Staffordshire University

School of Computing

SOCTR/97/01

On Making a Crossing Safe

Eur.Ing. Dr. Albert Alderson
School of Computing, Staffordshire University,
PO Box 334, Beaconside, Stafford, ST18 0DG
Email: cmtaa@soc.staffs.ac.uk

Dr. Judith Barnard
Dept. of Computer Science, University of Warwick

21st January 1997

Abstract

Barnard et al (1995) describe the design of a level crossing system using Real-time Communicating X-machines (CXMs). The authors remark that a CXM model is a formal model which is amenable to proof, but the verification techniques are not covered in their paper. This paper introduces then applies those techniques to the published design to locate a safety problem and goes on to validate further designs. This paper also takes into account the observation of Barnard (1996) and others, that the mathematical notation used to model CXMs has an off-putting effect on most system developers. Here, more familiar programming notations are used wherever possible to increase accessibility.

1. Introduction

Barnard et al (1995) describe a formal specification technique for modelling real-time systems called Real-time Communicating X-machines (CXMs). They develop a railway level-crossing case study using CXMs, but do not apply the associated verification techniques (Barnard 1996). This paper goes on to apply those techniques to their design, and to find an unsafe state. Further designs are developed to address the safety issue and to address the full specification of the level-crossing system. Attention then switches to verifying the real-time aspects of CXMs.

The real-time communicating X-machine (CXM) is an extension of the communicating X-machine model which, in turn, is an extension of the stream X-machine model. The stream X-machine (Laycock, 1993) is a typed finite state machine for modelling sequential systems, (Holcombe, 1988, Barnard, 1993). A CXM models a process as a timed typed finite state machine. Each CXM is able to communicate with other CXMs through its output windows and input ports. Communication between two CXMs is established through the use of a named channel which connects a window of the sending CXM to a port of the receiving CXM. A signal or an item of data may be transmitted through a channel.

The modelling of an individual process as a CXM focuses upon the behaviour of the process in terms of states and transitions. At any one time each process carries one instance of its data type X (the 'X' of X-machines) which consists of the CXM's memory and data at its ports and windows. Each transition has an associated condition based on the values of the CXM's memory and the data that may be at one or more of the CXM's ports. This condition determines whether or not a transition takes place. When a transition occurs, inputs from ports which are used to evaluate that condition are removed.

In general any input queued at any port can be involved in a condition. However, we restrict ourselves to the case in which only the input at the head of one input queue may be involved. Where inputs from more than one queue could be involved, we can treat each input separately and use either state or memory to record that an input has occurred. Where any input other than at the head of an input queue would have been involved, we can restore our restriction by interposing a new CXM which reorders the queue so that the required input is at the head of the queue.

2. The Level-Crossing Case Study

The level-crossing system involves a two-track railway (an *up* track and a *down* track) having a level crossing with gates that are lowered and raised, and flashing amber and steady red lights. Along the tracks are treadles that monitor the position of the trains. This is shown in Figure 1.

The signals from the treadles, one set on each track, are used as follows :

〈 Treadle 1 If the gates are down, keep them down,
〈 Treadle 2 Lower the gates (if not already down) within **n** seconds,
〈 Treadle 3 Raise gates, unless treadle 1 on the other track has been triggered,

where 'lower the gates' consists of flashing the amber lights for **x** seconds, then setting the red lights and lowering the gates.

We define the state of a train relative to the treadles as follows:

〈 Travelling Train is outside the area covered by the treadles
〈 Approaching Train is between treadle 1 and treadle 2.
〈 Crossing Train is between treadle 2 and treadle 3.

We denote the signals from the treadles of the up and down tracks, by the shorthand:

〈 1U, 2U, 3U for the up track,
〈 1D, 2D, 3D for the down track.

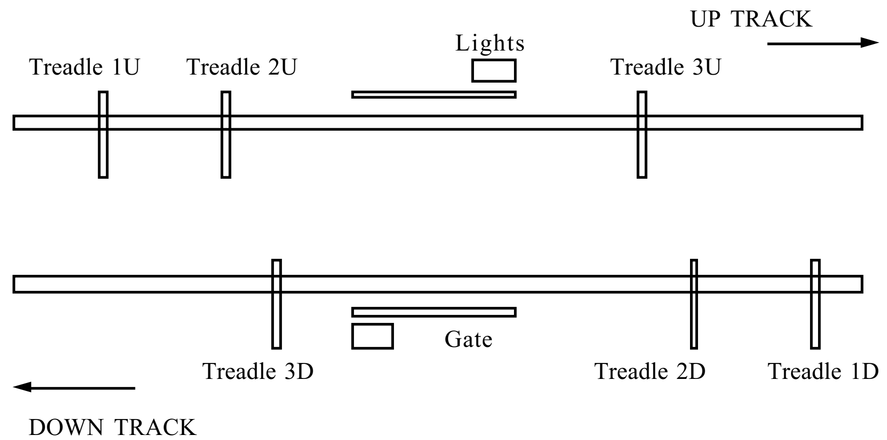


Figure 1

The external behavioural model (or communication diagram) of the solution in Barnard et al. (1995) is given in Figure 2. We will call this the “DARTS design” after the conference at which it was presented.

