, , 1–38 ()

Specification and Verification of Reactive System Behaviour: The Railroad Crossing Example

JAMES ARMSTRONG jma@minster.york.ac.uk
Dependable Computer Systems Centre, University of York, York Y01 5DD, U.K.

LEONOR BARROCA l.barroca@open.ac.uk
Department of Computing, The Open University, Milton Keynes MK7 6AA, U.K.

Abstract. In this paper we present an approach to the specification and verification of reactive systems. The approach uses Timed Statecharts and Real Time Logic for the specification of temporal behaviour, and theorem proving techniques for the verification of safety and utility properties. Formal verification is achieved through the automation of semi-formal (rigorous) proofs using a theorem prover (Proofpower HOL). To illustrate the approach, we use the Railroad Crossing Problem, which has been proposed, along with a set of criteria for assessment, as a benchmark for the comparison of real-time formalisms. We conclude with our assessment of the approach against the proposed criteria.

Keywords: Temporal Behaviour Specification, Reactive Systems, Rigorous Proof, Automated Proof, Safety Properties, Utility Properties, Timed Statecharts, Real Time Logic, Proofpower HOL

1. Introduction

The Railroad Crossing problem has been proposed as a benchmark for the comparison of different formalisms for real-time system specification [5]. This paper uses the problem to illustrate an approach based on the combination of Timed Statecharts [12], [11], Real Time Logic [9] and automated theorem proving techniques with Proofpower HOL [6]. We discuss the applicability of the approach according to a number of criteria for expressivity and usability which have been identified in [3].

Our approach involves the derivation of axiomatisations in Real Time Logic from Timed Statecharts; theorem proving is then used to verify that timing properties for safety and utility follow from the axioms. Proofs are initially developed in a "rigorous", rather than fully formal way; that is, they are sketched on paper, with steps which appear obvious, or which follow from the properties of arithmetic, omitted. This rigorous proof is then used as a design for a proof script which provides the missing logical detail. Having two proofs, one rigorous and readable, the other unreadable but fully formal, enables us to make the best use of the capabilities of the user and of the theorem proving tool. Current provers are unable to extract rigorous arguments from their internal representation of a proof. Provers which do a large amount of reasoning on their own do not necessarily help, since proofs often hinge crucially on very simple pieces of logical reasoning. The "cornerstones" of a

proof need to be brought into relief by the author of the proof, so that others can understand it.

The Railroad Crossing problem was first proposed as an example application of the Modechart formalism [10]. This formalism is similar to ours, though there are important differences. In the course of the paper we make some comparisons between our formalism and Modecharts.

Our main concern in this paper is with the specification of temporal behaviour. System functionality is dealt with in separate parts of our formalism, which we do not cover here.

2. Formal Approach

The principles of our approach have been reported in [1], [4] and will be briefly reviewed here. Timing behaviour is expressed in terms of events, states, and transitions between states. State transitions are instantaneous; they can be triggered either by the occurrence of an event, or after some time has elapsed without the system having changed state. In the latter case, this amount of time is described as an interval with a lower and an upper bound; the time spent in the state before the transition is triggered is at least greater than or equal to the lower bound and cannot exceed the upper bound.

Transitions triggered by events occur instantaneously with the event. Although it is problematic to argue that a cause and its effect can be simultaneous, the assumption can be understood in terms of the discrete time-base required in RTL. We assume that the event and transition appear to occur at the same time because the delay between them is too small (i.e. less than one time unit) to be measurable on the given discrete time-scale.

Real Time Logic [8], [7] is a formal logic for reasoning about the timing behaviour of real-time systems. It deals with time quantitatively, and so has no modal operators. RTL is used to reason about individual occurrences of events; its basic concepts are *events*, *actions* (represented by the occurrences of start and stop events), and *state predicates*. The *occurrence* relation θ captures the notion of time: $\theta(ev, i, t)$ asserts that the ith occurrence of event ev happens at time t. Formulas in RTL use the universal and existential quantifiers and first order logic.

Temporal behaviour expressed in Timed Statecharts can be translated into a set of RTL axioms describing the properties of each state machine. Also produced are axioms which describe how machines drive other machines through communication. We term these axioms "progress axioms".

The proof theory of RTL can then be employed in the validation of the global timing constraints imposed on the system using our simple two-stage proof method. We use Proofpower HOL [6], though any prover in which RTL can be axiomatised is suitable. By using RTL we have adopted the *event-action* model of computation as proposed by [8].

As already mentioned, the use of Timed Statecharts and RTL limits us to the specification and verification of temporal behaviour; some other notation is needed

for specifying the functionality of system activities. Fortunately, several suitable formalisms already exist; we have found Z satisfactory, for example.

However, because the work reported in this paper is concerned with temporal behaviour, activities are not covered in our example; but in brief, we associate activities with states, and assume that they take a non-zero time to execute. Atomic actions, such as assignments to variables, on the other hand, are associated with transitions, rather than states, thus keeping the number of states to a minumum. Atomic actions are assumed to be instantaneous.

In timed statecharts, communication between state machines is established through the detection of transition events, which in turn trigger transitions in other machines. In the label of a transition we make a simple distinction between the *trigger* and its *effect*.

The trigger may be an event, with or without a condition, or a time interval; in other words, there is a distinction between event-triggered and non-event-triggered transitions. Variables may be tested in the condition of a trigger.

The effect includes any events generated by the transition. In addition, events marking the exit of the source state and the entry of the target state are generated.

Activities "residing" in states take a (non-zero) minimum amount of time to execute, which is expressed by the lower bound on a transition out of that state. The upper-bound is used to specify the maximum execution time.

2.1. Brief Comparison with Modecharts

Although our formalism is similar to the Modechart formalism [7], [10], it differs in two major ways: firstly, in the way that communication between state machines is represented, and secondly, we do not distinguish between mode and state variable.

Communication happens when the instantaneous generation of events by the effect part of a transition label in one machine triggers transitions in other machines. In Modecharts, communication occurs through the updating and subsequent reading of state variables. State variables cannot change value instantaneously. This has several implications, the most obvious of which is that communication cannot induce mutual cycles in Modecharts, i.e. a machine cannot drive another machine which in turn drives it cyclically, without time progressing.

Nonetheless, the distinction between state variable and mode which is made in Modecharts is, we feel, largely unnecessary; we have found that given the right interpretation, the use of RTL state variables alone is sufficient for the representation of the temporal behaviour captured by a state diagram. However, we have had to alter the RTL logic slightly: we assume that it is possible for a state variable to make an instantaneous transition from true to false or vice versa, whereas, in ordinary RTL, state variables must hold their value for a measurable amount of time. In Appendix A, the state transition axioms of RTL have been weakened to allow a state variable to change state instantaneously.

This alteration enables us to model a state as a simple state variable, which is true when the state is entered, and goes false when it is exited. Otherwise, our

states are treated in the same way as Modecharts modes, and can be entered and exited simultaneously.

We are therefore forced to deal with the possibility of infinite cycles occurring within machines without time progressing. We do this by imposing constraints on the problem, rather than on the formalism: a tractable problem must guarantee either that a minimum amount of time is spent in a certain state, (imposing an interval with a lower bound greater than 0 on the transitions out of the state), or that consecutive occurrences of the same event are separated by some interval (this is an imposition made by the model of the environment).

In section 4 we show how reasonable assumptions about the Railroad Crossing problem can enable us to ensure that infinite cycles cannot occur.

2.2. Summary—Assumptions of the formalism

Before presenting our axiomatisations and proofs, we briefly review the assumptions underlying our formalism:

1. Transitions are instantaneous;

2. Arcs that leave and enter the same state S (loops) may mark the occurrence of an event, but they do not generate events of the form $S := F$; i.e. the state does not change;

3. Events that trigger a transition, or that are generated by a transition, occur instantaneously with that transition;

4. Labels on a transition are of the form:
   $event[condition]/actions$ or $[l, u]/actions$
   Any part of the label is optional; the actions on the right side of the label can consist of the creation of an event or represent the invocation of some operation (typically consisting of assignment to variables).

5. Time-consuming activities are associated with a state; the start of the operation corresponds to the transition into that state and the end of the operation corresponds to the exiting of that state. We assume that increasing or decreasing the value of a variable in a transition's action takes no measurable time.

6. The possibility that an event e which causes a transition in some state S, occurs while the system is not in state S, is ignored. Such events have no impact on the progress of the state machine, even though the possibility that e occurs outside S may exist in the problem domain.

7. State variables may transit from true to false instantaneously. This makes the transition axioms and state predicate definitions of RTL invalid. We therefore use weaker axioms. In each transition axiom described in [9] we replace $<$ by $\leq$. We also add an additional axiom stating that a variable may only be either

true or false at time zero (though recall that it may instantaneously change its value after that).

3. Derivation of axioms from Timed Statecharts

This section presents rules for the translation of our Timed Statecharts into RTL theories, forming the basis for the automatic derivation. The set of derived axioms is by no means minimal; however, systematic derivation can at least guarantee that the theory is complete.

Each machine will have an initial state and possibly a condition set upon initialisation. These are translated into initial state axioms.

For each state, there will be input and output axioms for every outgoing and incoming transition, respectively. These axioms only apply to transitions between different states. For loops, that is, transitions where the source and the target state are the same, the axioms will assert the effect of the labels without generating the exit event of the state.

Progress axioms guarantee that, for each individual machine, a transition is picked from the set of transitions which are currently enabled. Stability axioms capture the fact that when no stimuli occur, a machine remains in its current configuration. Thus stability and progress axioms capture our semantics for the firing of transitions.

3.1. A syntax for the definition of Statemachines

Each individual statemachine has an initial state associated with an initial condition.

$$
\begin{array}{lcrcl}
Machine & :: & initial & : & StateName \\
 & & initialcond & : & Condition \\
 & & states & : & \mathbb{P}\, State \\
State & :: & input & : & \mathbb{P}(Source \times Label) \\
 & & name & : & StateName \\
 & & output & : & \mathbb{P}(Target \times Label) \\
Source & = & StateName & & \\
Target & = & StateName & &
\end{array}
$$

A label will have a trigger and an effect. The trigger will be either an enabling condition or a delay. We disallow the use of an enabling condition and a delay in the same label. We have not come across any example where such a combination is necessary and the translation rules for it would be unnecessarily complex. The effect of a transition will be the generation of one or more events. Atomic actions such as assignments are treated as such events.

Triggers can therefore be divided into immediate, for transitions which are taken when an event occurs if a certain condition is true, and delayed. The latter transitions are taken within an interval relative to the time the source state was entered.

$$
\begin{array}{lcrcl}
\mathit{Label} & :: & \mathit{trigger} & : & \mathit{ITrigger} \mid \mathit{DTrigger} \\
 & & \mathit{effect} & : & \mathit{EventName}^* \\
\mathit{ITrigger} & :: & \mathit{event} & : & \mathit{EventName} \\
 & & \mathit{condition} & : & \mathit{Condition} \\
\mathit{DTrigger} & :: & l & : & \mathbb{N} \\
 & & u & : & \mathbb{N} \mid \infty
\end{array}
$$

A semantic constraint is that l has to be less or equal than u.

3.2. Translation Rules

The rules defined below apply to single state machines; where several state machines are specified this set of rules has to be applied to each individually. This assumes that the states of each machine are unique, and form disjoint sets.

