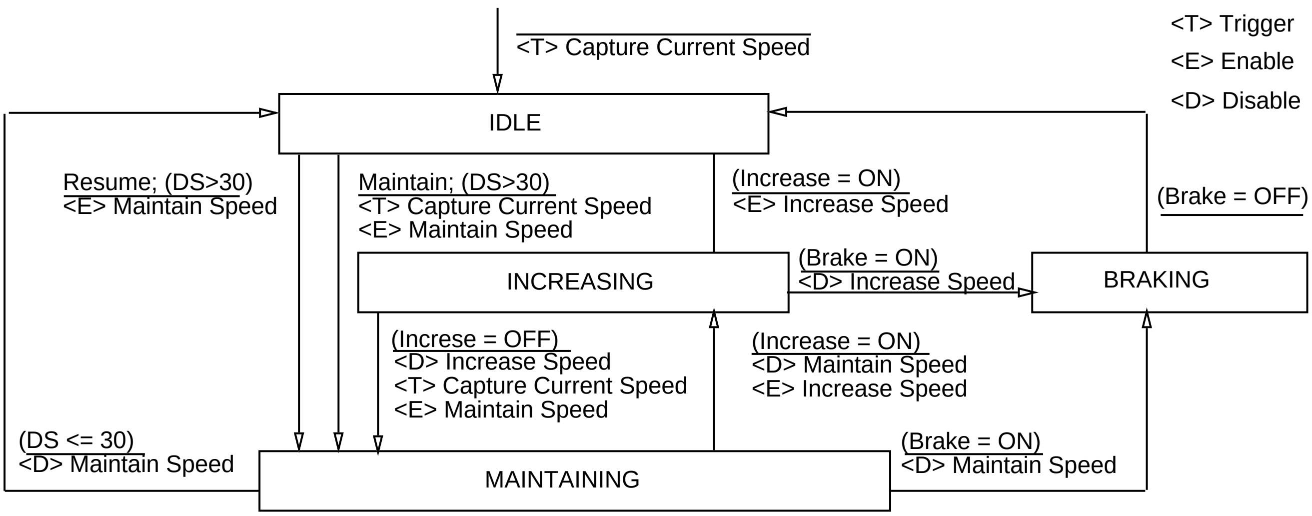

Figure 19. Control Speed State Diagram

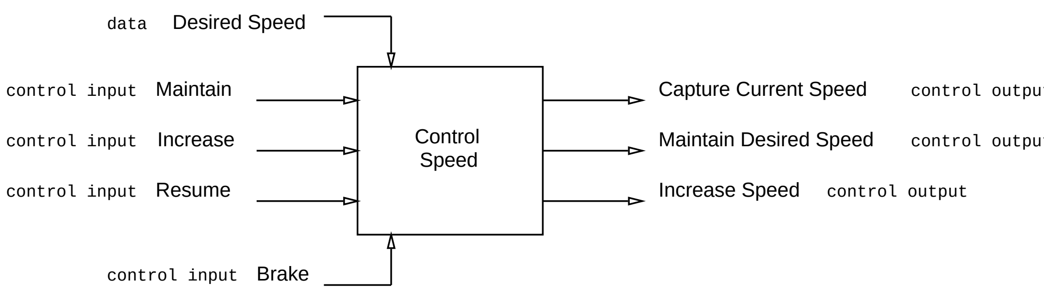

Figure 20. Control Speed Module

The *control speed* module (figure 20) has 4 control inputs, 1 data input and 3 control outputs. The control lines are operated directly by the driver through a control panel (with push buttons) and the motor car brake. Pressing the maintain button will cause the speed to remain constant if the speed is above 30 miles per hour. Pushing the brake will cause the control to disengage, remembering the set speed. Pressing resume will cause the speed to change to the previous setting, if there was one, and then held constant. Pressing and holding increase will cause the speed to gradually increase and then remain constant after release.

The data input is the captured speed which will be stored in a desired-speed memory register. This register could be internal or external to the control module depending on the implementation.

The 3 control outputs activate/deactivate the maintain and increase actuators linked to the throttle and trigger the capture speed mechanism.

4.2 Specification using Transition Assertions

In order to specify the system behaviour using transition assertions, some assumptions had to be made. Assumptions were necessary because some information cannot be extracted from the state diagram. For example, the state of the system, when pressing and holding the increase button and then pressing resume, cannot be determined from the state diagram. The assumptions were made to produce a system which the author thought was desirable and safe.