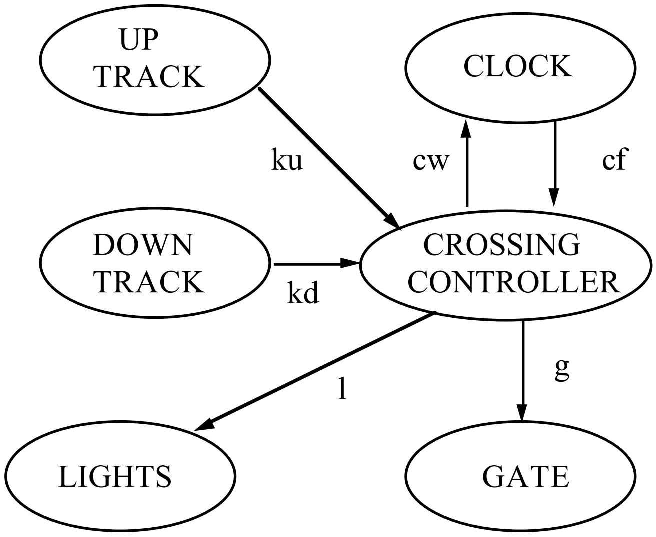


Figure 2

In this notation for external behaviour models, the objects are shown as labelled ellipses. Channels are shown as labelled directed lines from the window of one object to the port of another. In subsequent discussion and in state transition diagrams windows are identified by “channel_name!” and ports by “channel_name?”.

The state transition diagram of the crossing controller in the DARTS design is given in Figure 3.

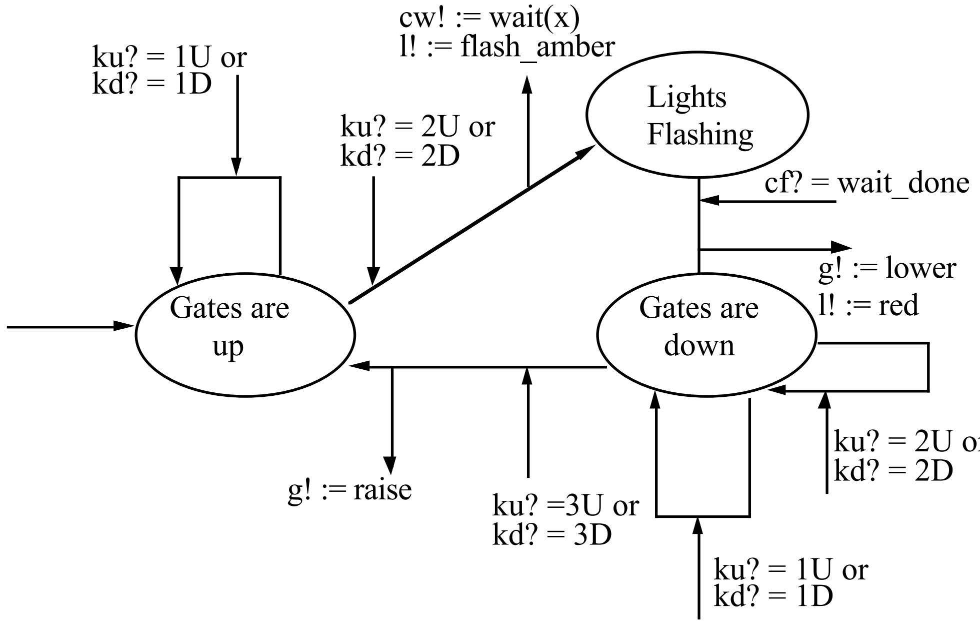


Figure 3

In this notation for state transition diagrams, states are represented by labelled ellipses and transitions by directed lines between states. A directed line to a transition denotes a triggering condition. A directed line from a transition denotes the action associated with the transition. The symbol "=" denotes equality. The symbol ":=" denotes assignment.

It is important that the gates are never up when a train is near or on the crossing. That is, the gate must never be up when a train is between treadles 2 and 3.

To clarify the application of the verification technique, it is helpful initially to omit the timing element of the specification involved with switching the traffic lights to amber and red. (Although, we note here that the solution with which we are presented in Figure 3 provides no means of switching the lights off again.) The essence of the DARTS design then is represented in the state transition diagram of the level-crossing controller module, shown in Figure 4.

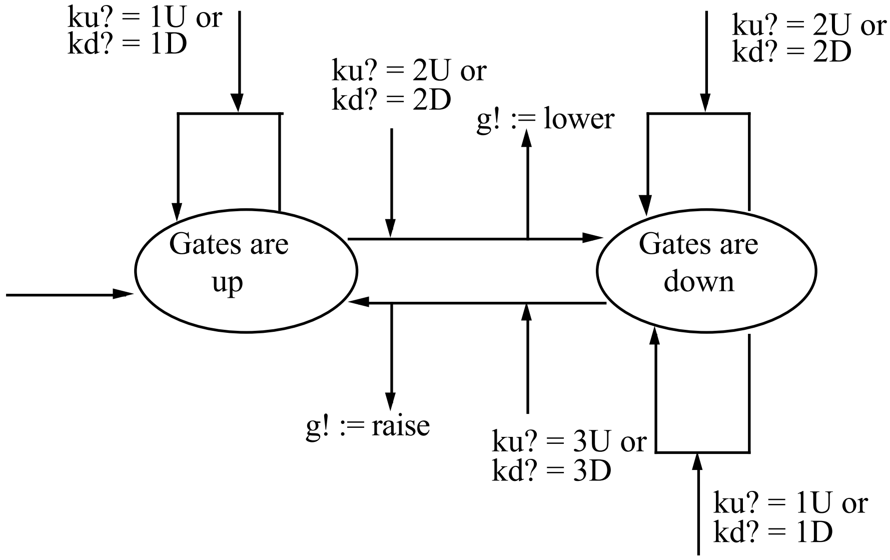


Figure 4

The random choice of the pattern of inputs 1U, 1D, 2U, 2D, 3U, 3D as a test case led to two observations:

1. that the gates would be raised between receipt of the inputs 3U and 3D, causing an unsafe state;
2. on re-checking this result, that the state transition diagram presented did not actually cope with the pattern of inputs supplied, since the input 3D occurs when the system is in state **Gates are up**, which has no transition to cope with input 3D or 3U.