In the rule schematas below (see Figure 1), *TRIGGER* should be expanded for either an immediate label type ($e[c]$, which we term *ITRIGGER*) or a delayed label type ($[l, u]$, *DTRIGGER*). The expansion is as follows (t refers to the variable t bound in the expression where the expansion is done)[1]:

$$
\begin{array}{ll}
\mathit{TRIGGER} & \\
e[c]\ (\mathit{ITRIGGER}) & \exists\, j : \mathit{Occ} \bullet \theta(e, j, t) \land \mathit{holds}(c, t) \\
[l, u]\ (\mathit{DTRIGGER}) & \exists\, t_1 : \mathit{Ti} \bullet \theta(S_1 := T, i, t_1) \land t_1 + l \leq t \leq t_1 + u
\end{array}
$$

EFFECT should also be expanded as follows (again, t refers to the variable t bound in the expression where the expansion occurs):

$$
\begin{array}{l}
\mathit{EFFECT} \\
\exists\, i \ldots k : \mathit{Occ} \bullet \theta(e_1, i, t) \land \ldots \land \theta(e_n, k, t)
\end{array}
$$

The derived RTL axioms fall into six categories: initial state axioms, output axioms, input axioms, progress axioms, loop axioms, and stability axioms.

3.2.1. Initial State Axioms

A state is either an initial state or a non-initial state. For each state there will be an initial state axiom.

A condition can be true or false on entering the initial state; we will treat changes in the truth value of conditions as events. There is one single initial state in each machine; the initial state is initially true (see Figure 1, diagram(i)),

$$
\begin{array}{ll}
I_{cond} & \theta(\mathit{cond}_1, 1, 0) \\
I_1 & \theta(S_1 := T, 1, 0)
\end{array}
$$

all the other states (S_i), in the machine, are initially false.

$$
I_i \qquad \theta(S_i := F, 1, 0)
$$

Figure 1. Schemas for the different types of rules for statemachines

Because they are easier to grasp, we will give the rules for single output and single input transitions first. Note however, that they are simply special cases of the rules for multiple output and multiple input transitions which we give later.

3.2.2. *Output Axioms*

Single Output

A single output transition will generate the following set of axioms (see Figure 1, diagram(ii)):

- SO_1—a rule relating exiting the source state (S_1) with entering the target state (S_2);
- SO_2—a rule that relates exiting the source state with the trigger of the transition;
- SO_3—a rule that relates exiting the source state with the effect of the transition.

SO_{2i} and SO_{2d} show the expansion of SO_2 for an immediate and a delayed trigger, respectively.

The generation rules assume that S_1 is initially true; if this is not the case then we have to ensure that $i > 1$ on the left hand side of the implication, and in SO_{2d} the index i will be replaced by $i - 1$ on the expansion of *TRIGGER* on the right hand side. Here, these two different versions of SO_{2d} are shown explicitly.

$$
\begin{array}{ll}
SO_1 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \Rightarrow \exists\, j : Occ \bullet \theta(S_2 := T, j, t) \\
SO_2 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \Rightarrow TRIGGER \\
SO_{2i} & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \Rightarrow \exists\, j : Occ \bullet \theta(e, j, t) \wedge holds(c, t) \\
SO_{2d} & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \Rightarrow \\
 & \quad \exists\, t_1 : Ti \bullet \theta(S_1 := T, i, t_1) \wedge t_1 + l \leq t \leq t_1 + u \\
SO_{2d} & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \wedge i > 1 \Rightarrow \\
 & \quad \exists\, t_1 : Ti \bullet \theta(S_1 := T, i - 1, t_1) \wedge t_1 + l \leq t \leq t_1 + u \\
SO_3 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \Rightarrow EFFECT
\end{array}
$$

SO_3 can be strengthened considerably given some extra information on the event generated in *EFFECT*; if this event is generated by this transition exclusively, the implication can be replaced by an equivalence. In this case, the occurrence index on the right side of the equivalence will be the same as that on the left (for an initially false state the index on the right hand side will be $i - 1$ and there is the imposition of $i > 1$).

Multiple Outputs

For each state S_1 with multiple output transitions the following axioms will apply (see Figure 1, diagram(iii)):

- MO_1—leaving S_1 implies that one of the labels is true and the corresponding target state is entered;
- MO_2—leaving S_1 implies that one of the effects (in the labels) is generated and the corresponding target state is entered.

Once again, these rules assume that S_1 is initially true; if this is not the case, then there is the imposition of $i > 1$, and the index i inside *TRIGGER* on the right hand side of the implication is replaced by $i - 1$. For example, in MO_1, if *TRIGGER* expands to a delayed trigger, then i in the expansion should be $i - 1$ as in:
$(\ldots \theta(S_1 := T, i-1, t_1) \wedge \ldots)$

$$
\begin{array}{ll}
MO_1 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \Rightarrow \\
 & \quad TRIGGER_2 \wedge \exists\, j : Occ \bullet \theta(S_2 := T, j, t) \vee \ldots \vee \\
 & \quad TRIGGER_n \wedge \exists\, j : Occ \bullet \theta(S_n := T, j, t) \\
MO_2 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := F, i, t) \Rightarrow \\
 & \quad EFFECT_2 \wedge \exists\, j : Occ \bullet \theta(S_2 := T, j, t) \vee \ldots \vee \\
 & \quad EFFECT_n \wedge \exists\, j : Occ \bullet \theta(S_n := T, j, t)
\end{array}
$$

Note that in the single output rules we have three rules (one for the target, one for the trigger, and one for the effect) instead of two; MO_1 is a generalisation of SO_1 and SO_2, and MO_2 is a generalisation of SO_1 and SO_3.

3.2.3. *Input Axioms*

Single Input

A single input transition will correspond to the following set of axioms (see Figure 1, diagram(ii)):

- SI_1—a rule that relates entering the target state with leaving the source state;
- SI_2—a rule that relates entering the target state with the trigger of the transition;
- SI_3—a rule that relates entering the target state with the effect of the transition.

SI_{2i} and SI_{2d} show the expansion of SI_2 for an immediate and a delayed trigger, respectively. In all these axioms, we assume that S_2 is initially false ; if S_2 is initially true then we must add the condition $i > 1$.

$$
\begin{array}{ll}
SI_1 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_2 := T, i, t) \Rightarrow \exists\, j : Occ \bullet \theta(S_1 := F, j, t) \\
SI_2 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_2 := T, i, t) \Rightarrow TRIGGER \\
SI_{2i} & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_2 := T, i, t) \Rightarrow \exists\, j : Occ \bullet \theta(e, j, t) \wedge holds(c, t) \\
SI_{2d} & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_2 := T, i, t) \Rightarrow \\
 & \quad \exists\, t_1 : Ti, \exists\, j : Occ \bullet \theta(S_1 := T, j, t_1) \wedge t_1 + l \leq t \leq t_1 + u \\
SI_3 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_2 := T, i, t) \Rightarrow EFFECT
\end{array}
$$

Multiple Inputs

For each state S_1 with input transitions the following axioms apply (see Figure 1, diagram(iv)):

- MI_1—entering S_1 implies that one of the input labels is true;
- MI_2—entering S_1 implies that one of the effects has been generated.

In all these axioms, we assume that S_1 is initially false ; if S_1 is initially true then we must add the condition $i > 1$. We assume that S_2 $(\ldots S_n)$ is initially true; if this is not the case then $j > 1$ (on the right hand side), and for a delayed trigger the expansion is: $\exists\, t_1 : Ti \bullet \theta(S_2 := T, j-1, t_1) \wedge t_1 + l \leq t \leq t_1 + u$

$$
\begin{array}{ll}
MI_1 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := T, i, t) \Rightarrow \\
 & \quad \exists\, j : Occ \bullet TRIGGER_2 \wedge \theta(S_2 := F, j, t) \vee \ldots \vee \\
 & \quad \exists\, j : Occ \bullet TRIGGER_n \wedge \theta(S_n := F, j, t) \\
MI_2 & \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := T, i, t) \Rightarrow \\
 & \quad EFFECT_2 \wedge \exists\, j : Occ \bullet \theta(S_2 := F, j, t) \vee \ldots \vee \\
 & \quad EFFECT_n \wedge \exists\, j : Occ \bullet \theta(S_n := F, j, t)
\end{array}
$$

Note that MI_1 is a generalisation of SI_1 and SI_2, and MI_2 is a generalisation of SI_1 and SI_3.

3.2.4. Progress axioms

Progress for single output

- PSO_1—there will be a progress axiom for each single output transition but this axiom will be different for immediate and delayed transitions.

PSO_{1i}—Progress axiom for single output immediate transition (i.e. the trigger, $ITRIGGER$, is of type e[c]).

As before, we need two different versions of PSO_{1i}, the first for states that are initially true, and the second for states that are initially false[2].

$$
\begin{array}{ll}
PSO_{1i} & \forall\, i : Occ, \forall\, t : Ti \bullet ini(S_1, i, t) \wedge ITRIGGER \Rightarrow \theta(S_1 := F, i, t) \\
PSO_{1i} & \forall\, i : Occ, \forall\, t : Ti \bullet ini(S_1, i, t) \wedge ITRIGGER \Rightarrow \theta(S_1 := F, i+1, t)
\end{array}
$$

PSO_{1d}—Progress axiom for single output delayed transitions (i.e. the trigger is of type [l,u]).

As before, PSO_{1d} has two versions, the first for states that are initially true, and the second for states that are initially false.

$$
\begin{array}{ll}
PSO_{1d} & \forall\, i : Occ, \forall\, t : Ti \bullet (S_1 := T, i, t) \Rightarrow \\
 & \quad \exists\, t_1 : Ti \bullet \theta(S_1 := F, i, t_1) \wedge t + l \leq t_1 \leq t + u \\
PSO_{1d} & \forall\, i : Occ, \forall\, t : Ti \bullet (S_1 := T, i, t) \Rightarrow \\
 & \quad \exists\, t_1 : Ti \bullet \theta(S_1 := F, i+1, t_1) \wedge t + l \leq t_1 \leq t + u
\end{array}
$$

Note in the second version that if S_1 is initially false, the right hand side of the implication expands to:
$\theta(S_1 := F, i+1, t_1) \wedge \ldots$

Progress for multiple outputs

For multiple outputs we consider immediate and the delayed transitions separately.

Given several exit transitions, some delayed and some immediate, we require that when an immediate transition is enabled due to an event occurrence and/or a condition being true, the state is indeed exited. However, if a delayed transition is also ready, we do *not* want the axioms to discount the possibility of the delayed transition being the one that is actually taken. $ITRIGGER_2 \vee \ldots \vee ITRIGGER_n$ are the triggers of the immediate transitions out of S_1.

$$PMO_{1i} \qquad \forall\, i : Occ, \forall\, t : Ti \bullet ini(S_1, i, t) \wedge (ITRIGGER_2 \vee \ldots \vee ITRIGGER_n) \Rightarrow \theta(S_1 := F, i, t)$$

or (for initially false states)

$$PMO_{1i} \qquad \forall\, i : Occ, \forall\, t : Ti \bullet ini(S_1, i, t) \wedge (ITRIGGER_2 \vee \ldots \vee ITRIGGER_n) \Rightarrow \theta(S_1 := F, i+1, t)$$

Furthermore, if there are several delayed transitions out of a state, progress implies that the state must be exited within some time which is within the minimum of all the upper bounds $(u_2, \ldots, u_n)$, relative to the time of entry to that state.

$$PMO_{1d} \qquad \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := T, i, t) \Rightarrow \exists\, t_1 : Ti \bullet \theta(S_1 := F, i, t_1) \wedge t_1 \leq t + minimum(u_2, \ldots, u_n)$$

or (for initially false states)

$$PMO_{1d} \qquad \forall\, i : Occ, \forall\, t : Ti \bullet \theta(S_1 := T, i, t) \Rightarrow \exists\, t_1 : Ti \bullet \theta(S_1 := F, i+1, t_1) \wedge t_1 \leq t + minimum(u_2, \ldots, u_n)$$

3.2.5. *Loop Axioms*

The previous axioms apply to transitions with disjoint source and target states. In the case that the source and target are the same state, there is no progress as such, i.e. the state is not exited and re-entered. Instead an axiom that relates the trigger and the effect in the label of the loop will be generated. Only an immediate type trigger is allowed in this case; if S_1 is initially false $\neg\theta(S_1 := F, i+1, t)$.

$$L_1 \qquad \forall\, i : Occ, \forall\, t : Ti \bullet ini(S_1, i, t) \wedge TRIGGER \Rightarrow EFFECT \wedge \neg\theta(S_1 := F, i, t)$$

Where there are other transitions exiting state S_1 then the generated axiom is as follows ($\neg\theta(S_1 := F, i+1, t)$ if S_1 is initially false):

$$\begin{array}{ll} L_1 & \forall\, i : Occ, \forall\, t : Ti \bullet ini(S_1, i, t) \wedge TRIGGER_2 \\ & \quad \neg(ITRIGGER_3 \vee \ldots \vee ITRIGGER_n) \wedge \\ & \quad t_1 \leq t < t_1 + minimum(l_3, \ldots, l_n) \Rightarrow \\ & \qquad EFFECT_2 \wedge \neg\theta(S_1 := F, i, t) \end{array}$$

Note that t_1 is bound in the definition of $ini \ldots \theta(S_1 := T, i, t_1)$.