The transition assertion graphical specification is shown in Figure 21:

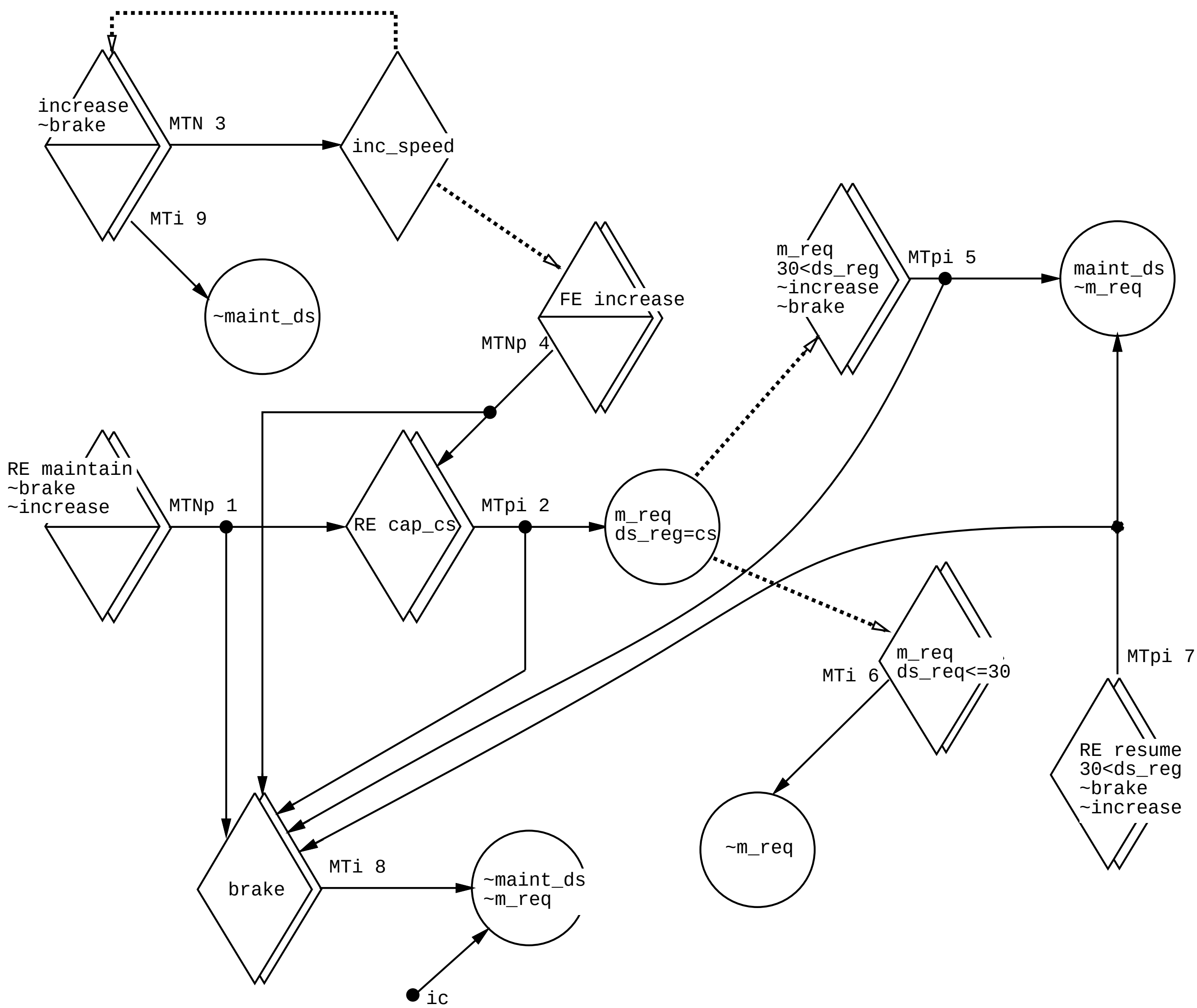

Figure 21. Transition Assertion Specification of Cruise Control

The specification uses 9 transitions and one initial condition, represented by the solid lines, to define the behaviour. The dashed lines in the diagram do not represent transitions and are used to connect two conditions that may occur simultaneously. The dashed lines do not have any semantical connotation and can be added or deleted without changing the specification. Transitions 1, 2, 4, 5, and 7 are preempted by the same condition *brake*. Each transition corresponds to an assertion and each variable with inertia requires an auxiliary assertion. The variables with inertia in this example are *m_req*, *ds_reg*, and *maint_ds*. If two or more transitions with negation update the same variable, an ORing assertion will be necessary for updating the variable. In this specification the variable *cap_cs*, capture current speed, has such property. The predicates FE and RE take a signal and a time and return true if the signal is a falling edge or rising edge respectively. The definition of RE is given in the foot note, page 8, and the definition of FE is: $FE\ sig\ t = (((sig(t-1)) = T) \wedge ((sig\ t) = F))$