3. The Verification

In order to proceed, extra transitions were added to give the state transition diagram in Figure 5. These are the transitions that had been inattentively assumed in the initial casual test with the 1U, 1D, 2U, 2D, 3U, 3D input pattern.

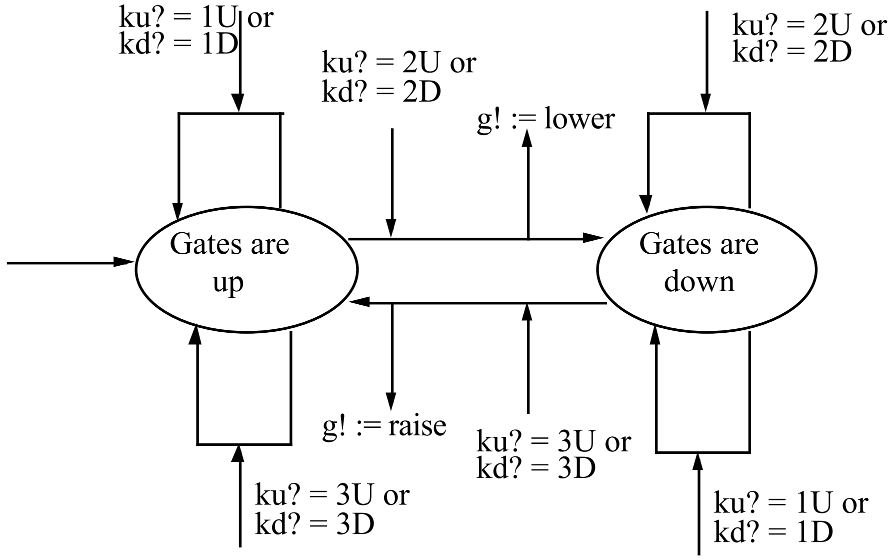


Figure 5

CXM verification (Barnard 1996) is based on reachability analysis, as in Petri Nets. Through the construction of a reachability tree, a design can be verified against a wide variety of properties such as correctness of design, freedom from deadlock and reachability. The reachability tree considers the global state of a system. This has the advantage of effectively reducing the number of states to be analysed, which is a particularly effective in controlling the explosion of states common in state based techniques.

Correctness of design is addressed by formalising the requirements from the requirement specification through the use of global state. Each requirement is represented as a global state or a transition from one global state to another. Once the reachability tree has been constructed then these 'global state requirements' can be identified on the tree to show the design has catered for them.

However, knowing the state of a object at a particular time may not be sufficient for a thorough description of that object. Memory values of the object may also need to be considered. Therefore, requirements of the system are formalised in terms of global states and any memory values that are of particular importance.

The reachability tree is constructed beginning with the initial global state and recursively constructing successor states by considering the transitions that can occur in each global state. The tree displays all the possible global states and the paths that the system can take. In its construction, predicates on transitions are assumed to be true and so the transition takes place. A more detailed description of this reachability tree construction can be found in Barnard (1996).

The tree can be used to determine various safety and liveness properties, such as a deadlocking state in which a global state at a leaf node on the tree does not contain all terminal states or is not a duplicate node. This suggests that the system will 'lock' in this global state with no way out.

The global state of the DARTS design can be described by a composite of the state of the trains and the gates as {up-train, down-train, gates} where:
〈 train states = {travelling, approaching, crossing} in shorthand {T, A, C},
〈 gate states = {up, down} in shorthand {U, D}.
The initial state is {T, T, U} and unsafe states are {C, *, U} and {*, C, U} where '*' denotes any value. The values of the global state reached by the transitions are shown in Figure 6.

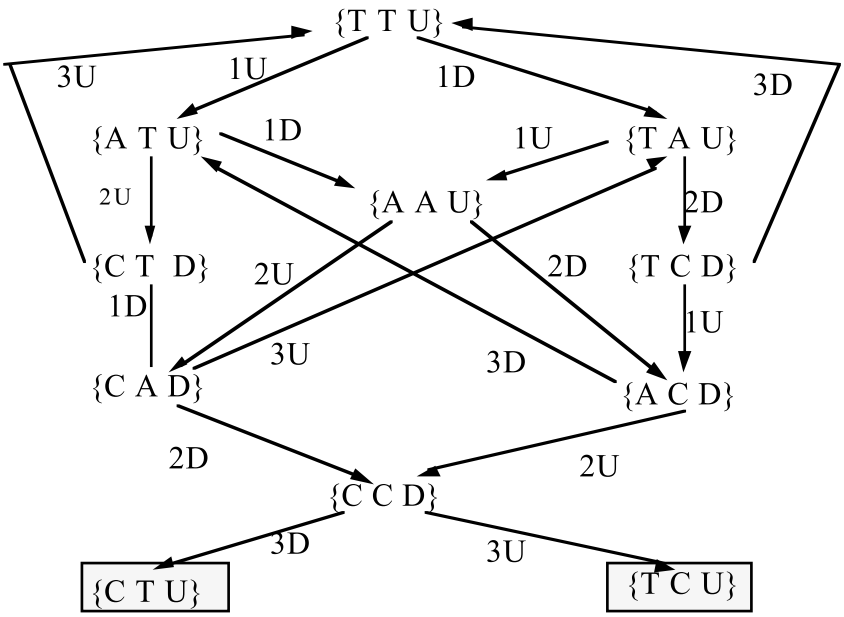


Figure 6

We see that where two trains are in the *crossing* state, the gates are down, {C C D}. However, when the first train passes its third treadle, the gates are opened with the other train still in the *crossing* state, {C T U} or {T C U}. The design is unsafe.