3.2.6. *Stability Axioms*

As a complement to the progress axioms, the stability axioms state that if no exit transition is enabled, the state will not be left.

Stability for single output

Stability axiom for single output immediate (the trigger, *ITRIGGER*, is of type e[c]) transition[3].

$$SSO_{1i} \qquad \neg ITRIGGER \Rightarrow \neg\, \exists\, i \bullet \theta(S_1 := F, i, t)$$

A stability axiom for the single output delayed transition (where the trigger is of type [l,u]) is unnecessary, since the non-occurrence of events is irrelevant.

Stability for multiple outputs

The stability rules for multiple outputs ensure that:

1. if we are in a certain state at some point in time and no trigger of an immediate transition ($e[c]$) is enabled,

   and

2. that the current time is earlier than the minimum of the lower bounds of any delayed transitions,

then the state will not be exited.

As before there are two versions of SMO_{1i}, the first for states initially true, and the second for states that are initially false.

$$\begin{array}{ll} SMO_{1i} & \forall\, i : Occ, \forall\, t, t_1 : Ti \bullet ini(S_1, i, t) \wedge \\ & \quad \neg(ITRIGGER_2 \vee \ldots \vee ITRIGGER_n) \wedge \\ & \quad t_1 \leq t < t_1 + minimum(l_2, \ldots, l_n) \Rightarrow \\ & \qquad \neg\theta(S_1 := F, i, t) \\ SMO_{1i} & \forall\, i : Occ, \forall\, t, t_1 : Ti \bullet ini(S_1, i, t) \wedge \\ & \quad \neg(ITRIGGER_2 \vee \ldots \vee ITRIGGER_n) \wedge \\ & \quad t_1 \leq t < t_1 + minimum(l_2, \ldots, l_n) \Rightarrow \\ & \qquad \neg\theta(S_1 := F, i+1, t) \end{array}$$

Note that t_1 is bound to the existentially quantified t_1 in the definition of *ini* (see previous footnote).

In addition to the axioms generated from these rules,constraints must also be placed on the time bounds used. An upper bound is always greater or equal to the lower bound, and by convention, if an upper bound is given explicitly then it should be greater than 0.

4. Problem Statement

The example system is a controller which operates a railroad crossing. There will be multiple tracks, and trains entering the crossing may be travelling in either direction. The crossing controller must manage a "critical region" which extends to either side of the crossing gate. Any train entering or leaving this region is detected by sensors at its boundaries.

Informally, the safety property of this system is that the crossing gate must not be up whilst any train is in the critical region. Since this constraint is met by a system which keeps the gate down at all times, a utility property must also be met: the gate should be up as often as is practical without compromising safety. For a more detailed description we refer the reader to [5].

5. Constraints on The Railroad Crossing Problem

In this section, we describe a number of assumptions about the Crossing problem which make it more amenable to our formalism.

Firstly, our model concentrates on the critical region and does not require any detailed modelling of the crossing. We simply assume that there is a minimum interval between a train entering the critical region and its passing in front of the gate; in other words, trains have some maximum speed which cannot be exceeded.

Secondly, we stipulate that a train is considered to be out of the critical region when it has left the crossing. That is, the critical region for an individual train is shorter than the actual critical region for all trains. Since trains can run in both directions, the critical region extends some distance either side of the crossing. However, an individual train is clear when it has left the crossing, under the assumption that trains never reverse direction. So, if a single train is in the critical region, traffic will not have so long to wait.

In Figure 2 we show the behaviour of the sensor system and the gate. The sensor system detects a train entering (*inregion*) the critical region and increments a counter. If there are currently no other trains in the critical region (i.e. the counter was previously zeroed) it generates a *lower* event. When a train is detected leaving the crossing (*exitregion*) the counter is decreased; if there are no more trains in the region then the sensor system goes back to its initial state and generates a *raise* event for the gate. We assume that initially the sensor is in state *None in Region* and that the counter is zero.

The gate is initially in state (Up) and starts moving down as soon as it detects a *lower* event. It will take at most ϵ_1 units of time to go down; it will stay down until it detects a *raise* event, taking ϵ_2 units of time to go up. If, while the gate is moving up, a *lower* event is detected, it reverses to moving down.

We make the following assumptions about the system environment:

- A train that has entered the critical region (*inregion*) will remain in that region for at least ϵ_1 units of time;
- A train may not stay forever in the critical region nor may it reverse, otherwise the gate would remain down for ever. These two assumptions combine in the following axiom;

$$\forall\, i : Occ, \forall\, t : Ti \bullet \theta(inregion, i, t) \Rightarrow \exists\, t_1 : Ti \bullet \theta(exitregion, i, t_1) \\ \wedge\ t + \epsilon_1 \leq t_1 \leq t + MAX$$

  Hence, there is a minimum separation of ϵ_1 between a *lower* event and a corresponding *raise* event. This discounts the possibility of an infinite loop between *MvUp* and *MvDown*; the gate cannot oscillate between *Up* and *Down* without ever reaching either state.

- For an *exitregion* to occur there must have been an *inregion* which occurred previously:

$$\forall\, i : Occ, t : Ti \bullet \theta(exitregion, i, t) \Rightarrow \\ \exists\, t_1 : Ti \bullet \theta(inregion, i, t_1) \wedge t_1 + \epsilon_1 \leq t$$

- Two or more trains cannot enter the critical region at the same instant in time; two *inregion* events cannot occur simultaneously due to RTL's monotonicity axioms (see Appendix A). This is not a realistic assumption and a real system that counted one train instead of two in this situation would not be safe. However, our concern in this paper has been to keep the specification example simple. We could have modelled the environment in a more realistic way by using distinguished events *inregionNorth*, and *inregionSouth* for each direction, likewise for *exitregion* events, and count variables. The signal to raise the gate would only be sent in the case that both counts drop to zero.

5.1. Specification of Timing Assertions

Using the set of rules defined in section 3 we derive the set of axioms for each of the statemachines shown in Figure 2.

We introduce operations *inc*, *dec*, *zero* to increment, decrement or set to zero the value of a counter variable[4]. This counter variable is treated as a global variable. We have also shortened the names of certain states of the gate and sensor, as follows: *Some*, *None*, *MvUp*, *MvDown*.

Figure 2. Timed Statecharts for the railroad crossing problem

Sensor Axioms

$SENSOR \triangleq \{None, Some\}$

Initial State Axioms
Icond
$S(1)\ \theta(zero(count), 1, 0)$
I1
$S(2)\ \theta(None := T, 1, 0)$
Ii
$S(3)\ \forall S_i : STATE \bullet S_i \in SENSOR \wedge S_i \neq NONE \Leftrightarrow$
$\theta(S_i = F, 1, 0)$

Output Axioms
SO1
$S(4)\ \forall i : Occ, \forall t : Ti \bullet \theta(None := F, i, t) \Rightarrow \exists j : Occ \bullet \theta(Some := T, j, t)$
$S(5)\ \forall i : Occ, \forall t : Ti \bullet \theta(Some := F, i, t) \wedge i > 1 \Rightarrow$
$\exists j : Occ \bullet \theta(None := T, j, t)$
SO2
$S(6)\ \forall i : Occ, \forall t : Ti \bullet \theta(None := F, i, t) \Rightarrow \exists j : Occ \bullet \theta(inregion, j, t)$
$S(7)\ \forall i : Occ, \forall t : Ti \bullet \theta(Some := F, i, t) \wedge i > 1 \Rightarrow$
$\exists j : Occ \bullet \theta(exitregion, j, t) \wedge holds(count = 1)$
SO3
$S(8)\ \forall i : Occ, \forall t : Ti \bullet \theta(None := F, i, t) \Rightarrow$
$\exists j, k : Occ \bullet \theta(lower, j, t) \wedge \theta(inc(count), k, t)$
$S(9)\ \forall i : Occ, \forall t : Ti \bullet \theta(Some := F, i, t) \wedge i > 1 \Rightarrow$
$\exists j, k : Occ \bullet \theta(raise, j, t) \wedge \theta(dec(count), k, t)$

Input Axioms
SI1
$S(10)\ \forall i : Occ, \forall t : Ti \bullet \theta(None := T, i, t) \wedge i > 1 \Rightarrow$
$\exists j : Occ \bullet \theta(Some := F, j, t)$
$S(11)\ \forall i : Occ, \forall t : Ti \bullet \theta(Some := T, i, t) \Rightarrow \exists j : Occ \bullet \theta(None := F, j, t)$
SI2
$S(12)\ \forall i : Occ, \forall t : Ti \bullet \theta(None := T, i, t) \wedge i > 1 \Rightarrow \exists j : Occ \bullet$
$\theta(exitregion, i, t) \wedge holds(count = 1)$
$S(13)\ \forall i : Occ, \forall t : Ti \bullet \theta(Some := T, i, t) \Rightarrow \exists j : Occ \bullet$
$\theta(inregion, j, t)$
SI3
$S(14)\ \forall i : Occ, \forall t : Ti \bullet \theta(None := T, i, t) \wedge i > 1 \Rightarrow$
$\exists j, k : Occ \bullet \theta(raise, j, t) \wedge \theta(dec(count), k, t)$
$S(15)\ \forall i : Occ, \forall t : Ti \bullet \theta(Some := T, i, t) \Rightarrow$
$\exists j, k : Occ \bullet \theta(lower, j, t) \wedge \theta(inc(count), k, t)$

Progress Axioms
PSO1
$S(16)\ \forall\, i : Occ, \forall\, t : Ti \bullet ini(None, i, t) \wedge \exists\, j : Occ \bullet \theta(inregion, j, t) \Rightarrow$
$\theta(None := F, i, t)$
$S(17)\ \forall\, i : Occ, \forall\, t : Ti \bullet ini(Some, i, t) \wedge \exists\, j : Occ \bullet \theta(exitregion, j, t) \wedge$
$holds(count = 1) \Rightarrow \theta(Some := F, i + 1, t)$

Loop Axioms
$S(18)\ \forall\, i : Occ, \forall\, t : Ti \bullet ini(Some, i, t) \wedge \exists\, j : Occ \bullet \theta(inregion, j, t) \Rightarrow$
$\exists\, k : Occ \bullet \theta(inc(count), k, t)$
$S(19)\ \forall\, i : Occ, \forall\, t : Ti \bullet ini(Some, i, t) \wedge holds(count > 1) \wedge$
$\exists\, j : Occ \bullet \theta(exitregion, j, t) \Rightarrow \exists\, k : Occ \bullet \theta(dec(count), k, t)$

StabilityAxioms
SSO1
$S(20)\ \forall\, t : Ti \bullet \nexists j : Occ \bullet \theta(inregion, j, t) \Rightarrow \nexists i : Occ \bullet \theta(None := F, i, t)$
$S(21)\ \forall\, t : Ti \bullet \nexists j : Occ \bullet \theta(exitregion, j, t) \vee holds(count \neq)1 \Rightarrow$
$\nexists i : Occ \bullet \theta(Some := F, i, t)$

