A CCS Case Study: A Safety-Critical System

Technical Report No. 103

Jean Baillie

June 1990

A CCS Case Study: a Safety-Critical System

Jean Baillie

June 1990

Abstract

A level crossing control system is specified in CCS, motivated by a temporal logic specification of the safety requirements. We show that with certain reservations, these can be satisfactorily stated entirely within CCS. The crossing system is divided into two smaller subsystems which are shown to be equivalent to the original single system, and whose behaviour is then analysed using the methods of the calculus. By applying the technique of bisimulation, it is proved that the crossing satisfies the full safety requirements.

1 Introduction

This paper is an attempt to specify a safety-critical system, namely a level crossing, in the formal language CCS. We show that the specification, including safety requirements, may be captured entirely within that language. The informal description of the level crossing is taken from [Gor87]. In that paper the system requirements, which are mainly safety-related, are specified in temporal logic; here, we show that whilst these requirements may seem on first consideration to be temporal, the sequential and synchronized features of CCS enable us to state them satisfactorily within the language itself.

We begin with an informal description of the level crossing system as set out in [Gor87]. In section 2 we give the main safety requirements, also taken from [Gor87] and expressed in the language of temporal logic; we then show how these may be translated into CCS. Section 3 consists of the CCS specification, including a modified version in which the single system is divided into two communicating subsystems, this being more manageable than the whole. In section 4 we show that the two versions are congruent and having proved this, we then examine the behaviour of the composed subsystems in section 5. Lastly, we show that the system as specified satisfies the safety requirements. The advantages and limitations of the exercise are assessed in the conclusion.

1.1 Informal description of the system

Fig.1 shows the layout of the crossing; the names and arrangement of the physical objects are taken from [Gor87]. The two sensors on the approach side of the line are located after the appropriate lights (in the direction of motion). The required behaviour of the train driver is that he stops the train on a red light and proceeds when the light is green; in proceeding, the train necessarily crosses the sensors in the line. This behaviour is explicitly specified in the agent *ATrain*. Cars making legitimate use of the crossing are sensed by *RSensor* and are counted in and out. Required behaviour of drivers is that they stop their vehicles when the light is red and do not proceed until the light changes to green. This is not specified explicitly. Drivers attempting to use the crossing after the light has changed to red are taking calculated risks and should be aware that no account of this is taken in specifying the safety features of the system.

2 Temporal logic safety requirements

2.1 Global model

The top level safety requirement $S1$, or 'global model' [Gor87], is that there should never be a train *and* a car inside the crossing at the same time. This may be expressed in temporal logic as follows;

$$(\forall t \in TRAINS)(\forall c \in CARS)(\Box \sim (in(t) \wedge inc(c)) \quad (S1)$$

'$\Box$' may be read as 'henceforward' (or the more usual modal interpretation, 'it is necessary that'), '$\sim$' means 'not', $in(t)$ refers to a train inside the crossing and similarly $inc(c)$ for cars.

This main safety requirement is achieved by the following lower level constraints on the application domain. An informal description is given first in each case.

2.2 Formalization of the application domain

In the temporal logic statements below, *red* and *green* refer to rail lights and *rred* and *rgreen* to road lights.
(1) For every train t, if t is outside the crossing and the railway light is red, then t remains outside unless ($\sqcup$) it 'sees' the green light.

$$(\forall t \in TRAINS)((red \wedge out(t) \rightarrow out(t) \sqcup (green \wedge out(t))))$$

(2) After the road light has been switched to red, cars in the crossing will be allowed to leave before ($\lhd$) the barrier is lowered.

$$(\forall c \in CARS)(\Box(rred \rightarrow (outc(c) \lhd down) \vee outc(c)))$$

(3) If the crossing is open for cars then the rail light must be red and there must be no train in the crossing.

$$(\forall t \in TRAINS)(\Box((rgreen \vee up) \rightarrow red \wedge out(t)))$$

(4) If the crossing is open for trains then the road is blocked (gate down) before a train enters the crossing.

$$(\forall t \in TRAINS)(\Box(green \rightarrow down \lhd in(t)))$$

(5) If the gates are closed then the road lights must be red.

$$\Box(down \rightarrow rred)$$

2.3 The CCS version

The global model, $S1$, may be translated directly into CCS:

$$\begin{array}{lcll}
SafeCrossing & \stackrel{def}{=} & \overline{train_{in}}.\overline{train_{out}}.SafeCrossing & (S1) \\
 & & + & \\
 & & \overline{car_{in}}.SafeCrossing_{cars}(1) & \\
SafeCrossing_{cars}(n) & \stackrel{def}{=} & \overline{car_{in}}.SafeCrossing_{cars}(n+1) & \\
 & & + & \\
 & & \overline{car_{out}}.\text{ if } n = 1 \text{ then } SafeCrossing & \\
 & & \quad \text{else } SafeCrossing_{cars}(n-1) &
\end{array}$$

SafeCrossing is regarded as a shared resource; it may be used either by cars or by trains but not by both together (since *car* and *train* signals cannot be interleaved). It does not deadlock; there is no state in which both cars and trains are prevented from using the crossing. Cars are counted in and out of the crossing and only when it is clear of cars does the system allow trains to enter.

3 The CCS specification

3.1 The components of the single system

3.1.1 Sensors and track lights

The sensors, TA(pproach), TI(n) and TO(ut), are triggered by the train (t_a, t_i, t_o) and send signals to the control (a, i, o) to indicate the train's position. The track lights are parameterized by colour; the A(pproach)Light is initially green and the I(n)Light initially red. These change on receiving the signal from Control. A function 'change' is assumed with domain $\{red, green\}$, where $change(red) = green$ and $change(green) = red$.

$$\mathbf{TA} \stackrel{def}{=} t_a.\overline{a}.\mathbf{TA}$$

$$\mathbf{TI} \overset{def}{=} t_i.\bar{i}.\mathbf{TI}$$

$$\mathbf{TO} \overset{def}{=} t_o.\bar{o}.\mathbf{TO}$$

$$\mathbf{ALight}(x) \overset{def}{=} \overline{send_a}(x).change_a.\mathbf{ALight}(change(x))$$

$$\mathbf{ILight}(y) \overset{def}{=} \overline{send_i}(y).change_i.\mathbf{ILight}(change(y))$$

3.1.2 The train

Condition (1) of 2.2 is made explicit in the specification of the train. An approaching train sees the ALight and stops if it is red. It 'polls' this light until the signal changes to green, when it can proceed and cross the TA sensor. Its behaviour is similar at the ILight. When this is green and the train has crossed the TI sensor it sends an observable signal $train_{in}$ to the environment. When the train leaves the crossing it sends an observable signal $train_{out}$ just before crossing TO.

$$\mathbf{ATrain} \overset{def}{=} send_a(red).\mathbf{ATrain} + send_a(green).\overline{t_a}.\mathbf{ITrain}^1$$

$$\mathbf{ITrain} \overset{def}{=} send_i(red).\mathbf{ITrain} + send_i(green).\overline{t_i}.\mathbf{CTrain}$$

$$\mathbf{CTrain} \overset{def}{=} \overline{train_{in}}.\overline{train_{out}}.\overline{t_o}.\mathbf{ATrain}$$

3.1.3 Road sensor

The Road sensor works in two ways. Firstly, it responds to the road light; when this is red, the sensor simply waits for it to turn to green. On the green signal, waiting cars are admitted to the crossing and are counted in and out by *Cars*. If there are no cars waiting the sensor will just wait for the lights to change back to red, or for a car to arrive, whichever is first. Each car is observed entering and leaving the crossing through car_{in} and car_{out} signals. Only when all cars have left, that is, when the number of *out* signals is equal to the number *in*, does the sensor check the lights again, so satisfying 2.2 (2). The lights might have changed to red in the meantime, modeling the situation in which cars may choose to ignore red lights and hence prevent the train from using the crossing. The sequential nature of the control makes it safe for them to do this.