[One thought that occurred was that if in Figure 5, the condition of the transition from down to up was 3D? and 3U?, then the gates would not be lowered until the second train had passed. Perhaps, the malfunction depended only upon the use of or where and had been intended. Considering this we see that there will be no transition until both 3U and 3D have been received. This could cause inputs to be queued for one of the tracks. This is not acceptable. Suppose we have the sequence 1U, 1D, 2U, 2D, 3U, 1U, 2U, 3D. The sequence 3U, 1U, 2U would be queued until 3D was received, then 3U and 3D would be consumed causing the gate to be raised even though a train is between treadles 2 and 3 in reality (although not in the system's representation of reality). 1U and 2U would then be dealt with, probably sending a lower gate demand before the gate is completely raised.]

4. A Safe Design

A systems engineering response to the unsafe situation is to treat it as a locking problem. The gate needs to be locked down when either train is crossing. The gate may only be raised when it is not locked by either train. A standard implementation is to count the number of trains in the *crossing* state and raise the gates only when this reduces to zero. This requires a simple modification to the design given in Figure 5. A counter, *count*, is introduced, which is initially zero; it is incremented when a train passes its second treadle; it is decremented when a train passes its third treadle. The design is represented in the state transition diagram of the level-crossing controller module, shown in Figure 7.

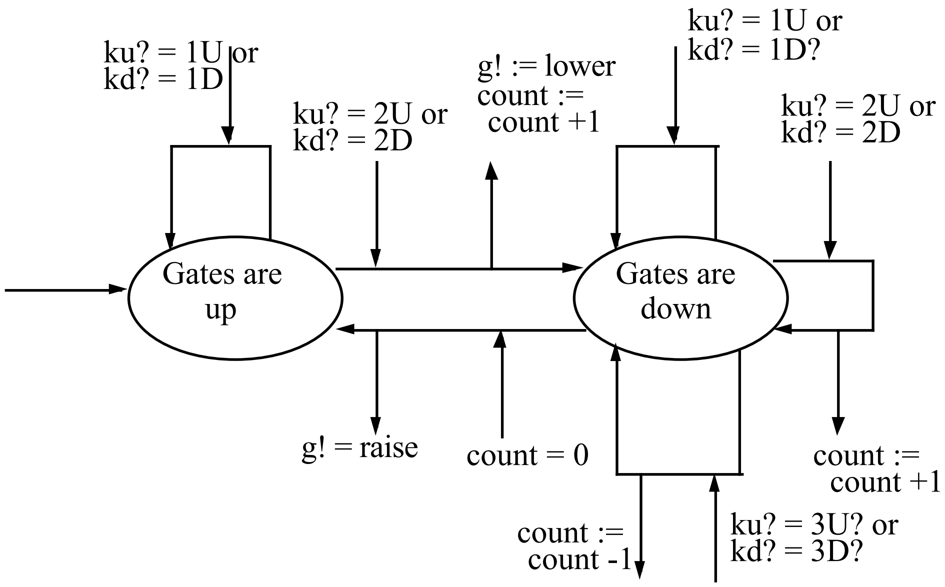


Figure 7

For the verification the global state must be extended to {up-train, down-train, gates, counter} where the value of the counter is one of {0, 1, 2}. The initial state is {T, T, U, 0} and unsafe states are {C, *, U, *} and {*, C, U, *} where '*' denotes any value. The reachability tree of the transitions is shown in Figure 8.

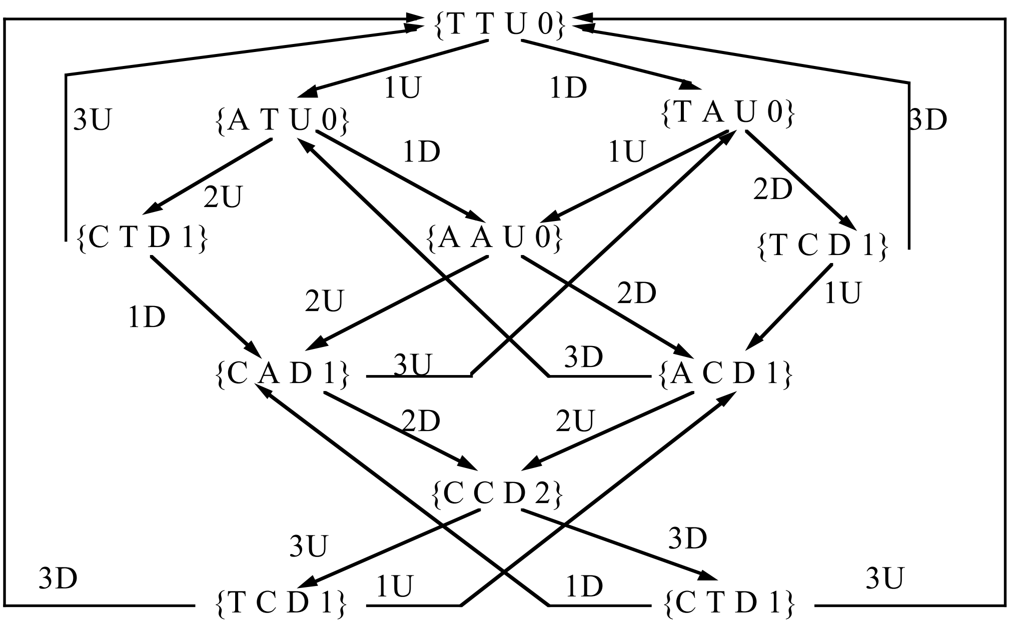


Figure 8

We see that this resolves the safety problem; the unsafe states are never reached.

However, returning to the original specification, we note the optimising requirement:
〈 Treadle 1 If the gates are down, keep them down (in the case of a second train coming).
We note that neither of the proposed designs meets this requirement.