Gate Axioms

$GATE \triangleq \{Up, MvDown, MvUp, Down\}$

Initial State Axioms
I1
$G(1)\ \theta(Up := T, 1, 0)$
Ii
$G(2)\ \forall\, S_i : State \bullet S_i \in GATE \wedge S_i \neq Up \Leftrightarrow$
$\theta(S_i := F, 1, 0)$

Output Axioms

SO1

$G(3)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Up := F, i, t) \Rightarrow \exists\, j : Occ \bullet \theta(MvDown := T, j, t)$

$G(4)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(MvDown := F, i, t) \wedge i > 1 \Rightarrow$
$\exists\, j : Occ \bullet \theta(Down := T, j, t)$

$G(5)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Down := F, i, t) \wedge i > 1 \Rightarrow$
$\exists\, j : Occ \bullet \theta(MvUp := T, j, t)$

SO2

$G(6)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Up := F, i, t) \Rightarrow \exists\, j : Occ \bullet \theta(lower, j, t)$

$G(7)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(MvDown := F, i, t) \wedge i > 1 \Rightarrow$
$\exists\, t_1 : Ti \bullet \theta(MvDown := T, i - 1, t_1) \wedge t_1 \leq t \leq t_1 + \epsilon_1$

$G(8)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Down := F, i, t) \wedge i > 1 \Rightarrow \exists\, j : Occ \bullet \theta(raise, j, t)$

MO1

$G(9)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(MvUp := F, i, t) \wedge i > 1 \Rightarrow$
$\exists\, j : Occ \bullet \theta(lower, j, t) \wedge \exists\, k : Occ \bullet \theta(MvDown := T, k, t) \vee$
$\exists\, t_1 : Ti \bullet \theta(MvUp := T, i - 1, t_1) \wedge t = t_1 + \epsilon_2 \wedge$
$\exists\, j : Occ \bullet \theta(Up := T, j, t)$

Input Axioms

SI1

$G(10)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Up := T, i, t) \wedge i > 1 \Rightarrow$
$\exists\, j : Occ \bullet \theta(MvUp := F, j, t)$

$G(11)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Down := T, i, t) \Rightarrow$
$\exists\, j : Occ \bullet \theta(MvDown := F, j, t)$

$G(12)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(MvUp := T, i, t) \Rightarrow \exists\, j : Occ \bullet \theta(Down := F, j, t)$

SI2

$G(13)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(MvUp := T, i, t) \Rightarrow$
$\exists\, j : Occ \bullet \theta(raise, j, t)$

$G(14)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Down := T, i, t) \Rightarrow$
$\exists\, j : Occ, \exists\, t_1 : Ti \bullet \theta(MvDown := T, j, t_1) \wedge t_1 \leq t \leq t_1 + \epsilon_1$

$G(15)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(Up := T, i, t) \wedge i > 1 \Rightarrow$
$\exists\, j : Occ, \exists\, t_1 : Ti \bullet \theta(MvUp := T, j, t_1) \wedge t = t_1 + \epsilon_2$

MI1

$G(16)\ \forall\, i : Occ, \forall\, t : Ti \bullet \theta(MvDown := T, i, t) \Rightarrow$
$\exists\, j : Occ \bullet \theta(lower, j, t) \wedge \exists\, k : Occ \bullet \theta(Up := F, k, t) \vee$
$\theta(lower, j, t) \wedge \exists\, k : Occ \bullet \theta(MvDown := F, k, t)$

Progress Axioms
PSO1
$G(17)\ \forall i : Occ, \forall t : Ti \bullet ini(Up, i, t) \wedge \exists j : Occ \bullet \theta(lower, j, t) \Rightarrow$
$\theta(Up := F, i, t)$
$G(18)\ \forall i : Occ, \forall t : Ti \bullet ini(Down, i, t) \wedge \exists j : Occ \bullet \theta(raise, j, t) \Rightarrow$
$\theta(Down := F, i + 1, t)$
$G(19)\ \forall i : Occ, \forall t : Ti \bullet \theta(MvDown := T, i, t) \Rightarrow$
$\exists t_1 : Ti \bullet \theta(MvDown := F, i + 1, t_1) \wedge t \leq t_1 \leq t + \epsilon_1$
PMO1
$G(20)\ \forall i : Occ, \forall t : Ti \bullet ini(MvUp, i, t) \wedge \exists j : Occ \bullet \theta(lower, j, t) \Rightarrow$
$\theta(MvUp := F, i + 1, t)$
$G(21)\ \forall i : Occ, \forall t : Ti \bullet \theta(MvUp := T, i, t) \Rightarrow$
$\exists t_1 : Ti \bullet \theta(MvUp := F, i + 1, t_1) \wedge t_1 \leq t + \epsilon_2$

Stability Axioms
SSO1
$G(22)\ \forall t : Ti \bullet \nexists j : Occ \bullet \theta(lower, j, t) \Rightarrow \nexists i : Occ \bullet \theta(Up := F, i, t))$
$G(23)\ \forall t : Ti \bullet \nexists j : Occ \bullet \theta(raise, j, t) \Rightarrow \nexists i : Occ \bullet \theta(Down := F, i, t))$
SMO1
$G(24)\ \forall i : Occ, \forall t : Ti \bullet ini(MvUp, i, t) \wedge \nexists j : Occ \bullet \theta(lower, j, t) \wedge$
$t_1 \leq t \leq t_1 + \epsilon_2 \Rightarrow \neg\, \theta(MvUp := F, i + 1, t)$
Note : t_1 *is bound in the expansion of ini.*
Other
$G(25)\ \epsilon_1 > 0$
$G(26)\ \epsilon_2 > 0$

5.2. Verification of Timing Assertions

In [5], the occupancy intervals were defined as being: $\lambda_i = [\tau_i, \nu_i]$, where τ_i is the i'th entrance of a train into the critical region if the region is otherwise empty, and ν_i represents the next time after τ_i that the critical region is again unoccupied. The occupancy intervals can be represented using the following event markers:

$$\theta(Some := T, i, \tau_i)$$
$$\theta(Some := F, i, \nu_i)$$

Recall that the **safety** property of the system is that for all times within all intervals of occupancy, the gate is down. However, there is an important implementation restriction on this property: a realistic gate will take some time (ϵ_1 at most) to lower. Consequently, the safety property can only be verified for all times after ϵ_1 within any interval of occupancy. The **utility** property of the system is that for all the times outside these intervals the gate will be up. Consider that the gate takes a maximum of ϵ_1 units of time to go down and ϵ_2 to go up. If the gate is

moving up and is requested to lower it reverses to moving down and will take some time $t \leq \epsilon_1$ to go down. However, if the gate is moving down, a request to move up cannot be attended to until after at least ϵ_1 units of time (since a train takes at least ϵ_1 to pass through its critical region). We represent the fact that the gate is down at time t as follows:

$$g(t) = 0 \Leftrightarrow \exists\, i : Occ \bullet ini(Down, i, t)$$

The same applies for the gate being up:

$$g(t) = 90 \Leftrightarrow \exists\, i : Occ \bullet ini(Up, i, t)$$

5.2.1. *Rigorous Proof of Safety*

As noted above, we can only verify that during each occupancy interval the gate will be down for all except the first ϵ_1 time units (during which it is moving down). At several points in the proofs we make use of the monotonicity axioms of RTL (see Appendix A).

$$\begin{aligned}\forall\, i : Occ, \forall\, t_1, t_2 : Ti \bullet \theta(Some := T, i, t_1) \wedge \theta(Some := F, i+1, t_2) \wedge t_2 > t_1 + \epsilon_1 \\ \Rightarrow \exists\, j : Occ, \exists\, t_3, t_4 : Ti \bullet \theta(Down := T, j, t_3) \wedge \theta(Down := F, j+1, t_4) \wedge \\ t_1 \leq t_3 \leq t_1 + \epsilon_1 \wedge t_4 = t_2\end{aligned}$$

We will consider a separate case for each state the gate can be in as the sensor enters $Some$[5]. Case 1 (The gate is up) will be shown here and Case 2 (The gate is moving up) can be found in Appendix B.

Case 1. The Gate is up

A_1 $\quad \theta(Some := T, i, t_1)$
A_2 $\quad \theta(Some := F, i+1, t_2)$
A_3 $\quad t_2 > t_1 + \epsilon_1$
A_4 $\quad ini(Up, j, t_1)$

1.	$\exists\, j, k : Occ \bullet \theta(lower, j, t_1) \wedge \theta(inc(count), k, t_1)$	by m-p(A_1, S(15))
2.	$\exists\, j : Occ \bullet \theta(lower, j, t_1)$	by $\wedge$-elim(1)
3.	$ini(Up, j, t_1) \wedge \exists\, j : Occ \bullet \theta(lower, j, t_1)$	by $\wedge$-intro(A_4,2)
4.	$\theta(Up := F, j, t_1)$	by m-p(3, G(17))
5.	$\exists\, j : Occ \bullet \theta(MvDown := T, j, t_1)$	by m-p(4, G(3))
6.	$\theta(MvDown := T, j, t_1)$	by $\exists$-elim(5)
7.	$\exists\, t_1^{'} : Ti \bullet \theta(MvDown := F, j+1, t_1^{'}) \wedge$ $t_1 \leq t_1^{'} \leq t_1 + \epsilon_1$	by m-p(6, G(19))
8.	$\theta(MvDown := F, j+1, t_1^{'}) \wedge t_1 \leq t_1^{'} \leq t_1 + \epsilon_1$	by $\exists$-elim(7)
9.	$\theta(MvDown := F, j+1, t_1^{'})$	by $\wedge$-elim(8)
10.	$j + 1 > 1$	by arithmetic
11.	$\exists\, j : Occ \bullet \theta(Down := T, j, t_1^{'})$	by m-p(9,10,G(4))