It is important to note here, however, that while the sensor is reading the red light it is not sensing cars on the crossing. Having checked once that the crossing is clear it sends the acknowledgment *sent* to *Control* which then begins to close the gate. If at this point a car enters the crossing it will not be sensed and an accident may be caused; responsible behaviour of drivers is assumed (see

[1]Strictly, $send_a$ should be parameterized and followed by an 'if then else' statement

caveat in 1.1). A car using the crossing legitimately and breaking down before it has left will be sensed and will therefore be safe.

$$\begin{aligned}
\mathbf{RSensor} &\stackrel{def}{=} \mathit{send}_r(\mathit{red}).\overline{\mathit{sent}}.\mathbf{Stop} \\
&\quad + \\
&\quad \mathit{send}_r(\mathit{green}).\mathbf{Go} \\[1em]
\mathbf{Stop} &\stackrel{def}{=} \mathit{send}_r(\mathit{green}).\overline{\mathit{sent}}.\mathbf{RSensor} \\[1em]
\mathbf{Go} &\stackrel{def}{=} \overline{\mathit{car}_{in}}.\mathbf{Cars}(1) + \mathit{send}_r(\mathit{red}).\overline{\mathit{sent}}.\mathbf{Stop} \\[1em]
\mathbf{Cars}(n) &\stackrel{def}{=} \overline{\mathit{car}_{in}}.\mathbf{Cars}(n+1) \\
&\quad + \\
&\quad \overline{\mathit{car}_{out}}.\ \text{if } n = 1 \text{ then } \mathbf{RSensor} \\
&\quad \text{else } \mathbf{Cars}(n-1)
\end{aligned}$$

3.1.4 Road lights and gate

RLight (which is initially green) is read by the road sensor. It continues to send the 'red' signal until it is told by *Control* to change. The gate receives the signal (a toggle) to change its state and having done so, sends the acknowledgment 'done' back to the control.

$$\begin{aligned}
\mathbf{RLight}(z) &\stackrel{def}{=} \overline{\mathit{send}_r(z)}.\mathbf{RLight}(z) \\
&\quad + \\
&\quad \mathit{change}_r.\overline{\mathit{send}_r}(\mathit{change}(z)).\mathbf{RLight}(\mathit{change}(z)) \\[1em]
\mathbf{Gate} &\stackrel{def}{=} \mathit{movegate}.\overline{\mathit{done}}.\mathbf{Gate}
\end{aligned}$$

3.1.5 Main control

All communications in 2.2 (3) to (5) take place through *Control*. These are entirely sequential, so it can be seen by inspection in what order the operations are performed, thus satisfying the conditions. *Control*, having sensed an approaching train, first changes the ALight (furthest up the track) to red to prevent any more trains entering the section. It then changes the road light to red and on receiving an acknowledgment, closes the gate. Only when this has been acknowledged does it change the ILight (on the track, closest to the crossing) to green to allow the train to proceed through the crossing. When it senses the train inside the crossing the ILight is changed back to red. When the train has left the crossing (both track lights are still red) the gate is opened, the road lights are changed to green and the traffic allowed to proceed. Lastly, the ALight is changed back to green.

$$\begin{aligned}\textbf{Control} \overset{def}{=}\ & a.\overline{change_a}.\overline{change_r}.sent.\overline{movegate}.done.\\ & \overline{change_i}.i.\overline{change_i}.o.\\ & \overline{movegate}.done.\overline{change_r}.sent.\overline{change_a}.\textbf{Control}\end{aligned}$$

3.1.6 The composed system (see Fig. 2)

The composed crossing system is given by

$$\begin{aligned}\textbf{Crossing} \overset{def}{=}\ & \textbf{ATrain}|\textbf{TA}|\textbf{TI}|\textbf{TO}|\textbf{ALight}(green)|\textbf{ILight}(red)\\ & |\textbf{Control}|\textbf{RLight}(green)|\textbf{Gate}|\textbf{RSensor} \setminus A\end{aligned}$$

where

$$\begin{aligned}A = \ & \{a, i, o, t_a, t_i, t_o, change_a, change_i, change_r,\\ & send_a, send_i, send_r, sent, movegate, done\}\end{aligned}$$

so that the only actions visible to the environment are $train_{in}$, $train_{out}$, car_{in}, car_{out}.

We now wish to analyse the behaviour of the Crossing as a whole. We can do this by applying the expansion theorem [Mil89] to the composed system; however, attempts to do this show that possible states for the whole system proliferate within one or two lines; the proof becomes unmanageably complex. We should like, if possible, to decouple the total system into smaller more manageable subsystems which may be analysed independently and then composed. The following is an attempt to achieve this.

3.2 Divide and conquer

We propose to divide the crossing system into two subsystems: the road vehicles with R_Control and the trains with T_Control. Only *Control* needs to be changed to do this; it is split into two smaller but communicating control subsystems, one each for the road vehicles and the trains. The extra signals introduced are *stopcars*, *gotrains*, *gocars* and *restore*; these are used to synchronize the two subsystems and prevent any interleaving of trains and cars.

3.2.1 Road control

R_Control cannot act until it has received the *stopcars* signal from T_Control. On receiving this, R_Control changes the road lights to red and on receiving an acknowledgment closes the gate. When this has been acknowledged, the signal *gotrains* is sent to T_Control and the signal *gocars* awaited. On receiving this, R_Control then changes the road lights back to green and opens the gate. Finally the *restore* signal is sent to T_Control.

$$\mathbf{R\_Control} \overset{def}{=} \begin{array}{l} stopcars.\overline{change_r}.sent.\overline{movegate}.done. \\ \overline{gotrains}.gocars.\overline{movegate}.done.\overline{change_r}.sent. \\ \overline{restore}.\mathbf{R\_Control} \end{array}$$

3.2.2 Train control

The T_Control having sensed an approaching train first changes the ALight to red to prevent any more trains entering the section. It then sends the signal *stopcars* to R_Control and cannot proceed until it has received the signal *gotrains*. On receiving this from R_Control, it sends the signal to change the ILight to green to allow the train to proceed through the crossing. When it senses the train inside the crossing the ILight is changed back to red. When the train has left the crossing (both track lights are still red) the signal *gocars* is sent to R_Control and the signal *restore* is awaited. On receiving this the ALight is changed back to green.

$$\mathbf{T\_Control} \overset{def}{=} \begin{array}{l} a.\overline{change_a}.\overline{stopcars}.gotrains. \\ \overline{change_i}.i.\overline{change_i}.o.\overline{gocars}.restore. \\ \overline{change_a}.\mathbf{T\_Control} \end{array}$$

3.2.3 The composed road vehicle and R_Control subsystem

The observable communications for the road subsystem are car_{in}, car_{out}, *stopcars*, *gotrains*, *gocars* and *restore*.

$$\begin{array}{rcl} \mathbf{RX} & = & \mathbf{RLight}(green)|\mathbf{Gate}|\mathbf{RSensor}|\mathbf{R\_Control} \\ & & \backslash\{change_r, movegate, send_r, sent, done\} \end{array}$$

3.2.4 The composed train and T_Control subsystem

The observable communications for the train subsystem are $train_{in}$, $train_{out}$, *stopcars*, *gotrains*, *gocars* and *restore*.