5. A Design for the Optimising Requirement

It might seem that a simple amendment is possible to the previous design to address this requirement. A U1 or D1 signal must be allowed to increment the counter, if the gate is already down for the other train (that is, if the counter is already 1). However, a U2 or D2 signal must only increment the counter if the corresponding U1 or D1 did not. This implies that a flag must be held individually for each train, in which case the counter can be dispensed with.

Therefore the proposed design introduces a flag for each train (designated as DF and UF) to indicate whether the gates are being held down for it. When a train passes treadle 1 it **requests** the gates to be held down for it. They will be held down if they are already being held down for the other train. The flag is set to Yes if the gate is being held down, otherwise it is No. When a train passes treadle 2, it unconditionally **demands** the gates be held down for it. When a train passes treadle 3, it **requests** that the gates be raised; they will be raised only if not held down for the other train. We are led to a design, shown in the external behaviour diagram of Figure 9. The state transition diagrams of the objects are shown in Figure 10, Figure 11, Figure 12 and Figure 13.

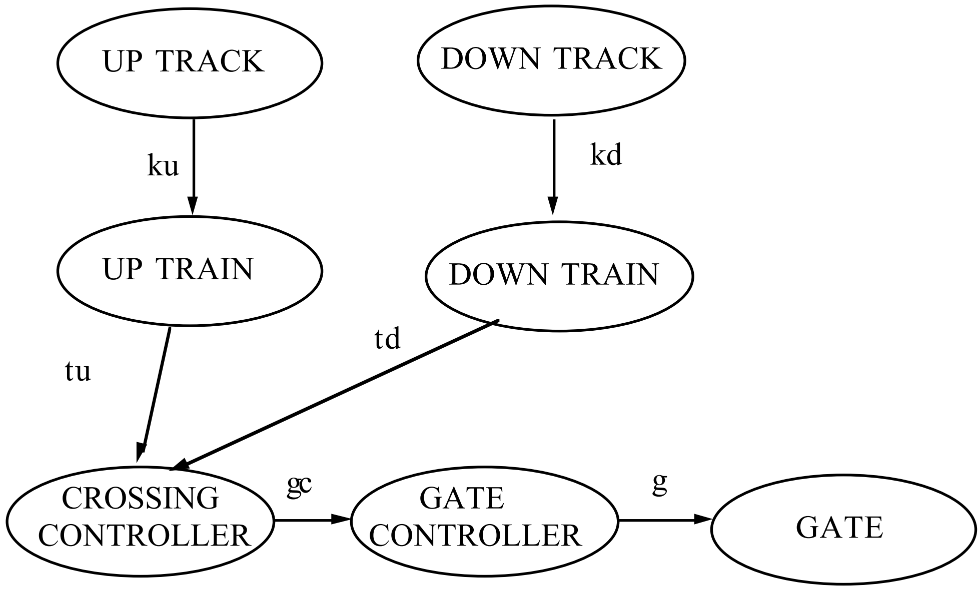


Figure 9

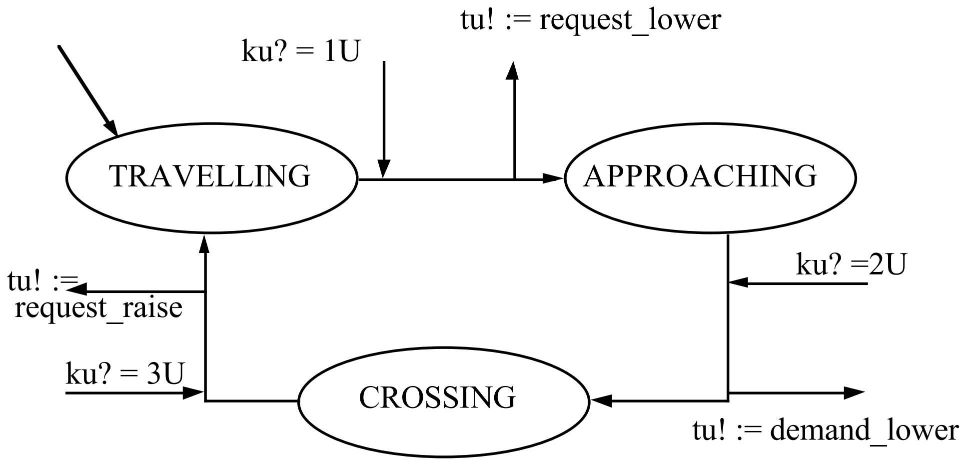
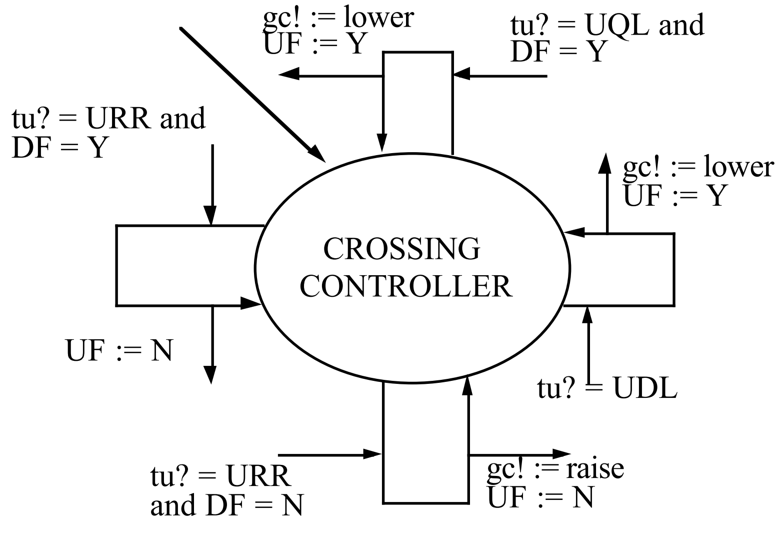