12.	$\theta(MvDown := F, 1, 0)$	by G(2)
13.	$\forall\, i : Occ, \forall\, t_1 : Occ \bullet \theta(MvDown := F, i+1, t_1) \Rightarrow$ $\exists\, t_2 : Ti \bullet \theta(MvDown := T, i, t_2) \wedge t_2 \leq t_1$	by m-p(12.,trans-ax-4)
14.	$\exists\, t_2 : Ti \bullet \theta(MvDown := T, j, t_2) \wedge t_2 \leq t_1^{'}$	by m-p(9,13)
15.	$\theta(MvDown := T, j, t_2^{'}) \wedge t_2^{'} \leq t_1^{'}$	by $\exists$-elim(14)
16.	$\theta(MvDown := T, j, t_2^{'})$	by $\wedge$-elim(15)
17.	$\theta(MvDown := T, j, t_1) \wedge \theta(MvDown := T, j, t_2^{'})$	by $\wedge$-intro(6,16)
18.	$t_1 = t_2^{'}$	by m-p(17,mono-ax-1)
19.	$t_2^{'} \leq t_1^{'}$	by $\wedge$-elim(15)
20.	$t_1 \leq t_1^{'}$	by subs of 19 using 18
21.	$i + 1 > 1$	by arithmetic
22.a.1.	*assume* $i = 1$	by cases from 21
22.a.2.	$\theta(Some := F, 1, t_2)$	by subs into A_2
22.a.3.	$\theta(Some := F, 1, 0)$	by S(3)
22.a.4.	$\epsilon_1 > 0$	by G(25)
22.a.5.	$t_2 > 0$	by arithmetic,A_3,22.a.4
22.a.6.	$\theta(Some := F, 1, t_2) \wedge \theta(Some := F, 1, 0)$	by $\wedge$-intro(22.a.2,22.a.3)
22.a.7.	$t_2 = 0$	by m-p(22.a.6, mono-ax-1)
22.a.8.	$i \neq 1$	by contr(22.a.5,22.a.7)
22.b.1.	$i > 1$	by arithmetic,21,22.a.8
22.b.2.	$\theta(Some := F, i+1, t_2) \wedge i > 1$	by $\wedge$-intro(A_2,22.b.1)
22.b.3.	$\exists\, j, k : Occ \bullet \theta(raise, j, t_2) \wedge$ $\theta(dec(count), k, t_2)$	by 22.b.2,S(9)
22.b.4.	$\theta(raise, j, t_2) \wedge \theta(dec(count), k, t_2)$	by $\exists$-elim(22.b.3)
22.b.5.	$\theta(raise, j, t_2)$	by $\wedge$-elim(22.b.4)
22.b.6.	$\forall\, t_4 : Ti \bullet t_4 < t_2 \Rightarrow$ $\neg\, \theta(raise, j, t_4)$	by mono-lemma(Appendix B)
22.b.7.	$\forall\, t_4 : Ti \bullet t_4 < t_2 \Rightarrow$ $\nexists k : Occ \bullet \theta(Down := F, k, t_4)$	by m-p(22.b.6,G(23))
22.b.8.	$\forall\, t_4 : Ti \bullet t_4 < t_2 \Rightarrow$ $\neg\, \theta(Down := F, k+1, t_4)$	by $\exists$-elim(22.b.7)
22.b.9.	$\theta(Down := F, 1, 0)$	by G(2)
22.b.10.	$t_1 \leq t_1^{'} \leq t_1 + \epsilon_1$	by $\wedge$-elim(8)
22.b.11.	$t_1^{'} < t_2$	by arithmetic,22.b.10., A_3
22.b.12.	$ini(Down, k, t_2)$	by ini-def,22.b.9,22.b.11,22.b.7
22.b.13.	$\exists\, j : Occ \bullet \theta(raise, j, t_2)$	by $\wedge$-elim(22.b.3)
22.b.14.	$ini(Down, k, t_2) \wedge$ $\exists\, j : Occ \bullet \theta(raise, j, t_2)$	by $\wedge$-intro(22.b.12,22.b.13)
22.b.15.	$\theta(Down := F, k+1, t_2)$	by m-p(22.b.14, G(18))
23.	$\exists\, j : Occ, \exists\, t_3, t_4 : Ti \bullet$ $\theta(Down := T, j, t_3)$	by 11.
	$\wedge\, \theta(Down := F, j+1, t_4)$	by $\exists$-intro 22.b.15
	$\wedge\, t_1 \leq t_3$	by 20
	$\wedge\, t_3 \leq t_1 + \epsilon_1$	by 22.b.10
	$\wedge\, t_4 = t_2$	by mono-ax-1,22.b.15

QED

6. Automatic proof using Proofpower HOL

The process of proof automation is central to our approach, and our work in this area raised some interesting practical issues.

Firstly, we found that several early rigorous proof attempts, which seemed reasonable on paper, were in fact insufficient because certain assumptions had been forgotten in our formulation of the safety and utility properties. Incorrect or incomplete formalisation of an intuitively understood property is a very common error.

Secondly, automation illustrated that the character of the axiomatisation has profound effects on the effort required to automate proofs. The problems of having such a large number of axioms were somewhat offset by their uniformity.

Thirdly, the problem of interrelating a readable formal proof with its automation in a theorem prover was an issue. In this section, we will briefly discuss the automation of the safety property proof, commenting on each of these issues in relation to it.

6.1. Automation of the safety proof

Our embedding of Real Time Logic into Proofpower HOL (see Appendix A) is based on the axiomatisation in [9]. It is a "shallow" embedding, rather than a "deep" one. That is, we have not built in a specialised semantics and proof theory for RTL, but have merely extended HOL with the RTL occurrence relation. This approach is sufficient for experimentation, so long as we do not make use of formulae, which while they may be well-formed HOL, cannot be expressed in RTL. For example, our RTL theory models events as HOL variables, which permits one to quantify over events, although this is explicitly disallowed in RTL. For higher integrity, a deep embedding would ensure that one could not stray outside of the RTL logic, whereas currently we have to avoid doing so ourselves. In addition, we currently have to derive the RTL rules from state diagrams by hand. This is error prone and we are looking at ways in which the process can be automated.

On the other hand, we have confidence that the tool is of sufficient reliability to make a deep embedding of RTL worthwhile. Proofpower's logical theories have been developed in bottom-up fashion from an extremely small set of axioms. We are forced to place trust in implementations of the tactics, but so far we have uncovered no problems with them.

Proofpower uses the ML programming language as a "metalanguage" in which to embed the HOL logic [6]. A Proofpower proof takes the form of a simple listing of function (tactic) invocations which manipulate formulae. This listing can be "played back" through the prover, and examined step-by-step, but in general it is rather difficult to read these scripts. For example, the tactics may behave differently, depending on minor details in the way a given assumption is formulated. It is therefore not easy to understand a Proofpower script as a standard Gentzen-style

deduction. This is one of our motivations for presenting rigorous proofs for human scrutiny, rather than proof scripts.

Automation begins by setting the property to be proved as the current goal:

$$
\begin{array}{l}
set{-}goal([], \ulcorner \forall\ i\ t1\ t2\ : \mathbb{N} \bullet \\
\qquad \theta(true\ Some, i, t1) \land \theta(false\ Some, i+1, t2) \land \\
\qquad (t2 > t1 + e1) \land \\
\qquad (\exists j : \mathbb{N} \bullet ini(Up, j, t1) \lor \exists j : \mathbb{N} \bullet ini(MvUp, j, t1)) \Rightarrow \\
\qquad\qquad \exists j\ t3\ t4 : \mathbb{N} \bullet \theta(true\ Down, j, t3) \land \\
\qquad\qquad \theta(false\ Down, j+1, t4) \land \\
\qquad\qquad t1 \leq t3 \land t3 \leq t1 + e1 \land \\
\qquad\qquad t4 = t2 \urcorner);
\end{array}
$$

(Note: some notational differences were imposed upon us by the tool, for example $(false\ sv)$ means $sv := F$. We have also modelled occurrence indexes and time as natural numbers). When broken down, this goal will produce two subgoals, one for $ini(Up, t1)$ and one for $ini(MvUp, t1)$:

(* *** Goal "1" *** *)

$$
\begin{array}{ll}
(*\ 4\ *) & \ulcorner \theta(true\ Some, i, t1) \urcorner \\
(*\ 3\ *) & \ulcorner \theta(false\ Some, i+1, t2) \urcorner \\
(*\ 2\ *) & \ulcorner t2 > t1 + e1 \urcorner \\
(*\ 1\ *) & \ulcorner ini(Up, j, t1) \urcorner \\
(*\ ?\vdash\ *) & \ulcorner \exists j\ t3\ t4 : \mathbb{N} \bullet \theta(true\ Down, j, t3) \land \theta(false\ Down, j+1, t4) \land \\
 & t1 \leq t3 \land t3 \leq t1 + e1 \land t4 = t2 \urcorner
\end{array}
$$

(* *** Goal "2" *** *)

$$
\begin{array}{ll}
(*\ 4\ *) & \ulcorner \theta(true\ Some, i, t1) \urcorner \\
(*\ 3\ *) & \ulcorner \theta(false\ Some, i+1, t2) \urcorner \\
(*\ 2\ *) & \ulcorner t2 > t1 + e1 \urcorner \\
(*\ 1\ *) & \ulcorner ini(MvUp, j, t1) \urcorner \\
(*\ ?\vdash\ *) & \ulcorner \exists j\ t3\ t4 : \mathbb{N} \bullet \theta(true\ Down, j, t3) \land \theta(false\ Down, j+1, t4) \land \\
 & t1 \leq t3 \land t3 \leq t1 + e1 \land t4 = t2 \urcorner
\end{array}
$$

These subgoals are solved by proof procedures which follow the steps given in the rigorous proof given above. For example, step 1 of the safety proof involves instantiating S(15) to get the term $lower(i, t1)$. This is done using the tactic $list{-}\forall{-}elim$, and the result is then "stripped" into the list of assumptions. That is, the left hand side is matched with an assumption (in this case, assumption 4 of goal 1, $\theta(true\ Some, i, t1)$, yielding the right hand side of the implication. This is accomplished by the command line:

$$a(strip-asm-tac(list-\forall-elim[\ulcorner i : \mathbb{N} \urcorner, \ulcorner t1 : \mathbb{N} \urcorner]\ sensor-axiom-15));$$

giving a new assumption 1:

$$\begin{array}{ll}
(*\ 5\ *) & \ulcorner \theta(true\ Some, i, t1) \urcorner \\
(*\ 4\ *) & \ulcorner \theta(false\ Some, i+1, t2) \urcorner \\
(*\ 3\ *) & \ulcorner t2 > t1 + e1 \urcorner \\
(*\ 2\ *) & \ulcorner ini(Up, j, t1) \urcorner \\
(*\ 1\ *) & \ulcorner \theta(lower, i, t1) \wedge (inc(count), k, t1) \urcorner \\
(*\ ?\vdash\ *) & \ulcorner \exists\ j\ t3\ t4 : \mathbb{N} \bullet \theta(true\ Down, j, t3) \wedge \theta(false\ Down, j, t4) \wedge \\
 & t1 \leq t3 \wedge t3 \leq t1 + e1 \wedge t4 = t2 \urcorner
\end{array}$$

The subsequent steps also involve the instantiation of left hand sides of implications, and so the rest of the lines in the proof procedure are largely the same.

Admittedly, our derived axiomatisations are not particularly elegant. There are many axioms and some are formally redundant. However, the uniformity imparted by the translation rules (nearly all the axioms are implications) is an advantage; more elegant and concise axiomatisations in pure RTL would require more intelligent manipulation in proofs. The uniformity of the axioms makes the rigorous proofs easy, if rather tedious, to automate.

However, a few issues were raised by the automation of the proof detail. We found that some of the "obvious" appeals to mathematics were quite difficult to automate. The level of fully automatic reasoning in Proofpower is currently rather unevenly spread. For example, we could not derive $a = b \wedge b = c \Rightarrow a = c$ without intervention, but other more complex mathematical theorems were proved automatically. Our experience is that, in general, the more automated mathematical reasoning the prover can do, and the more consistent it appears to be, the better. For example, RTL's logic and semantics are based on Pressburger arithmetic. Pressburger resolution procedures drastically reduce the effort involved in RTL proofs, and we are currently experimenting with PVS, which does incorporate them. However, we would also sound a note of caution: it is all too easy to let the prover "take over" without the user really understanding the reasoning behind the proof.

Much of the time and effort needed to discharge trivial subgoals would be reduced by a deep embedding of RTL into Proofpower. For instance, Proofpower can discharge trivial subgoals automatically due to its "knowledge" of High Order Logic, but the prover has no "knowledge" of RTL's concept of monotonicity. RTL proofs rely heavily on reducing false formulae to contradictions of monotonicity. Subgoals which violate monotonicity should ideally be discharged without user intervention. For example, an assumption that the i'th occurrence of an event happens at two different times is an obvious contradiction, but currently the user has to apply the monotonicity axioms explicitly to prove this. Much time can be wasted trying to prove a goal before such a contradiction is picked out in its lengthy assumption list.

It is our opinion based on this experience, that once theorem provers have been experimented with more widely, the industry will recognise the need for highly spe-

cialised proving tools, dedicated to specialised logics, rather than widely expressive logics suitable for embedding.

6.2. Remarks on proof as a verification method

Formal proof is a difficult and time-consuming process, so the value of a proof is sometimes judged more by the importance of what is being proved than the elegance of the proof itself. This seems to be the philosophy which underlies powerful tactic-based theorem provers, which are often capable of completing proofs without any real grasp of what is going on being conveyed to the user.

However, when an important property is proven, it is useful to know why it holds and not just that it holds, particularly when one considers the possibility that a subsequent change to a specification may invalidate the property. On the other hand, a well-designed proof strategy has two advantages. Firstly, one can make informed decisions about whether the same proof strategy can be reused to prove other theorems. Secondly, one can predict where changes to the specification will require a result to be re-proven.