$$\begin{array}{rcl} \mathbf{TX} & = & \mathbf{TA}|\mathbf{TI}|\mathbf{TO}|\mathbf{ALight}(green)|\mathbf{ILight}(red)|\mathbf{T\_Control}|\mathbf{ATrain} \\ & & \backslash\{a, i, o, t_a, t_i, t_o, change_a, change_i, send_a, send_i\} \end{array}$$

3.2.5 The subsystems composed (see Fig. 2a)

$$\mathbf{Crossing2} = ((\mathbf{TX} \setminus T)|(\mathbf{RX} \setminus R)) \setminus C$$

where

$$\begin{array}{rl} T = & \{a, i, o, t_a, t_i, t_o, change_a, change_i, send_a, send_i\}, \\ R = & \{change_r, movegate, send_r, sent, done\} \text{ and} \\ C = & \{stopcars, gotrains, gocars, restore\} \end{array}$$

Before proceeding, we need to show that that the composition above and that in 3.1.6 are congruent. We can then consider the composed subsystems in the confidence that whatever is shown to be true for **Crossing2** will also be true for **Crossing**.

4 Equivalence of the single and decoupled systems

Before continuing, we recall the particular Restriction Laws which we shall be invoking, and define some notation.

The Restriction Laws [Mil89]

(1) $P \setminus L = P$ if $\mathcal{L}(P) \cap (L \cup \overline{L}) = \emptyset$
(2) $P \setminus K \setminus L = P \setminus (K \cup L)$
(3) ......(not invoked)
(4) $(P|Q) \setminus L = (P \setminus L)|(Q \setminus L)$ if $\mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \overline{L}) = \emptyset$

Notation

T, R and C are as defined in 3.2.5. $\mathcal{L}(P)$ is the *sort* of P, that is, its label set. An overbar above the name of such a set refers to the complement of that set. So

$$\begin{aligned}\mathcal{L}(TX) = {} & \{a, i, o, t_a, t_i, t_o, change_a, change_i, send_a, send_i, \overline{stopcars}, \\ & gotrains, \overline{gocars}, restore, train_{in}, train_{out}\}\end{aligned}$$

$$\begin{aligned}\mathcal{L}(RX) = {} & \{change_r, movegate, send_r, sent, done, stopcars, \overline{gotrains}, gocars, \\ & \overline{restore}, car_{in}, car_{out}\}\end{aligned}$$

From this we note that

(i) $T \cup R = A$ (as defined in 3.1.6)
(ii) $\mathcal{L}(TX) \cap (R \cup \overline{R}) = \emptyset$
(iii) $\mathcal{L}(RX) \cap (T \cup \overline{T}) = \emptyset$

4.1 Proving equivalence

First we need the following lemma:

Lemma 4.1 *Control* $= (T\_Control|R\_Control) \setminus C$

Proof

$$
\begin{aligned}
(\mathbf{T\_Control}|\mathbf{R\_Control}) \setminus C \quad & = (a.\overline{change_a}.\overline{stopcars}.gotrains.\overline{change_i}.i. \\
& \overline{change_i}.o.\overline{gocars}.restore.\overline{change_a}. \\
& \mathbf{T\_Control}|stopcars.\overline{change_r}.sent. \\
& \overline{movegate}.done.\overline{gotrains}.gocars. \\
& \overline{movegate}.done.\overline{change_r}.sent.\overline{restore}. \\
& \mathbf{R\_Control}) \setminus C \\
\\
& = (a.\overline{change_a}.\tau.\overline{change_r}.sent. \\
& \overline{movegate}.done.\tau.\overline{change_i}.i.\overline{change_i}.o. \\
& \tau.\overline{movegate}.done.\overline{change_r}.sent.\tau.\overline{change_a}. \\
& (\mathbf{T\_Control}|\mathbf{R\_Control}) \setminus C) \\
\\
& = (a.\overline{change_a}.\overline{change_r}.sent. \\
& \overline{movegate}.done.\overline{change_i}.i.\overline{change_i}.o. \\
& \overline{movegate}.done.\overline{change_r}.sent.\overline{change_a}. \\
& (\mathbf{T\_Control}|\mathbf{R\_Control}) \setminus C) \\
\\
& = \mathbf{Control} \ \Box
\end{aligned}
$$

We use this congruence in the following theorem.

Theorem 4.1 *Crossing2 = Crossing*

Proof

$$
\begin{aligned}
\textbf{Crossing2} &= ((TX \setminus T)|(RX \setminus R)) \setminus C \\
&= ((TX \setminus T \setminus R)|(RX \setminus R)) \setminus C \\
&\quad \text{(by (1) and (ii))} \\
&= (TX \setminus T|RX) \setminus R \setminus C \\
&\quad \text{(by (4))} \\
&= (TX \setminus T|RX \setminus T) \setminus R \setminus C \\
&\quad \text{(by (1) and (iii))} \\
&= (TX|RX) \setminus T \setminus R \setminus C \\
&\quad \text{(by (4))} \\
&= (TX|RX) \setminus C \setminus T \cup R \\
&\quad \text{(by (2))} \\
&= (ATrain|TA|\ldots|T\_Control)|(R\_Control|\ldots|RSensor) \\
&\quad \setminus C \setminus T \cup R) \text{ (by commutativity of `—')} \\
&= ATrain|TA|\ldots|(T\_Control|R\_Control)|\ldots|RSensor \\
&\quad \setminus C \setminus T \cup R \text{ (by associativity)} \\
&= (ATrain \setminus C|TA \setminus C|\ldots|(T\_Control|R\_Control) \setminus C| \\
&\quad \ldots|RSensor \setminus C) \setminus T \cup R \text{ (by (4))} \\
&= (ATrain|TA|\ldots|(T\_Control|R\_Control) \setminus C| \\
&\quad \ldots|RSensor) \setminus T \cup R \text{ (by (1))} \\
&= (ATrain|TA|\ldots|Control|\ldots|RSensor) \setminus T \cup R \\
&\quad \text{(by lemma 4.1)} \\
&= (ATrain|TA|\ldots|Control|\ldots|RSensor) \setminus A \\
&\quad \text{by (i)} \\
&= \textbf{Crossing} \qquad \square
\end{aligned}
$$

5 Analysis of the subsystems' behaviour

We now use the expansion theorem to analyse the behaviour of the two subsytems.

5.1 Expansion of the train subsystem

In the interests of clarity, only those agents which are able to communicate will be shown in full. Notice that there is no communication possible within this subsystem until a train approaches. We have specified the train so that having left the crossing, it returns to its original state; in other words, we are assuming a steady stream of trains with no notion of elapsed time in between. This is a limitation of (untimed) CCS.

Lemma 5.1.1

$$
\begin{aligned}
TX \setminus T = {} & \tau.\overline{stopcars}.gotrains.\overline{train_{in}}. \\
& \overline{train_{out}}.\overline{gocars}.restore.TX \setminus T
\end{aligned}
$$

Proof

$$\begin{aligned}
TX \setminus T &= ATrain|ALight(green)|TA|TI|ILight(red)|TO|T\_Control \setminus T \\
&= send_a(green).\overline{t_a}.ITrain|\overline{send_a}(green).change_a.ALight(red)| \\
&\quad TA|\ldots|a.\overline{change_a}\ldots T\_Control \setminus T \\
&= \tau.(\overline{t_a}.ITrain|change_a.ALight(red)|t_a.\overline{a}.TA| \\
&\quad \ldots|a.\overline{change_a}\ldots T\_Control) \setminus T \\
&= \tau.\tau.(ITrain|change_a.ALight(red)|\overline{a}.TA|ILight(red) \\
&\quad \ldots|a.\overline{change_a}\ldots T\_Control) \setminus T
\end{aligned}$$

At this point two sets of communications are running concurrently: the red signal between the ILight and the ITrain (modelling the train waiting at the lights while they show red), and the communication through the a and then $change_a$ ports of the control. These last two must take place before the visible signal *stopcars* is sent; the first must act, at the latest, immediately after the *gotrains* signal. Strictly speaking, the expansion hereafter should be expressed as a sum of the form $\tau \sum s_i$, where the s_i are the different states induced by the various placings of τ's. However, as interleaved τ's are absorbed then $s_i = s_j$ for all i, j and we are justified in examining one particular summand. The significant fact is the order in which the visible actions occur and this is fixed.