Figure 10



plus equivalent transitions for Down Train

Figure 11

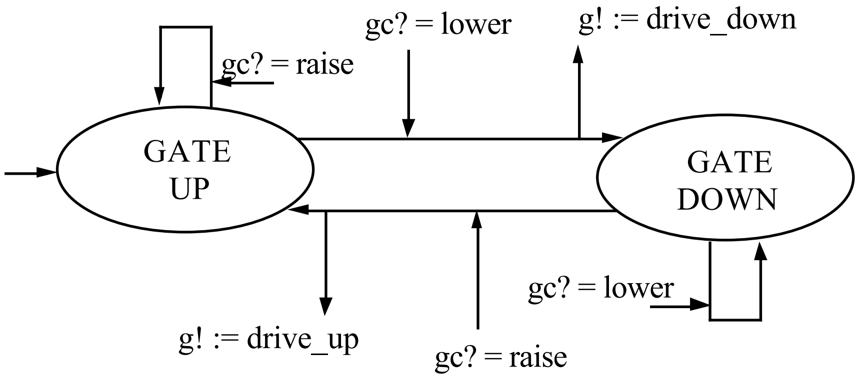


Figure 12

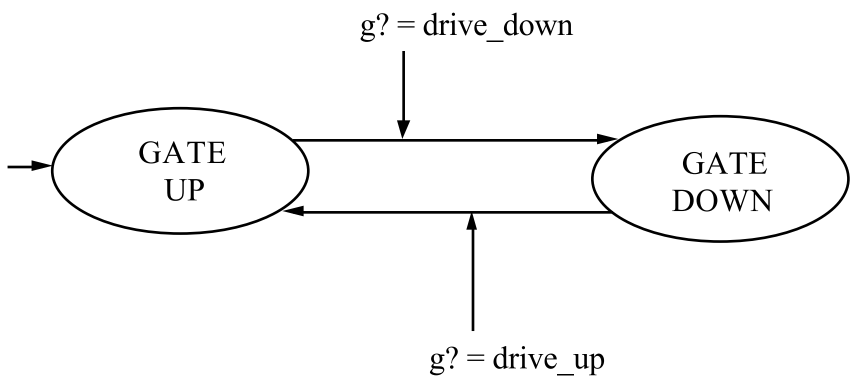


Figure 13

The global state for verification requires the state of the trains, of the gate and of the flags represented by {up-train, up-flag, down-train, down-flag, gates}. Y and N represent the states of the flags - Yes and No. The initial state is {T, N , T, N, U}, the unsafe states are {C, *, *, *, U} and {*, *, C, *, U}. The optimising states are {C, Y, A, Y, D}, {A, Y, C, Y, D} and {A, Y, A, Y, D}. The values of the global state reached by the transitions are shown in Figure 14.

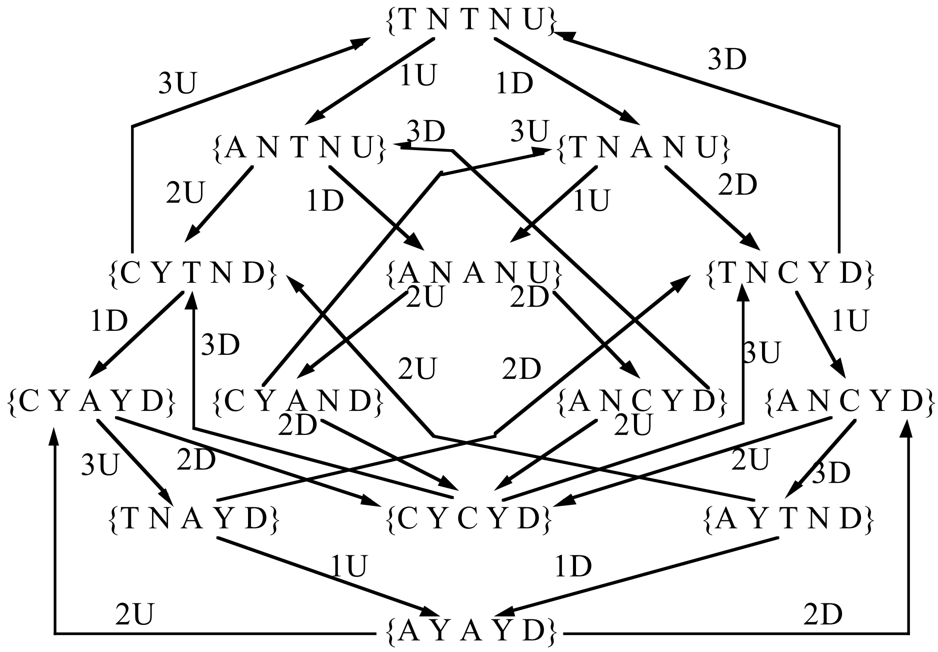


Figure 14

We see that this is a safe design, since none of the unsafe states is reached, and that it meets the optimising requirement, shown by the reachability of the states {T N A Y D}, {A Y T N D} and {A Y A Y D}.

6. Addressing the Timing Requirements

We now return to the timing requirements and address the specification for the lights. The external behaviour diagram is shown in Figure 15.