The corresponding textual specification is given next:

$$
\begin{array}{lcl}
T1 & = & MTNp\ \ d\ D1\ (\lambda\ t.(RE\ maintaint) \wedge (\neg braket) \wedge (\neg increaset)) \\
 & & (\lambda\ t\ t'.update1\ t')\ (\lambda\ tp.brake\ tp) \\
T2 & = & MTpi\ \ d\ D2\ (\lambda\ t.RE\ cap\_cs\ t)\ (\lambda\ t\ t'.(m\_req\ t') \wedge ((ds\_reg\ t') = (cs\ t))) \\
 & & (\lambda\ tp.brake\ tp)\ update2 \\
T3 & = & MTN\ \ d\ D3\ (\lambda\ t.(increase\ t) \wedge (\neg brake\ t)) \\
 & & (\lambda\ t\ t'.inc\_speed\ t') \\
T4 & = & MTNp\ \ d\ D4\ (\lambda\ t.(FE\ increase\ t) \wedge (\neg brake\ t)) \\
 & & (\lambda\ t\ t'.update4\ t')\ (\lambda\ tp.brake\ tp) \\
T5 & = & MTpi\ \ d\ D5\ (\lambda\ t.(m\_req\ t) \wedge (30 < (ds\_reg\ t)) \wedge (\neg increaset) \wedge (\neg braket)) \\
 & & (\lambda\ t\ t'.(maint\_ds\ t') \wedge (\neg m\_req\ t'))\ (\lambda\ tp.brake\ tp)\ update5 \\
T6 & = & MTi\ \ d\ D6\ (\lambda\ t.(m\_req\ t) \wedge ((ds\_reg\ t) <= 30))\ (\lambda\ t\ t'.\neg m\_req\ t')\ update6 \\
T7 & = & MTpi\ \ d\ D7\ (\lambda\ t.(RE\ resume\ t) \wedge (30 < (ds\_reg\ t)) \wedge (\neg brake\ t) \wedge (\neg increaset)) \\
 & & (\lambda\ t\ t'.(\neg inc\_speed\ t') \wedge (maint\_ds\ t') \wedge (\neg m\_req\ t'))(\lambda\ tp.brake\ tp)\ update7 \\
T8 & = & MTi\ \ d\ D8\ (\lambda\ t.brake\ t) \\
 & & (\lambda\ t\ t'.(\neg maint\_ds\ t') \wedge (\neg m\_req\ t'))\ update8 \\
T9 & = & MTi\ \ d\ D9\ (\lambda\ t.(increase\ t) \wedge (\neg brake\ t))(\lambda\ t\ t'.\neg maint\_ds\ t')\ update9 \\
ic & = & (\neg maint\_ds\ 0) \wedge (\neg m\_req\ 0) \wedge (\forall\ k.(k < (D3-1)) \Rightarrow (\neg inc\_speed\ k))
\end{array}
$$

$$
\begin{array}{lcl}
maint\_ds\_i & = & \forall\ t.((\neg update5\ t) \wedge (\neg update7\ t) \wedge (\neg update8\ t) \wedge (\neg update9\ t)) \Rightarrow \\
 & & ((maint\_ds\ t) = (maint\_ds(t-1))) \\
m\_req\_i & = & \forall\ t.((\neg update2\ t) \wedge (\neg update5\ t) \wedge (\neg update6\ t) \wedge (\neg update7\ t) \wedge \\
 & & (\neg update8\ t)) \Rightarrow ((m\_req\ t) = (m\_req(t-1))) \\
ds\_reg\_i & = & \forall\ t.\neg update2\ t|Rightarrow((ds\_reg\ t) = (ds\_reg(t-1))) \\
cap\_cs\_or & = & \forall\ t.(update1\ t) \vee (update4\ t) = (RE\ cap\_cs\ t) \\
SPECIF & = & T1 \wedge T2 \wedge T3 \wedge T4 \wedge T5 \wedge T6 \wedge T7 \wedge T8 \wedge T9 \wedge ic \wedge maint\_ds\_i \wedge \\
 & & m\_req\_i \wedge ds\_reg\_i \wedge cap\_cs
\end{array}
$$

As in the graphical representation, there are 9 transitions and an initial conditions assertion. The auxiliary assertions are explicitly shown by 3 inertia assertions and an ORing assertion for variable *cap_cs*. The specification is the conjunction of all the assertions as shown by the last statement.

5 Verification

Three top level requirements, to insure safe operation of the system, were formulated to use in the example. Although many other requirements are also desirable, the 3 chosen were thought to be representative.

The Verification process is then conducted by constructing a mathematical proof showing that the specification implies the requirements. The proofs were done with the HOL theorem prover.

The requirements are informally described as follow:

1. If at any given time the brake is activated, the control will disable the maintain current speed and increase speed actuators.