A high-level design for a proof is as important as a high-level design for a program; the amount of effort spent on the rigorous proof avoids the much greater effort that would be wasted trying to salvage complicated ad hoc proof scripts which no longer run. The process of proof is very much like the process of navigation. Arriving at the right place by serendipity is possible, but hardly a sound basis for setting out. Without a proof structure, ad hoc experimentation with the prover will not help; indeed it is often difficult to know whether an alteration has made any difference or not.

For example, auto-assisted proof attempts can become bogged down for three reasons:

- The axiomatisation is incomplete. In this case, the theorem is unprovable from the theory given, even though intuition suggests that it ought to be true.
- The proof strategy being used is too periphrastic. Even though the theorem could be proved true in principle, the user becomes overwhelmed by the amount of information produced by the prover, and the number of subproofs required.
- The theorem is not true, e.g. it may have been incorrectly formulated.

Since current provers cannot indicate which of these situations is the cause of arrested progress, it is essential that the user has an overview of what is going on.

It is fairly easy to produce a proof script which follows the structure of a rigorous proof, and it is also a sure way of uncovering hidden assumptions and errors. On the one hand, to do a rigorous proof, the user must understand the process behind it at the strategic level. On the other, the ad hoc process of experimentation with the prover is inefficient and confusing, and only reproducible with some difficulty. Using our approach, if the progress of the proof is arrested, the user knows precisely

which inference step is causing the problem, and can then try to correct either proof in a focused way.

The production of two artifacts instead of a single formal proof also has certain advantages.

Firstly, the rigorous proof is a more readable artifact in system documentation than either an exhaustive formal proof or a HOL-like proof script. Admittedly, the latter can be re-executed line-by-line and observed. However, proof-tactics are powerful enough to do different things in different situations, so it is usually hard to reconstruct the progress of a proof from a proof-script.

Secondly, the approach is a very natural way of developing proofs incrementally. It involves a simple feedback loop, in which the user argues a case rigorously, attempts to automate the logical argument, and if necessary amends the rigorous argument.

Checking for correctness is not so highly automated in our approach, as with say, the model checking approach used in the Modechart toolset. Verification by proof is more time consuming and skills-intensive than "brute force" methods like model checking.

This is not to say that we disparage the model checking approach used in the Modechart toolset [13]; in fact, we envisage similar support for our formalism in the future, because the option of quick feedback is useful when experimenting with changes to a specification.

However, we feel that in the final analysis formal proof offers higher levels of assurance than model checking. A good way to assess the quality of a specification is to look at what can be argued from it, and how easily it can be argued, rather than simply checking that it is a model of some property. Proof also encourages the closer analysis of any informal arguments for safety and utility, which motivate the design in the first place. A proof script and rigorous proof serve as evidence to support the informal arguments.

7. Assessment

The Railroad Crossing example was originally proposed in [5] as a benchmark for the comparison of different formalisms. We will therefore assess our approach against the criteria proposed in [3].

- How easy it is to reason about time in the formalism? How understandable are specifications and proofs in the formalism?

  The advantage of this approach is that the specifications of temporal behaviour using Statecharts are relatively easy to use for engineers without extensive training in formal methods. However, the derived axiomatisations are somewhat clumsy, and verification requires theorem proving skills. On the other hand, for those familiar with RTL, the proofs are not difficult. Their automation is hard work, but there is the potential for improvement in the tools currently available.

- For what classes of timing properties is the formalism suitable?

Timing properties in our formalism are essentially constraints on the separation of event occurrences: i.e. if x happens at a certain moment in time, y must happen within a certain amount of time. We believe that most (if not all) timing constraints can be expressed in this way. Timeouts, for example, are easily expressed in the formalism.

- What is the quality of mechanical tools available to support the formalism?

  This is currently quite weak and we have to derive our axiomatisations manually. However, the Modechart toolset illustrates that a model-checking tools for statechart-like formalisms are a practical proposition, and advances in theorem provers are likely in the near future. There is a need for rigorously engineered theorem provers with powerful user interfaces, dedicated to specific logics.

- How general is the formalism? Is it designed to specify and verify only timing properties? What other properties can be specified and verified using the formalism?

  The formalism presented here is intended to deal only with the temporal behaviour of real-time systems. We are looking at ways of combining it with other "views" of the system, in particular with specifications of functionality.

- Is the formalism more suited to a particular phase of software development than to others, e.g., requirements rather than detailed design?

  The formalism aims to encourage specifiers to consider the specification of timing constraints as early as possible in the lifecycle. It is intended for use in the design of the *logical* architecture of the system, rather than in the detailed *physical* system design. We are currently examining the problem of relating the underlying computational model to lower level concerns, such as schedulability and resource usage, but this work is at an early stage.

- Does the formalism handle continuous as well as discrete time?

  The formalism handles discrete time only. If a discrete time scale fine enough to capture the necessary properties is chosen, we feel that the modelling of continuous time is unnecessary.

8. Summary and Conclusions

In this paper we have presented the principles of a formalism for describing the temporal behaviour of reactive systems. The approach combines the convenience of a graphical notation with the potentially high levels of integrity offered by formal proof. Information in Timed Statecharts is represented as formulae in Real Time Logic. These are then subjected to a verification process which combines rigorous handwritten proof with proof automation. The Railroad Crossing problem was used to illustrate the approach and the relevant safety and utility proofs were presented. Our preliminary assessment of the approach was also discussed.

An approach to reactive system specification which emphasises formal proof is the distinguishing characteristic of our formalism. For example, the Modechart formalism is designed to minimise the possibility of erroneous specifications by constraining the expressivity of the *language*, so that specifications are amenable to model-checking techniques. By contrast, we provide a more expressive language, e.g. by weakening the transition axioms of RTL, and require the *user* to record assumptions which discount the possibility of lack of progress, or of a system being un-implementable. Over and above the usual proof rules of first-order logic, only simple properties, such as the monotonicity properties of the occurrence relation, and the transitivity of precedence relations, need to be used in the formalism. Rigorous proofs are made readable by frequent appeals to these properties. Formal proofs constructed from a small set of rules are tedious and lengthy, but the use of a theorem prover helps to overcome this.

We are currently considering further simplifications. For example, the initialisation of state variables by events appears to be unnecessary. Intuitively, an entry i into a state should be followed by the i'th exit event, rather than exit event $i + 1$, as is the case where the state variable is initially false. This lack of uniformity can be a source of error when formulating properties to be proved. It might be better to stipulate that state variables have no provable value until such time as they are set by some event or condition.

Simplicity will continue to be the priority in the future development of the temporal formalism. The provision of multiple "views", each of which represents a certain aspect of the system, is central to the work of the Dependable Computing Systems Centre. The issues involved have been discussed in [2]. It is important that each view can be related formally to the other views. This is easier when each formalism is semantically simple. The exploration of how our formalism relates to views based on other computational models will be our greatest concern in the future.

Acknowledgments

We would like to thank John Fitzgerald, Jon Hall, Ralph Jeffords, Lynn Spencer, and the anonymous referees who have commented on an earlier draft of this paper. This work is part of the Dependable Computing Systems Centre, at the Universities of Newcastle and York, funded by British Aerospace.

Notes

1. *Occ* is the set of occurrence indexes defined as $\{i : \mathbb{N} \mid i \geq 1\}$, and *Ti* the set of times.

2. The definition of *ini* is as follows.

$$
\begin{array}{|l}
ini : State \times Occ \times T \to Bool \\
\hline
\forall S : State, i : Occ, t : Ti \bullet ini(S, i, t) \Leftrightarrow \\
\quad \theta(S := F, 1, 0) \ \wedge \\
\quad\quad \exists t_1 : Ti \bullet \theta(S := T, i, t_1) \wedge t_1 \leq t \wedge \forall t_2 : Ti \bullet t_2 < t \Rightarrow \neg\theta(S := F, i + 1, t_2) \\
\quad \vee \theta(S := T, 1, 0) \ \wedge \\
\quad\quad \exists t_1 : Ti \bullet \theta(S := T, i, t_1) \wedge t_1 \leq t \wedge \forall t_2 : Ti \bullet t_2 < t \Rightarrow \neg\theta(S := F, i, t_2)
\end{array}
$$

3. If S_1 is initially false then we also require that $i > 1$ appear on the right hand side of the implication.
4. When a condition is tested, the time at which the value of variables is tested is implicit.
5. Note that *Some* is initially false; that is why entering state *Some* for the ith time is matched with leaving state *Some* for the $i + 1$th time.

References

1. Barroca, L. 1992. An approach to architectural specification—the representation of behaviour and functionality, DCSC/TR/92/7. Dependable Computer Systems Centre, Universities of York and Newcastle-upon-Tyne, U.K.
2. Barroca, L. & J. McDermid 1993. Specification of real-time systems—a view-oriented approach. In *Proceedings of XIII Congresso da Sociedade Brasileira de Computação, XX SEMISH, Seminário Integrado de Software e Hardware, Florianópolis, Brasil*, Sociedade Brasileira de Computação.
3. Clements, P., C. Gasarch & R. Jeffords 1992. Evaluation criteria for real-time specifcation languages, Rpt. 6935. Naval Research Laboratories.
4. Fitzgerald, J. S. & L. Barroca 1993. The feasibility of providing semantics for DCSC architectural specification techniques, DCSC/TR/93/3. Dependable Computer Systems Centre, Universities of York and Newcastle-upon-Tyne, U.K.
5. Heitmeyer, C., R. Jeffords & B. Labaw 1993. Benchmark for comparing different approaches to specifying and verifying real-time systems. In *IEEE Workshop on Real-Time Operating Systems and Software*, IEEE.
6. ICL 1992. Proofpower user documention: Reference, DS/FMU/IED/USR006, issue 1.9. FST Group, ICL Computers LTD.
7. Jahanian, F., R. Lee & A. K. Mok 1988. Semantics of Modechart in Real Time Logic. In *Proceedings 21st Annual Hawai International Conference on System Science*, 479–489.
8. Jahanian, F. & A. K. Mok 1986. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, **SE-12 (9)**, 890–903.
9. Jahanian, F., A. K. Mok & D. Stuart 1988. Formal specification of real-time systems, Tr-88-25. Dept. of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712.
10. Jahanian, F. & D. Stuart 1988. A method for verifying properties of Modechart specifications. In *Proc 9th Real Time Systems Symposium*, 12–21.
11. Kesten, Y. & A. Pnueli 1991. Timed and hybrid statecharts and their textual representation. In *Formal Techniques in Real Time and Fault Tolerant Systems, LNCS 571* Vytopil, J. (ed.), 591–620. Springer Verlag.
12. Maler, O., Z. Manna & A. Pnueli 1992. From timed to hybrid systems. presented at the School on Formal Techniques in Real-Time and Fault-Tolerant Systems, Nijmegen, The Netherlands.
13. Rose, A., M. Pérez & P. Clements 1993. *Modechart Toolset User's Guide*. Human Computer Interaction Laboratory, U.S. Naval Research Laboratory, Washington, DC 20375-5320, draft edition.