The sequential *T_Control* ensures that lights are not changed until the appropriate signals have been received. In particular, the ILight is not changed to green until after the *gotrains* signal. The state at which the system is ready to communicate *stopcars* is shown next; at this point, both sets of track lights are showing red (indicating also that any approaching train will be stopped outside the critical zone).

$$\begin{aligned}
&= \tau.(ITrain|ALight(red)|TA|TI|TO| \\
&\quad change_i.ILight(green)| \\
&\quad \overline{stopcars}.gotrains\ldots T\_Control) \setminus T \\
&= \tau.\overline{stopcars}.gotrains.(ITrain|ALight(red)|TA|TI|TO| \\
&\quad change_i.ILight(green)|\overline{change_i}.i\ldots T\_Control) \setminus T \\
&= \tau.\overline{stopcars}.gotrains.\tau.(send_i(green).\overline{t_i}.CTrain|ALight(red)|TA|TI| \\
&\quad TO|\overline{send_i}(green).ILight(green)|i\ldots T\_Control) \setminus T \\
&= \tau.\overline{stopcars}.gotrains.\tau.\tau.(\overline{t_i}.CTrain|ALight(red)|TA|t_i.\overline{i}.TI|TO| \\
&\quad ILight(green)|i\ldots T\_Control) \setminus T \\
&= \tau.\overline{stopcars}.gotrains.\tau.\tau.\tau.(CTrain|ALight(red)|TA|\overline{i}.TI|TO| \\
&\quad ILight(green)|i\ldots T\_Control) \setminus T
\end{aligned}$$

As above, we have more than one possible state to move to next, but for the same reasons we consider just one of these states. The most important safety feature is that *T_Control* must have received the *o* signal from *TO* (that is, the train must have left) before the *gocars* signal is sent.

$$\begin{array}{rl}
= & \tau.\overline{stopcars}.gotrains.\tau.\tau.\tau.\overline{train_{in}}.\overline{train_{out}}. \\
& (\overline{t_o}.ATrain|ALight(red)|TA|TI|t_o.\overline{o}.TO| \\
& ILight(green)|o.\overline{gocars}.restore.\overline{change_a}.T\_Control) \setminus T \\
& \\
= & \tau.\overline{stopcars}.gotrains.\tau.\tau.\tau.\overline{train_{in}}.\overline{train_{out}}.\overline{gocars}.restore. \\
& (ATrain|ALight(red)|TA|TI| \\
& TO|ILight(green)|\overline{change_a}.T\_Control) \setminus T \\
& \\
= & \tau.\overline{stopcars}.gotrains.\tau.\tau.\tau.\overline{train_{in}}.\overline{train_{out}}.\overline{gocars}.restore. \\
& (ATrain|ALight(green)|TA|TI|TO|ILight(red)|T\_Control) \setminus T \\
& \\
= & \tau.\overline{stopcars}.gotrains.\tau.\tau.\tau.\overline{train_{in}}.\overline{train_{out}}.\overline{gocars}.restore.\tau. \\
& TX \setminus T \\
= & \tau.\overline{stopcars}.gotrains.\overline{train_{in}}.\overline{train_{out}}.\overline{gocars}.restore.TX \setminus T \ \square
\end{array}$$

5.2 Expansion of the road subsystem

We show the following result:

Lemma 5.2.1

$$\begin{array}{rcl}
RX \setminus R & = & \tau. \\
& & (\overline{car_{in}} \ldots stopcars \ldots \overline{car_{out}}. \\
& & \overline{gotrains}.gocars.\overline{restore}.RX \setminus R \\
& & + \\
& & stopcars.\overline{gotrains}.gocars.\overline{restore}.RX \setminus R \\
& & + \\
& & \overline{car_{in}} \ldots \overline{car_{out}}.RX \setminus R) \\
& & + \\
& & stopcars.\tau. \\
& & (\tau.\overline{car_{in}} \ldots \overline{car_{out}}.\overline{gotrains}.gocars.\overline{restore}.RX \setminus R \\
& & + \\
& & \tau.\overline{gotrains}.gocars.\overline{restore}.RX \setminus R)
\end{array}$$

Proof

We begin by examining the behaviour of the road traffic *before* a train approaches, that is, before *R_Control* receives the *stopcars* signal. (There may, of course, be no cars present, in which case the road sensor will continue to 'watch' the lights until they change to red.) This involves communication between the cars and road lights only. Note that while *RLight* stays green, the movement of cars is independent of *Control*.

$$\begin{aligned}
\textbf{RSensor}|\textbf{RLight}(green) \setminus R &= send_r(green).Go.|(\overline{send_r}(green).\\
&\quad RLight(green) \setminus R\\
&= \tau.(\overline{car_{in}}Cars(1)|RLight(green)\\
&\quad +\\
&\quad (send_r(red).\overline{sent}.Stop|RLight(green)) \setminus R\\
&= \tau.(\overline{car_{in}}.(Cars(1)|RLight(green))\\
&\quad +\\
&\quad send_r(red).\overline{sent}.Stop|RLight(green)) \setminus R
\end{aligned}$$

In the *Cars* state, there can be no further communication with *RLight*; only *RSensor*, *Stop* and *Go* can 'see' the lights. It can be seen from the specification of *Cars* that in order to leave this state and return to a state which can see the lights, the number of car_{out} signals must equal the number of car_{in} signals, that is, the crossing area must be clear of cars before *RLight* can be read again. When this happens, *RLight* must be read before another car enters the crossing area; this will give rise to interleaved τ's between groups of car_{in}, car_{out} signals. Considering the above as part of RX, we see that since *stopcars* is visible, it can be received at any time during the activity described above. When this happens then $change_r$ is sent to *RLight*; *RSensor* cannot send the acknowledgment *sent* back to *R_Control* until all the cars that have entered the crossing have also left. We consider the behaviour of RX, then, from this point:

$$\begin{aligned}
RX \setminus R &= \tau.\\
&\quad (\overline{car_{in}} \ldots stopcars.\tau.(Cars(n)|\\
&\quad \overline{send_r}(red).RLight(red)|sent \ldots R\_Control|Gate)\\
&\quad +\\
&\quad stopcars.(send_r(red).\overline{sent}.Stop|\overline{send_r}(red).RLight(red)|\\
&\quad sent \ldots R\_Control|Gate)\\
&\quad +\\
&\quad \overline{car_{in}} \ldots \overline{car_{out}}.(RSensor|RLight(green)|RControl|Gate)) \setminus R\\
&\quad +\\
&\quad stopcars.\\
&\quad (RSensor|RLight(green)|change_r \ldots RControl|Gate) \setminus R\\
&= (\tau.(RX_1 + stopcars.RX_2 + \overline{car_{in}} \ldots \overline{car_{out}}.RX)\\
&\quad +\\
&\quad stopcars.RX_3) \setminus R
\end{aligned}$$