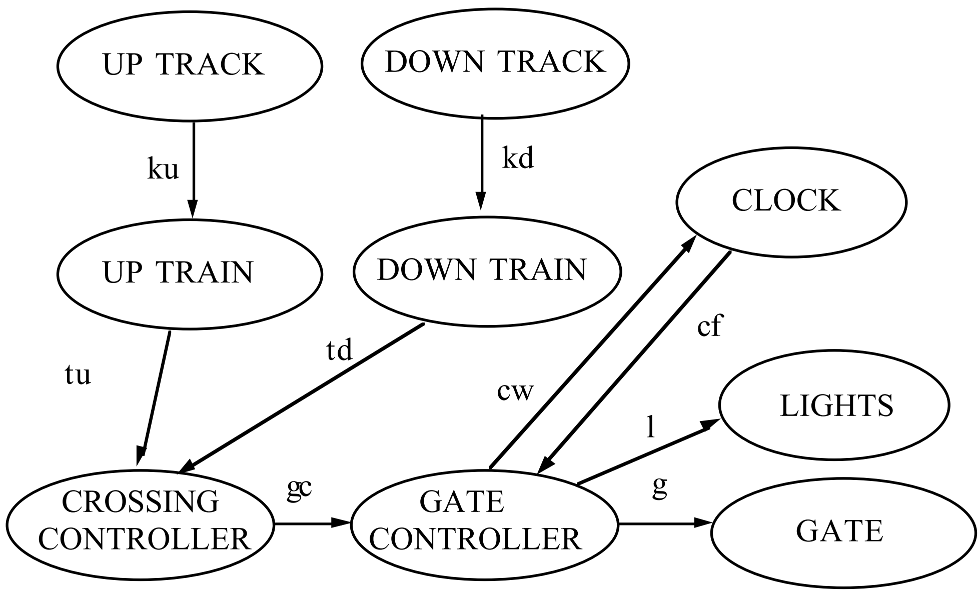


Figure 15

The state transition diagrams for the trains, Figure 10, and the crossing controller, Figure 11, remain unchanged. In order to create the state transition diagram for the gate controller, we must specify what happens when a contra-command is received while the gates are in the process of being raised or lowered. We add the specification:

〈 The gate only accepts inputs when the gate is up or the gate is down, and not at any other time.

〈 The gate gives a signal when it has finished its movement up or down.

The state transition diagram for the gate controller is given in Figure 16, extending Figure 12 by the addition of a state when the amber lights are flashing to warn that the gates are lowering, a state during which the gates are lowering, and a state during which the gates are raising. Commands are made to the lights to switch them to the necessary states. Note that a command to switch the lights off has been included. The state transition diagram of the gate is given in Figure 17 extending Figure 13 by the signals when movement has finished. The state transition diagram of the lights is given in Figure 18. The external behaviour diagram and state transition diagrams of the clock are given in Figure 19.