Appendix A. RTL Axiomatisation

THE THEORY ASM RTL

Parents

$\mathbb{N}$ *Theorems*

Children

Railroad

Constants

$start$	$ACTION \rightarrow EVENT$
$stop$	$ACTION \rightarrow EVENT$
$true$	$STATE\_VAR \rightarrow EVENT$
$false$	$STATE\_VAR \rightarrow EVENT$
θ	$EVENT \times \mathbb{N}_1 \times \mathbb{N} \rightarrow BOOL$

Types

$EVENT$
$ACTION$
$STATE\_VAR$

Axioms

$mono\_axiom\_1$
$$\vdash \forall\, e\ i\ t1\ t2 \bullet \theta(e,i,t1) \land \theta(e,i,t2) \Rightarrow t1 = t2$$

$mono\_axiom\_2$
$$\vdash \forall\, e\ i\ t1 \bullet \theta(e,i,t1) \land i > 1 \Rightarrow (\exists\, t2 \bullet \theta(e,i-1,t2) \land t2 < t1)$$

$start\_stop\_axiom$
$$\vdash \forall\, a\ i\ t1 \bullet \theta(stop\ a,i,t1) \Rightarrow (\exists\, t2 \bullet \theta(start\ a,i,t2) \land t2 < t1)$$

trans_axiom_1

$$\vdash \forall\, sv \bullet \theta(true\ sv, 1, 0) \Rightarrow (\forall\, i\ t1 \bullet \theta(false\ sv, i, t1) \Rightarrow (\exists\, t2 \bullet \theta(true\ sv, i, t2) \wedge t2 \leq t1))$$

trans_axiom_2

$$\vdash \forall\, sv \bullet \theta(true\ sv, 1, 0) \Rightarrow (\forall\, i\ t1 \bullet \theta(true\ sv, i+1, t1) \Rightarrow (\exists\, t2 \bullet \theta(false\ sv, i, t2) \wedge t2 \leq t1))$$

trans_axiom_3

$$\vdash \forall\, sv \bullet \theta(false\ sv, 1, 0) \Rightarrow (\forall\, i\ t1 \bullet \theta(true\ sv, i, t1) \Rightarrow (\exists\, t2 \bullet \theta(false\ sv, i, t2) \wedge t2 \leq t1))$$

trans_axiom_4

$$\vdash \forall\, sv \bullet \theta(false\ sv, 1, 0) \Rightarrow (\forall\, i\ t1 \bullet \theta(false\ sv, i+1, t1) \Rightarrow (\exists\, t2 \bullet \theta(true\ sv, i, t2) \wedge t2 \leq t1))$$

state_var_init_ax

$$\vdash \forall\, sv \bullet \theta(true\ sv, 1, 0) \Leftrightarrow \neg\theta(false\ sv, 1, 0)$$

Theorems

occ_ind_mono_thm

$$\vdash \forall\, i\ j\ t1\ t2 \bullet \theta(e, i, t1) \wedge \theta(e, j, t2) \wedge i < j \Rightarrow t1 < t2$$

Appendix B. Safety Proof (continuation)

Case 2. The Gate is moving up

A_1	$\theta(Some := T, i, t_1)$
A_2	$\theta(Some := F, i+1, t_2)$
A_3	$t_2 > t_1 + \epsilon_1$
A_4	$ini(MvUp, j, t_1)$

1.	$\exists\, j, k : Occ \bullet \theta(lower, j, t_1) \wedge \theta(inc(count), k, t_1)$	by m-p(A_1, S(15))
2.	$\exists\, j : Occ \bullet \theta(lower, j, t_1)$	by $\wedge$-elim(1)
3.	$\theta(MvUp := F, j+1, t_1)$	by m-p(A_4, G(20))
4.	$j + 1 > 1$	by arithmetic
5.	$\theta(MvUp := F, j+1, t_1) \wedge j + 1 > 1$	by $\wedge$-intro(3,4)
6.*a*.	$\exists\, j : Occ \bullet \theta(lower, j, t_1) \wedge$ $\exists\, k : Occ \bullet \theta(MvDown := T, k, t_1) \vee$	
6.*b*.	$\exists\, t_2 : Ti \bullet \theta(MvUp := T, j, t_2) \wedge$ $t_1 = t_2 + \epsilon_2 \wedge \exists\, j : Occ \bullet \theta(Up := T, j, t_1)$	by m-p(5.,G(9))
7.*a*.	*The gate starts to move down*	
7.*a*.1.	$\exists\, k : Occ \bullet \theta(MvDown := T, k, t_1)$	by $\wedge$-elim(6.a)
7.*a*.2.	$\exists\, t_1^{'} : Ti \bullet \theta(MvDown := F, k+1, t_1^{'}) \wedge t_1 \leq t_1^{'} \leq t_1 + \epsilon_1$	by m-p(7.a.1,G(19))
7.*a*.3.	*follow steps* $8. - 23.$ *from Case* 1.(7.*a*.2. *for* 7.)	
7.*b*.	*The gate goes up*	
7.*b*.1.	$\exists\, j : Occ \bullet \theta(Up := T, j, t_1)$	by $\wedge$-elim(6.b)
7.*b*.2.	$\theta(Up := T, j, t_1)$	by $\exists$-elim(7.b.1)
7.*b*.3.	$\exists\, t_3 : Ti \bullet \theta(Up := F, j, t_3) \wedge t_3 < t_1$	Assumption
7.*b*.4.	$\theta(Up := F, j, t_3) \wedge t_3 < t_1$	by $\exists$-elim(7.b.3)
7.*b*.5.	$\forall\, i : Occ, \forall\, t : Ti \bullet \theta(Up := F, i, t) \Rightarrow$ $\exists\, t_2 : Ti \bullet \theta(Up := T, i, t_2) \wedge t_2 \leq t_1$	by m-p(G(1),trans-axiom-1)
7.*b*.6.	$\exists\, t_2 : Ti \bullet \theta(Up := T, j, t_2) \wedge t_2 \leq t_3$	by m-p(7.b.2,7.b.5)
7.*b*.7.	$\theta(Up := T, j, t_2^{'}) \wedge t_2^{'} \leq t_3$	by $\exists$-elim(7.b.6)
7.*b*.8.	$\theta(Up := T, j, t_2^{'})$	by $\wedge$-elim(7.b.7)
7.*b*.9.	$\theta(Up := T, j, t_1) \wedge \theta(Up := T, j, t_2^{'})$	by $\wedge$-intro(7.b.2,7.b.8)
7.*b*.10.	$t_1 = t_2^{'}$	by m-p(7.b.9,mono-axiom-1)
7.*b*.11.	$t_3 < t_1$	by $\wedge$-elim(7.b.4)
7.*b*.12.	$t_3 < t_2^{'}$	by subs(7.b.10,7.b.11)
7.*b*.13.	$t_2^{'} \leq t_3$	by $\wedge$-elim(7.b.7)
7.*b*.14.	$\neg\, (t_3 < t_2^{'})$	by arithmetic from 7.b.13
7.*b*.15.	$\neg\, (\exists\, t_3 : Ti \bullet \theta(Up := F, j, t_3) \wedge t_3 < t_1)$	by contr(7.b.12,7.b.14)
7.*b*.16.	$\forall\, t_3 : Ti \bullet \neg\, (\theta(Up := F, j, t_3) \wedge t_3 < t_1)$	by 7.b.15
7.*b*.17.	$\forall\, t_3 : Ti \bullet t_3 < t_1 \Rightarrow \neg\, \theta(Up := F, j, t_3)$	by $\Rightarrow$-intro(7.b.11,7.b.16)
7.*b*.18.	$ini(Up, j, t_1)$	by ini-def,G(1),7.b.1,7.b.17
7.*b*.19.	$ini(Up, j, t_1) \wedge \exists\, j : Occ \bullet \theta(lower, j, t_1)$	by $\wedge$-intro(7.b.18,2)
7.*b*.20.	$\theta(Up := F, j, t_1)$	by m-p(7.b.19,G(17))
7.*b*.21.	*follow steps* $5 - 23$ *from Case* 1.(7.*b*.20. *for* 4)	

The cases for *Gate is moving down* and *Gate is down* are repetitions of the steps of cases 1. and 2.

Mono-lemma

$$\forall\, t_1, t_2 : Ti \bullet \theta(e, i, t_1) \wedge t_2 < t_1 \Rightarrow \neg\theta(e, i, t_2)$$

1. $\theta(e, i, t_1) \wedge t_2 < t_1$ assumption
2. $\forall\, t_2 : Ti \bullet \theta(e, i, t_2) \Rightarrow t_2 = t_1$ mono-axiom-1
3. *Goal by contradiction*$(1, 2)$

Appendix C. Utility Proof

We need to verify that if, for any length of time greater than ϵ_2 there is no train in the critical region, then the gate is up for that length of time.

$$\forall\, i : Occ, \forall\, t, t_1, t_2 : Ti \bullet \theta(Some := F, i, t_1) \wedge \theta(Some := T, i, t_2) \wedge t_1 + \epsilon_2 < t < t_2 \Rightarrow \exists\, j : Occ \bullet ini(Up, j, t)$$

We will again consider a separate case for each state the gate might be in when the Sensor leaves state *Some*.

Case 1. The Gate is up

A_1 $\theta(Some := F, i, t_1)$
A_2 $\theta(Some := T, i, t_2)$
A_3 $t_1 + \epsilon_2 < t < t_2$
A_4 $ini(Up, k, t_1)$
A_5 $\forall\, t_3 : Ti \bullet t_1 \leq t_3 < t_2 \Rightarrow \nexists j : Occ \bullet \theta(lower, j, t_3)$

1.	$\exists\, t_1^{'} : Ti \bullet \theta(Up := T, k, t_1^{'}) \wedge t_1^{'} \leq t_1 \wedge$ $\forall\, t_2 : Ti \bullet t_2 < t_1 \Rightarrow$ $\neg(Up := F, k, t_2)$	by expansion of A_4 using *ini*
2.	$\exists\, t_1^{'} : Ti \bullet \theta(Up := T, k, t_1^{'}) \wedge t_1^{'} \leq t_1$	by $\wedge$-elim(1)
3.	$t_1^{'} \leq t_1$	by $\wedge$-elim(2)
4.	$t_1^{'} \leq t_1 \leq t$	by arithmetic(3,A_3)
5.	$t_1^{'} \leq t$	by transitivity of $\leq$
6.	$t_4 < t$	Assumption
6.a.	$t_1 \leq t_4 < t$	by cases from 6
6.a.1.	$t_4 < t_2$	by transitivity of $<$, 6.a,A_3
6.a.2.	$t_1 \leq t_4 < t_2$	by arithmetic (6.a,6.a.1)
6.a.3.	$\nexists j : Occ \bullet \theta(lower, j, t_4)$	by m-p(6.a.2, A_5)
6.a.4.	$\nexists k : Occ \bullet \theta(Up := F, k, t_4)$	by m-p(6.a.3, G(22))
6.a.5.	$\forall\, k : Occ \bullet \neg\theta(Up := F, k, t_4)$	by rewriting 6.a.4
6.a.6.	$\neg\theta(Up := F, k, t_4)$	by $\forall$-elim(6.a.5)
6.b.	$t_4 \leq t_1 < t$	by cases from 6
6.b.1.	$\forall\, t_2 : Ti \bullet t_2 < t_1 \Rightarrow \neg\theta(Up := F, k, t_4)$	by $\wedge$-elim(1)
6.b.2.	$t_4 < t_1$	by arithmetic (6.b)
6.b.3.	$\neg\theta(Up := F, k, t_4)$	by m-p(6.b.2,6.b.1)
7.	$\neg\theta(Up := F, k, t_4)$	by 6.a.6,6.b.3
8.	$t_4 < t \Rightarrow \neg\theta(Up := F, k, t_4)$	by $\Rightarrow$-intro(6,7)
9.	$\exists\, t_1^{'} : Ti \bullet \theta(Up := T, k, t_1^{'})$	by $\wedge$-elim(2)
10.	$\exists\, t_1^{'} : Ti \bullet \theta(Up := T, k, t_1^{'}) \wedge t_1^{'} \leq t$	by $\wedge$-intro(9,5)
11.	$\forall\, t_4 < t \Rightarrow \neg\theta(Up := F, k, t_4)$	by $\forall$-intro(8)
12.	*Goal by* $G(1)$, 10, 11, *and* ini *def*	

Case 2. The Gate is down

A_1	$\theta(Some := F, i, t_1)$
A_2	$\theta(Some := T, i, t_2)$
A_3	$t_1 + \epsilon_2 < t < t_2$
A_4	$ini(Down, k, t_1)$
A_5	$\forall\, t_3 : Ti \bullet t_1 \leq t_3 < t_2 \Rightarrow \nexists j : Occ \bullet \theta(lower, j, t_3)$