We consider the behaviour of RX_1 first:

$$\begin{aligned}
RX_1 \setminus R &= \overline{car_{in}} \ldots stopcars \ldots \tau.\overline{car_{out}}.(Cars(1)| \\
&\quad \overline{send_r}(red).RLight(red)|sent \ldots R\_Control|Gate) \setminus R \\
&= \overline{car_{in}} \ldots stopcars \ldots \tau..\overline{car_{out}}.RSensor| \\
&\quad \overline{send_r}(red).RLight(red)|sent \ldots R\_Control|Gate) \setminus R \\
&= \overline{car_{in}} \ldots stopcars \ldots \tau.\overline{car_{out}}.(send_r(red).\overline{sent}.Stop| \\
&\quad \overline{send_r}(red).RLight(red)|sent \ldots R\_Control|Gate) \setminus R \\
&= \overline{car_{in}} \ldots stopcars \ldots \tau.\overline{car_{out}}.RX_2 \setminus R
\end{aligned}$$

We now consider the behaviour of RX_2; the first communications are the hidden ones through $send_r$ and then $sent$:

$$\begin{aligned}
RX_2 \setminus R &= \tau.\tau.(Stop|RLight(red)|movegate \ldots R\_Control|Gate) \setminus R \\
&= \tau.\tau.(Stop|RLight(red)|\overline{movegate}.done \ldots R\_Control| \\
&\quad movegate.\overline{done}.Gate) \setminus R
\end{aligned}$$

The next several actions all communicate through *R_Control* sequentially, up to the point where *RLight* is changed back to green: the gate is closed and acknowledgment received, and then *gotrains* is sent to *T_Control*; on receiving *gocars* the gate is opened and the lights changed back to green. (As in the case of TX, the expansion here is a sum $\sum s_i$ with all summands s_i congruent.)

$$\begin{aligned}
RX_2 \setminus R &= \tau.\tau.\tau.\tau.\overline{gotrains}.gocars.\tau.\tau.\tau.\tau. \\
&\quad (RSensor|RLight(green)|\overline{restore}.R\_Control|Gate) \setminus R \\
&= \tau.\tau.\tau.\tau.\overline{gotrains}.gocars.\tau.\tau.\tau.\tau.\overline{restore}.RX \setminus R \\
&= \tau.\overline{gotrains}.gocars.\overline{restore}.RX \setminus R
\end{aligned}$$

Lastly we consider RX_3:

$$\begin{aligned}
RX_3 \setminus R &= (RSensor|RLight(green)|change_r \ldots RControl|Gate) \setminus R \\
&= \tau.(Go|RLight(green)|\overline{change_r} \ldots RControl|Gate) \setminus R \\
&\quad + \\
&\quad \tau.(RSensor|send_r(red).RLight(red)|sent \ldots RControl|Gate) \setminus R \\
&= \tau. \\
&\quad (\tau.car_{in} \ldots (Cars(k)|RLight(green)|\overline{change_r} \ldots RControl|Gate) \\
&\quad + \\
&\quad (\tau.(send_r(red).\overline{sent}.Stop|\overline{send_r}(red).RLight(red)| \\
&\quad sent \ldots RControl|Gate))) \\
&\quad + \\
&\quad \tau. \\
&\quad (send_r(red).\overline{sent}.Stop|\overline{send_r}(red).RLight(red)| \\
&\quad sent \ldots RControl|Gate) \setminus R \\
&= (\tau.(\tau.RX_4 + \tau.RX_2) + \tau.RX_2) \setminus R \\
&= (\tau.(\tau.RX_4 + \tau.RX_2)) \setminus R
\end{aligned}$$

The behaviour of RX_4 is substantially the same as that of RX_1, the only difference being in the position of *stopcars*. Substituting back into RX gives

$$\begin{aligned}
RX \setminus R &= \tau. \\
&\quad (\overline{car_{in}} \ldots stopcars \ldots \overline{car_{out}}.\overline{gotrains}.gocars.\overline{restore}. \\
&\quad RX \setminus R \\
&\quad + \\
&\quad stopcars.\overline{gotrains}.gocars.\overline{restore}.RX \setminus R \\
&\quad + \\
&\quad \overline{car_{in}} \ldots \overline{car_{out}}.RX \setminus R) \\
&\quad + \\
&\quad stopcars.\tau. \\
&\quad (\tau.\overline{car_{in}} \ldots \overline{car_{out}}.\overline{gotrains}.gocars.\overline{restore}.RX \setminus R \\
&\quad + \\
&\quad \tau.\overline{gotrains}.gocars.\overline{restore}.RX \setminus R) \qquad \Box
\end{aligned}$$

(Strictly speaking, *restore* may be sent at any time during the movement of cars from *Go* when the train has left, that is if there are cars present. However, as *restore* is hidden at the next level of composition, it will simply give rise to a τ interleaved between car_{in} and car_{out} signals. It has also been assumed that while the *RSensor* is in $Cars(k)$ state, the $change_r$ signal will be sent to *RLight* by *RControl*; in fact, this communication cannot be forced while the visible car_{in}, car_{out} are able to act. There is a possible world where cars use the crossing indefinitely, so preventing the trains from using it; i.e., $\#car_{in}$ tends to infinity.)

5.3 Expansion of Crossing2

From 3.2.5 and the congruence proved in 4.1

$$\begin{aligned}\textbf{Crossing} &= \textbf{Crossing2}\\ &= ((\textbf{RX} \setminus R)|(\textbf{TX} \setminus T)) \setminus C\end{aligned}$$

and so by composing the subsystems above, we prove the following result:

Theorem 5.1

$$\begin{aligned}
Crossing \quad = \quad & \tau.(\overline{car_{in}} \ldots \overline{car_{out}}.\overline{train_{in}}.\overline{train_{out}}.Crossing\\
& +\\
& \tau.\overline{train_{in}}.\overline{train_{out}}.Crossing\\
& +\\
& \overline{car_{in}} \ldots \overline{car_{out}} Crossing)\\
& +\\
& \tau.(\tau.\overline{car_{in}} \ldots \overline{car_{out}}.\overline{train_{in}}.\overline{train_{out}}.Crossing\\
& +\\
& \tau.\overline{train_{in}}.\overline{train_{out}}.Crossing)
\end{aligned}$$

Proof

$$\begin{aligned}
Crossing \quad = \quad & \tau.\\
& (\overline{car_{in}} \ldots \tau \ldots \overline{car_{out}}.\tau.\overline{train_{in}}.\overline{train_{out}}.\tau.\tau.Crossing\\
& +\\
& \tau.\tau.\overline{train_{in}}.\overline{train_{out}}.\tau.\tau.Crossing\\
& +\\
& \overline{car_{in}} \ldots \overline{car_{out}}.Crossing)\\
& +\\
& \tau.\\
& (\tau.\overline{car_{in}} \ldots \tau \ldots \overline{car_{out}}.\tau.\overline{train_{in}}.\overline{train_{out}}.\tau.\tau.Crossing\\
& +\\
& \tau.\tau.\overline{train_{in}}.\overline{train_{out}}.\tau.\tau.Crossing)\\
= \quad & \tau.(\overline{car_{in}} \ldots \overline{car_{out}}.\overline{train_{in}}.\overline{train_{out}}.Crossing\\
& +\\
& \tau.\overline{train_{in}}.\overline{train_{out}}.Crossing\\
& +\\
& \overline{car_{in}} \ldots \overline{car_{out}} Crossing)\\
& +\\
& \tau.(\tau.\overline{car_{in}} \ldots \overline{car_{out}}.\overline{train_{in}}.\overline{train_{out}}.Crossing\\
& +\\
& \tau.\overline{train_{in}}.\overline{train_{out}}.Crossing) \qquad \square
\end{aligned}$$

where $\#car_{in} = \#car_{out}$.
We can simplify the appearance of this by making the following substitution:
Let

- $\sigma_1 = \overline{car_{in}} \ldots \overline{car_{out}}$, where $\#car_{in} = \#car_{out}$:
- $\sigma_2 = \overline{train_{in}}.\overline{train_{out}}$ (either of these may include 0 or more τ's):
- $C = Crossing$.