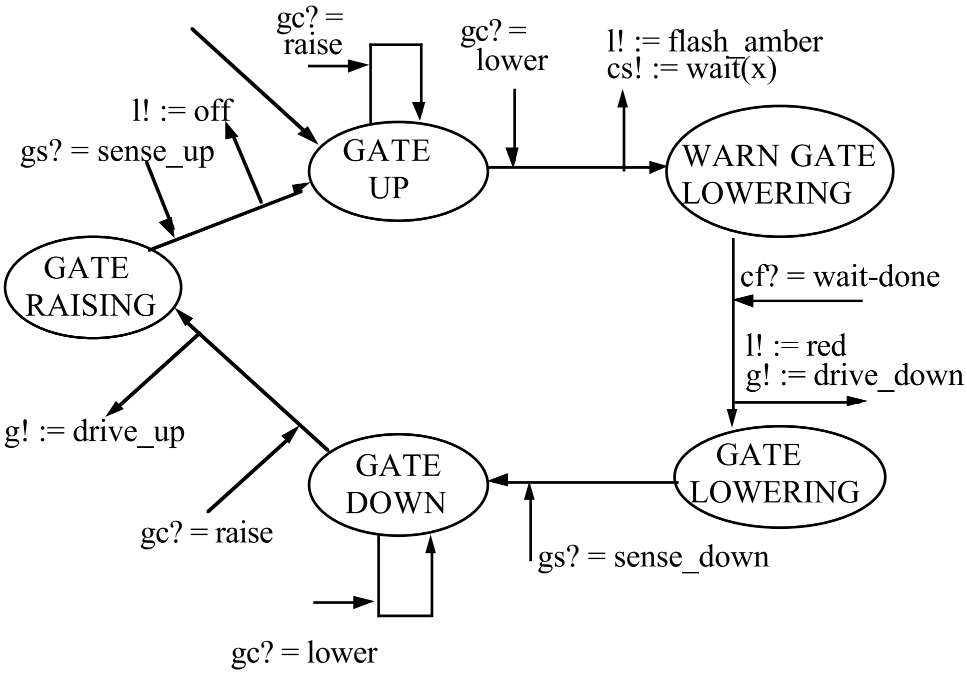


Figure 16

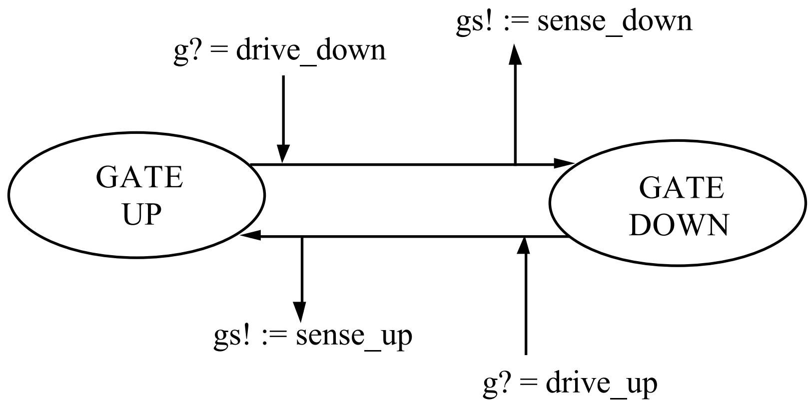


Figure 17

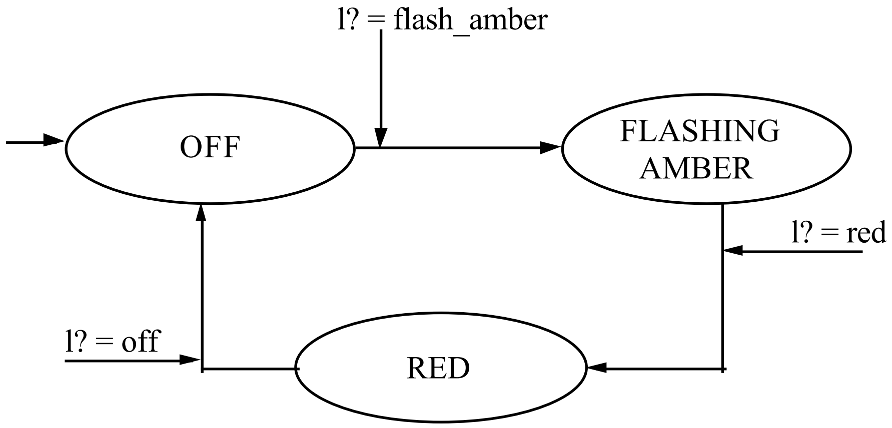


Figure 18

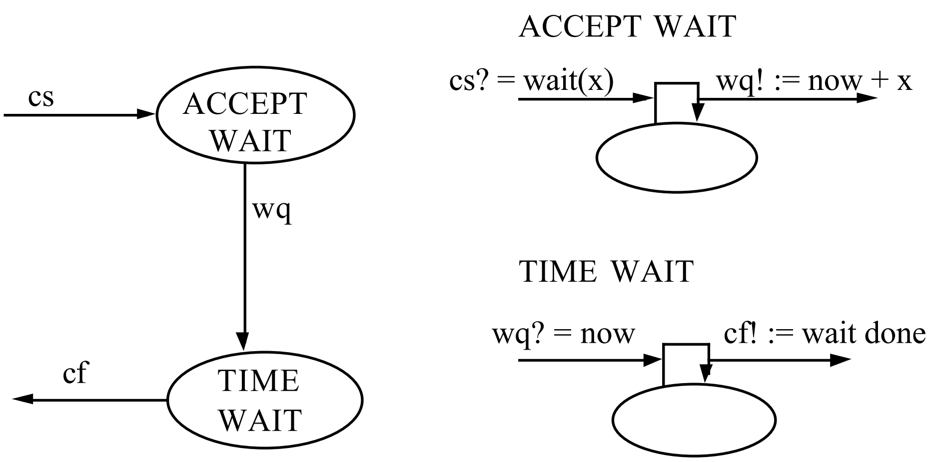


Figure 19

Note is that the validation can be partitioned reducing complexity. Since neither the gate nor the lights create inputs to the trains or the crossing controller, the validation of those items is independent of the gates and lights and unchanged from the design in the previous section. The validation of the crossing controller depends on input from the trains and the clock. This leaves the validation of the gate controller, the gates and the lights. The verification of the lights depends only upon inputs from the gate controller. The verification of the gate controller depends on input from the crossing controller and the gates. The verification of the gates depends on input from the gate controller.

The sequence of commands output by the crossing controller to the gate controller can be seen to be a repeating sequence of lower and raise commands and we can see by inspection that that the state transition diagram of Figure 16 deals correctly with these inputs, in terms of raising and lowering the gate appropriately.

The sequence of commands output by the gate controller to the lights can be seen to be a repeating sequence of flash amber, red, off and we can see by inspection that the state transition diagram of Figure 18 deals correctly with these inputs.

We must now turn to the timing considerations. The requirement is that all lower commands cause the gate to lower within n seconds or they the find the gate down and it remains down. A lower command can occur:

〈 while the gate is in the state Gate Down. Correctly the gate remains down.

〈 while the gate is in the state Gate Raising. It must wait for the gate to finish rising. The command will then be dealt with as if it had occurred in the state Gate Up.

〈 while the gate is in the state Gate Up. It must wait for the lights to flash amber for **x** seconds and then for the gate to finish lowering.

〈 while the gate is in the state Warn Gate Lowering. It must wait for the lights to cease flashing amber, which is a maximum of **x** seconds and for the gate to finish lowering. It is then dealt with as if it had been received in the state Gate Down.

〈 while the gate is in the state Gate Lowering. It must wait for the gate to finish lowering. It is then dealt with as if it had been received in the state Gate Down.

Assuming negligible time for executing the logic within a state, we see that the longest wait for completion of the action of a Lower command is when it occurs in the state Gate Raising. Therefore the following inequality must hold:

〈 n > maximum gate raising time + x + maximum gate lowering time.

The time taken for the train to reach the level crossing having passed treadle 2, depends upon the distance to the crossing and the speed of the train. Therefore, the timing constraint can only be fully dealt with in the context of the environment, where the speed of the trains through the section can be controlled. It is outside the control of the level-crossing system.

7. Summary

This paper utilises the CXM verification technique to isolate an unsafe state in a proposed design for a level-crossing controller. Its use to demonstrate the safety of an amended design and then to illustrate its potential for verifying the design against the requirements specification. These studies show the CXM method and its verification technique to be a viable means of creating designs which can be verified formally against requirements.

References

Barnard J., 1993, X-machine specification, refinement and verification techniques, M.Sc. dissertation, Dept. of Computer Science, University of Sheffield.

Barnard J., 1996, COMX: A methodology for the formal design of computer systems using Communicating X-machines, Ph.D. Thesis, Staffordshire University.

Barnard J., Theaker C.J., Whitworth J. and Woodward M., 1995, Real-time Communicating X-machines *In:* Proceedings of DARTS'95, University Libre de Brussels, Belgium, 9-11th November 1995.

Holcombe M., 1988, X-machines as a basis for dynamic system specification, Software Engineering Journal, March 1988.

Laycock G.T., 1993, Introduction to X-machines, Research Report CS-93-13, Department of Computer Science, Sheffield University.