2. If the button increase is never pushed, the increase speed actuator will never be enabled.

3. If buttons increase, maintain and resume are never pushed, the maintain current speed actuator will never be enabled.

The higher order logic representation of the requirements are:

1. $R1 = \forall\, t.(brake\ t) \Rightarrow \exists\, t'.(\neg inc\_speed\ t') \wedge (\neg maint\_ds\ t')$

2. $R2 = (\forall\, t.(\neg increase\ t)) \Rightarrow \forall\, t.(\neg inc\_speed\ t)$

3. $R3 = (\forall\, t.\neg RE\ maintain\, t) \wedge (\forall\, t.\neg RE\ resume\, t) \wedge (\forall\, t.\neg increase\, t) \Rightarrow \forall\, t.(\neg maint\_ds\, t)$

It was shown, using the HOL theorem prover, that the specification implies each of these requirements. The first proof, $SPECIF \Rightarrow R1$, was not difficult since transition T8 and T3 guarantees this requirement. The second proof was also easily derived. The third proof, however, was very difficult. Thirty four lemmas were pre-proved and used to show $SPECIF \Rightarrow R3$.

The difficulty on the third proof arises from the fact that, contrary to the first two proofs, it is necessary to show that certain transitions never occur. That is, the preconditions of the transitions leading to $maintain\_ds = true$ are always false.

In general, proving that a system will not behave in a given way (or that a system will not enter a given state) is more difficult than proving that a system will act in a predetermined way. There are two reasons for this: First, it is sufficient to give an example to show that a system will reach a state, but all possible paths must be examined to show that it will not. Second, specifications are written to define how the system will perform, rather than how the system will not perform.

During the proofs, some errors were found in the specification and some changes had to be made to comply with the requirements. One term was also added to the initial conditions to comply with one of the requirements.

6 Conclusion

The Transition Assertions specification method was successfully used to define a real-time control system. It was demonstrated that properties of the specification can be mathematically proven. The level of difficulty of the proofs was high. However, it was found that many of the lemmas needed for the third proof had similar structures. As the Transition Assertions theory is expanded, many lemmas can be reused, making the proofs easier and the method more useful.

One of the problems yet to be addressed is the fact that inconsistent specifications can be written, especially as the complexity of the system increases. A consistency proof of the specification appears to be difficult and time consuming at the moment.

Future work on the Transition Assertions method could include executable specifications to aid in the understanding of the system behaviour. Simulation of specification is incorporated in other specification methods and has been implemented for higher order logic in previous works [2, 3].

7 Acknowledgements

The idea for Transition Assertions are based on Mike Gordon's work on State Transition Assertions. Most of the work on Transition Assertions and the proofs for the cruise control was done at the Cambridge University Computer Laboratory under NASA sponsorship.

References

[1] Rajeev Alur and Thomas A. Henzinger, *Logics and Models of Real Time: A Survey,* To appear in the proceedings of the 1991 REX workshop "Real Time: Theory in Practice" (Springer-Verlag LNCS series).

[2] A.J. Camilleri, *Executing Behavioral Definitions in Higher Order Logic* University of Cambridge Computer Laboratory, Technical Report No. 140, July 1988.

[3] V.A. Carreño, *Definition and Partial Verification of Data Routing Circuit in Higher Order Logic*, 10 July 1991.

[4] M.J.C. Gordon, *A Formal Method for Hard Real-Time Programming.*

[5] D. Harel, *Statecharts: A Visual Formalism for Complex Systems*, Sci. Computer Program., vol. 8, pp. 231-274, 1987.

[6] Leslie Lamport, *A Temporal Logic of Actions,* Technical Report 57, Digital Equipment Corporation, Systems Research Center, April 1, 1990.

[7] Aloysius K. Mok, *Towards Mechanization of Real-Time System Design,* (workshop on foundations of real-time computing, Office of Naval Research, Oct 1990) in Foundation

of Real-Time Computing: Formal Specifications and Methods, Andre M. van Tilborg and Gary M. Koob, editors, Kluwer Academic Publishers, 1991.

[8] B. C. Moszkowski, Zohar Manna, *Reasoning in Interval Temporal Logic*, Report No. STAN-CS-83-969, Stanford University, July 1983.

[9] A. Pnueli, *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends*, Current Trends in Concurrency (deBakker et al. eds.) Lecture Notes in Computer Science, Vol. 224, Spring-verlag, Berlin, 1986, pp 510-584.

[10] C. Ingvar Svensson *ESML: An Extended System Modeling Language Based on the Data Flow Diagram* Appendix B, NASA Contract Report 187526, Oct. 1991.

[11] P.A. Zave, *A Distributed Alternative to Finite-State-Machine Specification,* ACM Transactions on Programming Language Systems, 7, 1 (Jan 1985), 10-36.