Then

$$\begin{aligned} C = \;& \tau.(\sigma_1.\sigma_2.C + \tau.\sigma_2.C + \sigma_1.C) \\ & + \\ & \tau.(\tau.\sigma_1.\sigma_2.C + \tau.\sigma_2.C) \end{aligned}$$

(See Fig.3)

6 Proving the system safe using bisimulation

We recall $S1$ from 2.3:

$$\begin{aligned} SafeCrossing \;&\stackrel{def}{=}\; \overline{train_{in}}.\overline{train_{out}}.SafeCrossing \\ & \quad + \\ & \quad \overline{car_{in}}.SafeCrossing_{cars}(1) \\ SafeCrossing_{cars}(n) \;&\stackrel{def}{=}\; \overline{car_{in}}.SafeCrossing_{cars}(n+1) \\ & \quad + \\ & \quad \overline{car_{out}} \text{ if } n = 1 \text{ then } SafeCrossing \\ & \quad \text{else } SafeCrossing_{cars}(n-1) \end{aligned}$$

Making the substitution as above with $SC = SafeCrossing$, we have

$$SC = \sigma_2.SC + \sigma_1.SC$$

(See Fig. 4a).
Clearly this is not observation equivalent to *Crossing*. Every choice in *Crossing* is guarded by one or more τ actions, representing either the reading of a green light by *RSensor*, the sensing of an approaching train by *T_Control*, or some other internal signal informing the system behaviour.

In $S1$, however, there is an implicit choice which the observer may make but which, it could be argued, does not model the real situation. Taking Milner's [Mil89] analogy of agents of the form $p = a.p' + b.p''$ with a box having two buttons marked a and b, then either button may be pressed successfully; the observer may *choose* which button to press. Applying this to $S1$, it would seem that the choice between a car and a train is made by the observer, which is clearly not the case.

We could remove this choice by guarding each summand in *SafeCrossing* with a τ thus:

$$\begin{array}{lll} SafeCrossing' & \overset{def}{=} & \tau.\overline{train_{in}}.\overline{train_{out}}.SafeCrossing' \quad (S1') \\ & & + \\ & & \tau.\overline{car_{in}}.SafeCrossing_{cars}(1) \end{array}$$

Now, however, we are presented with the possibility of deadlock; the system might 'slip' into the first state, allowing trains to enter (so barring cars from doing so) when no train is approaching. τ's arise because of hidden internal communications whose nature is not known to the observer; all τ's look the same, whether they arise from a whim of the system or from responsible decision making procedures; and these may include desirable failsafe mechanisms. We cannot distinguish between 'good' and 'bad' τ's.

What we wish to model is the situation where the observer sees either a train or cars (but not both at the same time) and is never prevented from seeing either, but cannot choose between them; a proper 'choice' is made by the system, e.g., when an approaching train is sensed actions are performed which will eventually allow the train to cross. It might be possible to model this by guarding the choices in *SafeCrossing* with visible actions but this is moving further away from the real situation; the observer, from his helicopter presumably, sees cars and trains and nothing else.

We leave the question open for the present and make the above substitution in $S1'$, giving

$$SC' = \tau.\sigma_2.SC' + \tau.\sigma_1.SC'$$

(See Fig. 4b)
Assuming, then, that $S1'$ is the better model of the top level safety requirement, we can show that *Crossing* satisfies this new condition. The bisimulation, $\mathcal{R}$, is

$$\begin{array}{ll} \mathcal{R} = & \{(S_0,T_0),(S_1,T_0),(S_2,T_0),(S_3,T_2), \\ & (S_4,T_2),(S_5,T_1),(S_6,T_2),(S_7,T_2)\} \end{array}$$

and so *Crossing* and *SafeCrossing'* are observationally equivalent.

7 Conclusion

The level crossing is a 'real world' application of CCS and as such, illustrates some of the strengths and limitations of the language. Despite being a small system it is nevertheless too large to be analysed as a whole due to the proliferation of states quite early on in the analysis. However it was possible within the language to separate the single system into smaller communicating subsystems, provably equivalent to the original, with each subsystem smaller and so more amenable to analysis (though even here the states are numerous).
The behaviour of the system may be summarized as follows:

- the system has been shown to be equivalent to the required safe behaviour of a level crosssing, provided we considered condition $S1'$; given that CCS has no means of distinguishing between one silent action and another, a case can be made for either $S1$ or $S1'$:
- apparently the system can deadlock, since in $SafeCrossing'$ we know nothing about the nature of the τ's. At this level the system may 'choose' to stay indefinitely in, say, the state which allows cars to use the crossing and so prevent trains from doing so. We need to assume that the hidden actions are taken responsibly and safely by the system, but there is no mechanism within CCS for making this explicit. Without the τ's, $SafeCrossing$ seems to allow the observer to experiment with the system, rather than merely to observe it and this also is undesirable. Nevertheless, it can still be seen that at no time are both cars and trains prevented from using the crossing:
- the system fails safe; cars must be counted out of the crossing before a train may enter and the train must leave before cars may re-enter. If either a train or a car enters the crossing but fails to leave (whether it breaks down or for any other reason), the appropriate lights will remain on red, preventing other vehicles from entering the crossing:
- the required behaviour of the train (i.e., the driver) has been specified explicitly. However, in the case of road vehicles, responsible behaviour of drivers has been assumed. Drivers who attempt to use the crossing unlawfully do so at their own risk.

7.1

The required safe behaviour of the crossing was specified initially at a high level of abstraction; then, without going outside the language, individual components were specified in greater detail and it was shown that the composition of these was equivalent to the top level specification – or in other words, the system was proved safe. On this basis, we could continue 'down' through the life-cycle showing, by proving equivalence at each level, that the safety requirements were still satisfied and eventually mapping onto a high level language such as Ada.

7.2

The model has limitations, however. It is not possible within CCS to model the (apparently) non-deterministic arrival of trains wishing to use the crossing; in fact, if we were to incorporate two trains within TX, say, $ATrain$ and $BTrain$, then it is theoretically possible for the second to follow the first past each set of lights and into the next sector *before* the lights change. We have assumed that the safety features of the wider system will prevent a second train from following closely behind the first. This assumption is not unreasonable, but we need to be aware that we are making it. On the other hand, changing the *ALight* to red is the first action taken by *Control* on sensing an approaching train and changing it back to green is the last; so although we cannot model a second train approaching at a safe distance, we know from the state of *ALight* that it will be held on a red light until the first train has left the system.

CCS cannot detect the absence of a signal, only its presence. Since all communications are synchronized, it is not possible, for instance, for *Rlight* just to 'show' green; something must be there to 'see' it and so allowance has to be made for this in the specification of *RSensor*. 'If ... then ... else' statements may be used to test the parameters of an action, but not to test for the presence of the action itself; we cannot model interrupts in CCS. This is not insurmountable here but in other circumstances might give problems.