1.	$i = 1 \vee i > 1$	by def. of Occ
1.*a*.	$i = 1$	case assumption
1.*a*.1.	$\theta(Some := F, 1, 0)$	by S(3)
1.*a*.2.	$\theta(Some := F, 1, t_1)$	by subs of 1.a. into A_1
1.*a*.3.	$\theta(Some := F, 1, 0) \wedge \theta(Some := F, 1, t_1)$	by $\wedge$-intro(1.a.1, 1.a.2)
1.*a*.4.	$t_1 = 0$	by m-p(1.a.3, mono-ax-1)
1.*a*.5.	$\theta(Up := T, 1, t_1)$	by subs of 1.a.4 into G(1)
1.*a*.6.	$t_1 \leq t$	by arithmetic from A_3
1.*a*.7.	$t_4 < t$	by assumption of arbitr. t_4
1.*a*.8.	$t_4 < t_2$	by $<$ transitivity from A_3
1.*a*.9.	$t_1 \leq t_4$	by arithmetic from 1.a.4
1.*a*.10.	$t_1 \leq t_4 < t_2$	by $\wedge$-intro(1.a.8,1.a.9)
1.*a*.11.	$\nexists j : Occ \bullet \theta(lower, j, t_4)$	by m-p(1.a.10, A_5)
1.*a*.12.	$\nexists k : Occ \bullet \theta(Up := F, k, t_4)$	by m-p(1.a.11, G(22))
1.*a*.13.	$\forall k : Occ \bullet \neg\theta(Up := F, k, t_4)$	by rewrit. of 1.a.12
1.*a*.14.	$\neg\theta(Up := F, 1, t_4)$	by $\forall$-elim(1.a.13)
1.*a*.15.	$t_4 < t \Rightarrow \neg\theta(Up := F, 1, t_4)$	by $\Rightarrow$-intro(1.a.7,1.a.14)
1.*a*.16.	$\theta(Up := F, 1, t_1) \wedge t_1 \leq t$	by $\wedge$-intro(1.a.5,1.a.6)
1.*a*.17.	$\exists t_1 : Ti \bullet \theta(Up := F, 1, t_1) \wedge t_1 \leq t$	by $\exists$-intro(1.a.16)
1.*a*.18.	$\forall t_4 : Ti \bullet t_4 < t \Rightarrow \neg\theta(Up := F, 1, t_4)$	by $\forall$-intro(1.a.15)
1.*a*.19.	*Goal by* $G(1)$, $1.a.17$, $1.a.18$, *and* ini *def*	
1.*b*.	$i > 1$	case assumption
1.*b*.1.	$\theta(Some := F, i, t_1) \wedge i > 1$	by $\wedge$-intro(A_1, 1.b)
1.*b*.2.	$\exists j : Occ \bullet \theta(None := T, j, t_1)$	by m-p(1.b.1, S(5))
1.*b*.3.	$\theta(None := T, i, t_1)$	by $\exists$-elim(1.b.2)
1.*b*.4.	$\theta(None := T, i, t_1) \wedge i > 1$	by $\wedge$-intro(1.b.3,1.b)
1.*b*.5.	$\exists j, k : Occ \bullet \theta(raise, j, t_1) \wedge \theta(dec(count), k, t_1)$	by m-p(1.b.4,S(14))
1.*b*.6.	$\exists j : Occ \bullet \theta(raise, j, t_1)$	by $\wedge$-elim(1.b.5)
1.*b*.7.	$ini(Down, k, t_1) \wedge \exists j : Occ \bullet \theta(raise, j, t_1)$	by$\wedge$-intro(A_4,1.b.6)
1.*b*.8.	$\theta(Down := F, k + 1, t_1)$	by m-p(1.b.7, G(18))
1.*b*.9.	$k + 1 > 1$	by def. of Occ
1.*b*.10.	$\theta(Down := F, k + 1, t_1) \wedge k + 1 > 1$	by $\wedge$-intro(1.b.8,1.b.9)
1.*b*.11.	$\exists j : Occ \bullet \theta(MvUp := T, j, t_1)$	by m-p(1.b.10,G(5))
1.*b*.12.	$\theta(MvUp := T, j, t_1)$	by $\exists$-elim(1.b.11)

1.b.13.	$\exists\, t_4 : Ti \bullet \theta(MvUp := F, j+1, t_4) \wedge$ $t_4 \leq t_1 + \epsilon_2$	by m-p(1.b.12, G(21))
1.b.14.	$\theta(MvUp := F, j+1, t_4) \wedge t_4 \leq t_1 + \epsilon_2$	by $\exists$-elim(1.b.13)
1.b.15.	$j + 1 > 1$	by def. of Occ
1.b.16.	$\theta(MvUp := F, j+1, t_4)$	by $\wedge$-elim(1.b.14)
1.b.17.	$\theta(MvUp := F, j+1, t_4) \wedge j + 1 > 1$	by $\wedge$-intro(1.b.16,1.b.15)
1.b.18.a.	$\exists\, j : Occ \bullet \theta(lower, j, t_4) \wedge$ $\exists\, k : Occ \bullet \theta(MvDown := T, k, t_4) \vee$	
1.b.18.b.	$\exists\, t_1 : Ti \bullet \theta(MvUp := T, j, t_1) \wedge$ $t_4 = t_1 + \epsilon_2 \wedge$ $\exists\, j : Occ \bullet \theta(Up := T, j, t_4)$	by m-p(1.b.17,G(9))
1.b.19.a.	$\exists\, j : Occ \bullet \theta(lower, j, t_4) \wedge$ $\exists\, k : Occ \bullet \theta(MvDown := T, k, t_4)$	by case split on 1.b.18.a
1.b.19.a.1.	$\exists\, j : Occ \bullet \theta(lower, j, t_4)$	by $\wedge$-elim(1.b.19.a)
1.b.19.a.2.	$t_4 \leq t_1 + \epsilon_2$	by $\wedge$-elim(1.b.14)
1.b.19.a.3.	$\theta(MvUp := F, 1, 0)$	by G(2)
1.b.19.a.4.	$\forall\, i : Occ, \forall\, t_1 : Ti \bullet \theta(MvUp := F, i+1, t_1)$ $\Rightarrow \exists\, t_2 : Ti \bullet \theta(MvDown := T, i, t_2) \wedge$ $t_2 \leq t_1$	by m-p(1.b.19.a.3,trans-ax-4)
1.b.19.a.5.	$\exists\, t_2 : Ti \bullet \theta(MvUp := T, j, t_2) \wedge t_2 \leq t_4$	by m-p(1.b.16,1.b.19.a.4)
1.b.19.a.6.	$\theta(MvUp := T, j, t_5) \wedge t_5 \leq t_4$	by $\exists$-elim(1.b.19.a.5)
1.b.19.a.7.	$\theta(MvUp := T, j, t_5)$	by $\wedge$-elim(1.b.19.a.6)
1.b.19.a.8.	$\theta(MvUp := T, j, t_1) \wedge \theta(MvUp := T, j, t_5)$	by $\wedge$-intro(1.b.12,1.b.19.a.7)
1.b.19.a.9.	$t_1 = t_5$	by m-p(1.b.19.a.8, mono-ax-1)
1.b.19.a.10.	$t_5 \leq t_4$	by $\wedge$-elim(1.b.19.a.6)
1.b.19.a.11.	$t_1 \leq t_4$	by subst(1.b.19.a.9,1.b.19.a.10)
1.b.19.a.12.	$t_4 \leq t1 + \epsilon_2$	by $\wedge$-elim(1.b.14)
1.b.19.a.13.	$t_4 < t_2$	by arithmetic(1.b.19.a.12,A_3)
1.b.19.a.14.	$t_1 \leq t_4 < t_2$	by $\wedge$-intro(1.b.19.a.11,1.b.19.a.13)
1.b.19.a.15.	$\nexists j : Occ \bullet \theta(lower, j, t_4)$	by m-p(1.b.19.a.14, A_5)
1.b.19.a.16.	*Goal by contradiction of* 1.b.19.a.1 *with* 1.b.19.a.15	
1.b.19.b.1.	$\exists\, j : Occ \bullet \theta(Up := T, j, t_4)$	by $\wedge$-elim(1.b.18.b)
1.b.19.b.2.	$\exists\, t_1 : Ti \bullet \theta(MvUp := T, j, t_1) \wedge$ $t_4 = t_1 + \epsilon_2$	by $\wedge$-elim(1.b.18.b)
1.b.19.b.3.	$\theta(MvUp := T, j, t_1^{'}) \wedge t_4 = t_1^{'} + \epsilon_2$	by $\exists$-elim(1.b.19.b.2)
1.b.19.b.4.	$\theta(MvUp := T, j, t_1^{'})$	by $\wedge$-elim(1.b.19.b.3)
1.b.19.b.5.	$\theta(MvUp := T, j, t_1^{'}) \wedge \theta(MvUp := T, j, t_1)$	by $\wedge$-intro(1.b.19.b.3,1.b.12)
1.b.19.b.6.	$t_1^{'} = t_1$	by m-p(1.b.19.b.5,mono-ax-1)
1.b.19.b.7.	$t_4 = t_1^{'} + \epsilon_2$	by $\wedge$-elim(1.b.19.b.3)
1.b.19.b.8.	$t_4 = t_1 + \epsilon_2$	by subst(1.b.19.b.6 into 1.b.19.b.7)
1.b.19.b.9.	$t_4 < t < t_2$	by subst of 1.b.19.b.8 into A_3
1.b.19.b.10.	$t_4 \leq t$	by arithmetic 1.b.19.b.8)
1.b.19.b.11.	$t_5 < t$	Assumption of arbitr t_5
1.b.19.b.12.	*follow steps* $6-12$ *of Case* 1 *using* t_5 *for* t_4	

Case 3. The Gate is moving up

A_1 $\quad \theta(Some := F, i, t_1)$
A_2 $\quad \theta(Some := T, i, t_2)$
A_3 $\quad t_1 + \epsilon_2 < t < t_2$
A_4 $\quad ini(MvUp, k, t_1)$
A_5 $\quad \forall\, t_3 : Ti \bullet t_1 \leq t_3 < t_2 \Rightarrow \neg\, \exists\, j : Occ \bullet \theta(lower, j, t_3)$

1. $\exists\, t_4 : Ti \bullet \theta(MvUp := T, k, t_4) \wedge t_4 \leq t_1 \wedge$
   $\forall\, t_2 : Ti \bullet t_2 < t_1 \Rightarrow \neg\theta(MvUp := F, k+1, t_2)$ by exp of A_4 using *ini* def
2. $\theta(MvUp := T, k, t_4) \wedge t_4 \leq t_1$ by ∃-elim(1)
3. $\theta(MvUp := T, k, t_4)$ by ∧-elim(2)
4. $\exists\, t_1 : Ti \bullet \theta(MvUp := F, k+1, t_1) \wedge t_1 \leq t_4 + \epsilon_2$ by m-p(3, G(21))
5. $\theta(MvUp := F, k+1, t_5) \wedge t_5 \leq t_4 + \epsilon_2$ by ∃-elim(4)
6. *follow steps* 1.*b*.15 *to* 1.*b*.19.*b*.11 *of Case* 2. *using* 3 *in place of* 1.*b*.12, *and* 5, *in place of* 1.*b*.14

Case 4. The Gate is moving down
Trivial; proof ommitted.

Affiliation of authors

Jim Armstrong
Dept. of Computer Science
Dependable Computer Systems Centre
University of York
York Y01 5DD
U.K.
jma@minster.york.ac.uk

Leonor Barroca
Dept. of Computing
The Open University
Milton Keynes MK7 6AA
U.K.
L.Barroca@open.ac.uk

Mail address for correspondence

Leonor Barroca
Dept. of Computing
The Open University
Milton Keynes MK7 6AA
U.K.
L.Barroca@open.ac.uk
tel:44 908 65 4864
fax:44 908 65 2140