7.3

CCS contains no notion of elapsed time. Despite this, we were able to model the (apparently) temporal safety constraints as set out in [Gor87]. This was achieved in two ways. First, it *is* possible to express temporal ordering in CCS and it is this ordering, rather then time itself, which was required in safety conditions (3) to (5) of 2.2. Second, the fact that all signals produced have to be consumed means that an *ack* signal cannot be 'lost'; *Control* will wait indefinitely before allowing further actions and so the system as specified fails safe. On the other hand, as was stated in 5.2, it is possible in the CCS system for the visible actions car_{in}, car_{out} to prevent the $change_r$ action from being sent by *R_Control* to *RLight*; interpreted, this means that a steady stream of cars may prevent the train from using the crossing indefinitely. There is no way of forcing this communication in CCS, whereas in temporal logic it could be made explicit that this action must happen.

Condition (2), (which said that cars must have time to leave the crossing before the gate is lowered), seems to need some means of measuring time explicitly. In the specification of *RSensor* we have achieved the same end but by different means. Instead of measuring time, we count cars in and out and only allow the gate to be closed after the crossing has cleared. This is inherently safer than allowing a fixed period of time to elapse.

References

[Gol87] R. Goldblatt. *Logics of time and computation.* Center for the study of Language and information, 1987.

[Gor87] Janusz Gorski. Formal support for development of safety related systems. In *Safety and reliability Symposium*, 1987.

[HW] W. T. Harwood and J. C. P. Woodcock. Temporal observations over CCS - a notation. As yet unpublished.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic.* Springer-Verlag, 1971.

[Wal87] David Walker. Introduction to a calculus of communicating systems. Technical report, Laboratory for Foundations of Computer Science, University of Edinburgh, 1987. Expository Report.

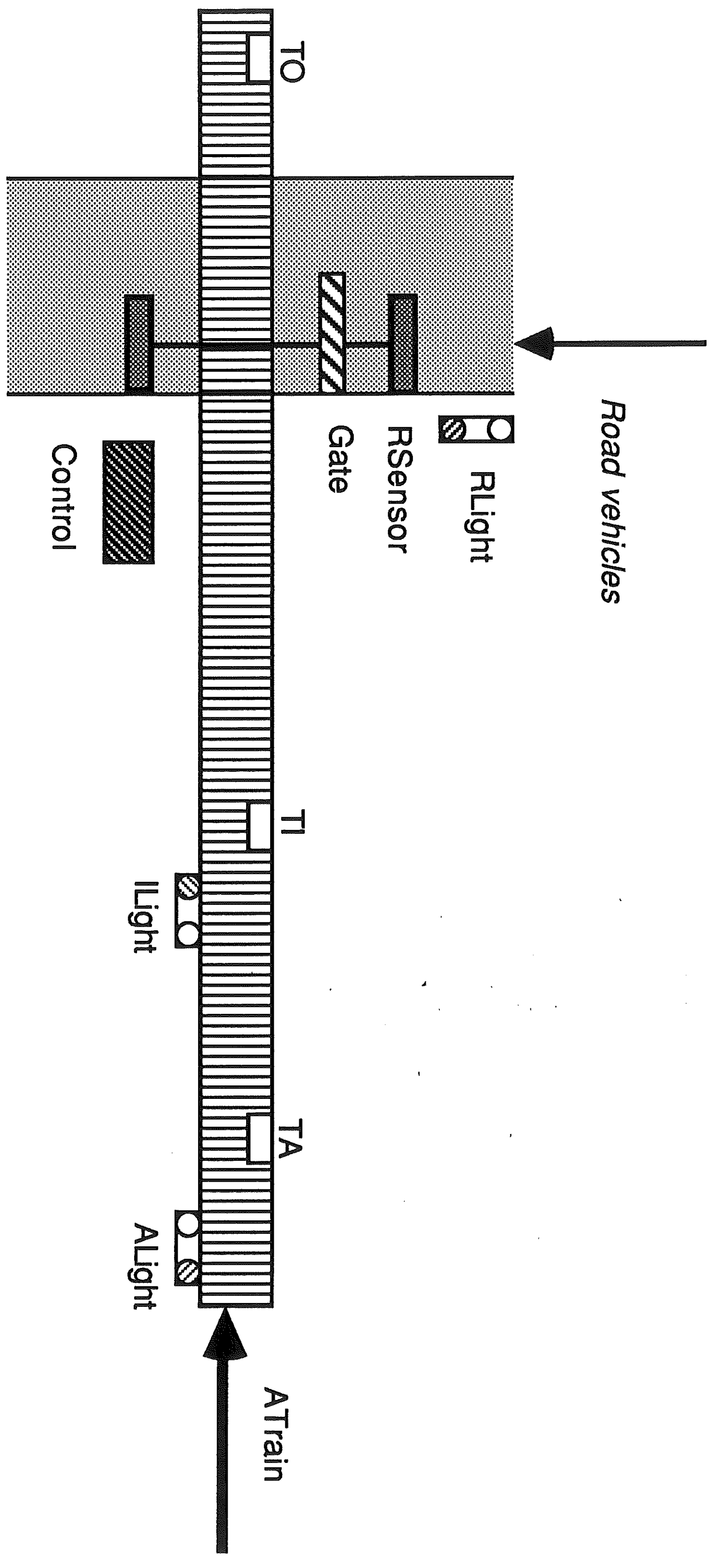


Fig 1. Crossing Layout

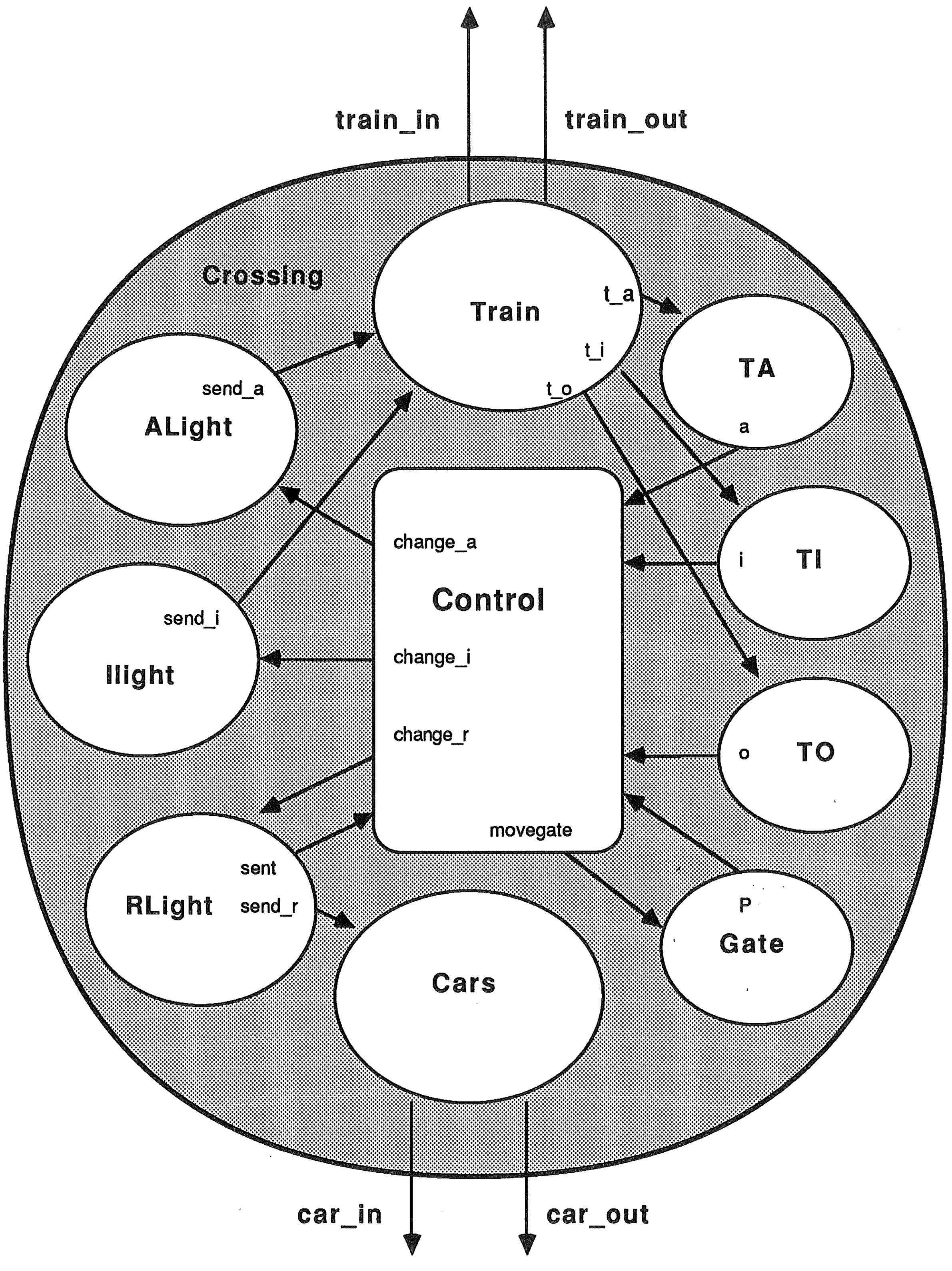


Fig 2. Crossing communications: single system with restricted and observable actions.

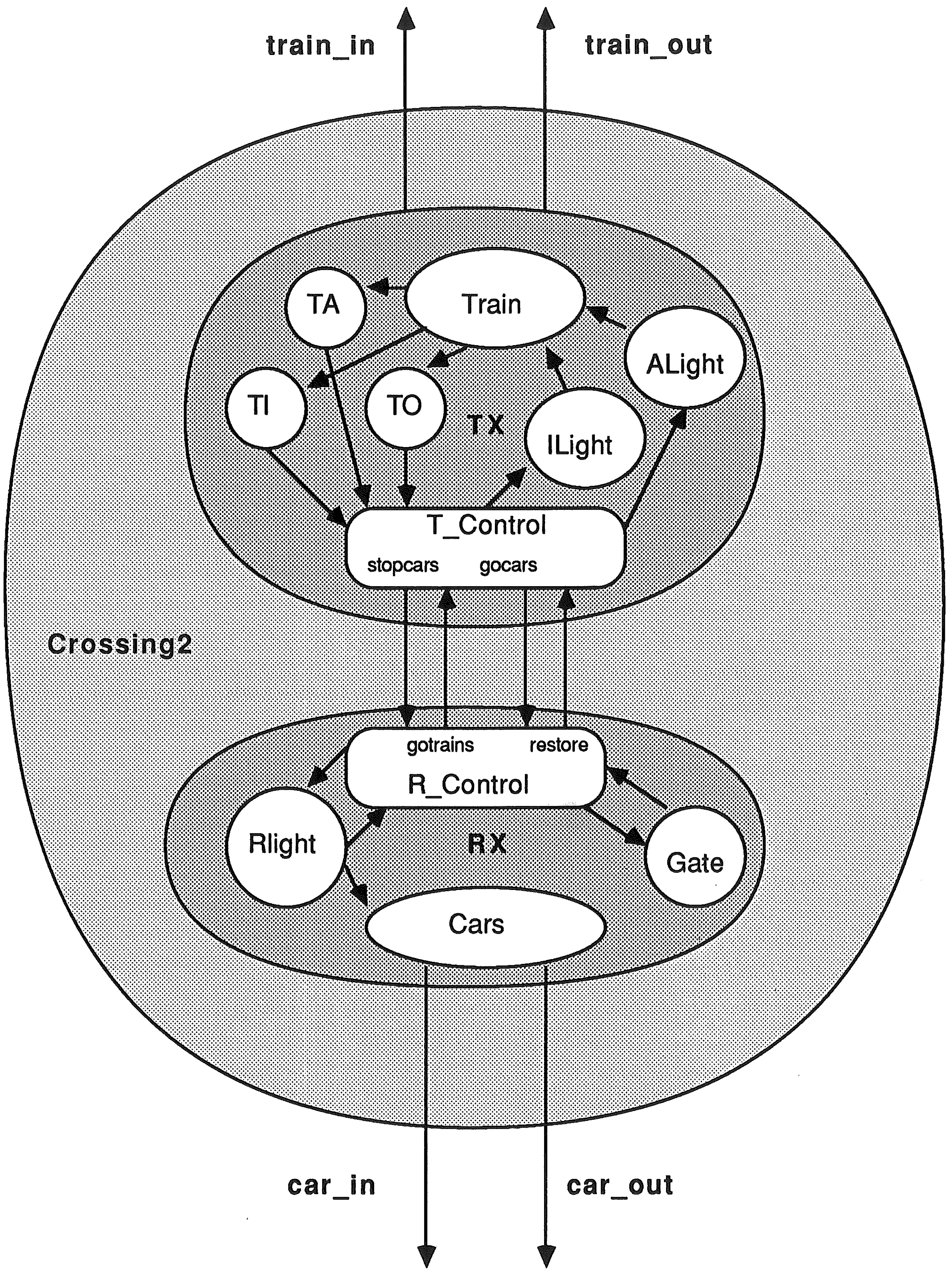


Fig.2a: Crossing2 communications: composition of divided system showing scope and visibility of actions

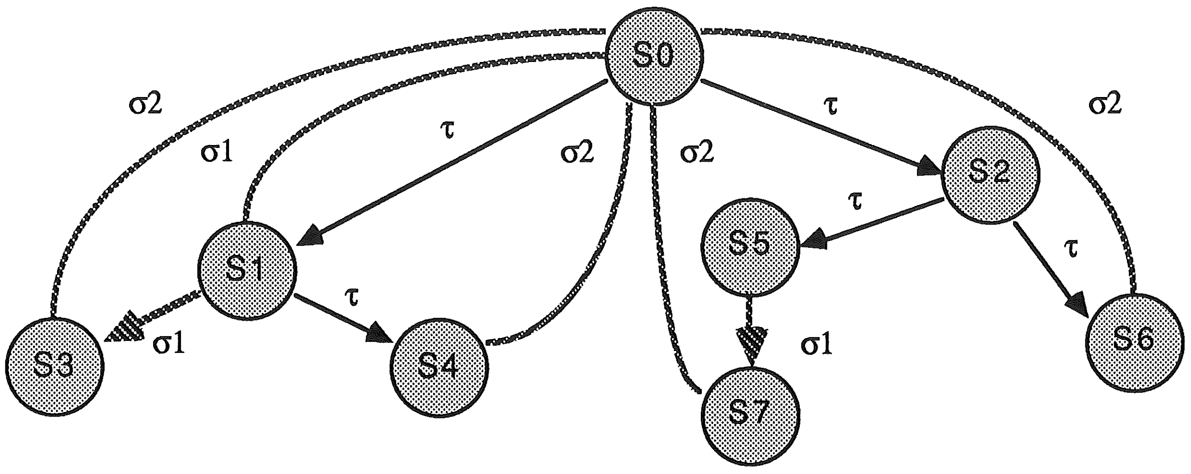
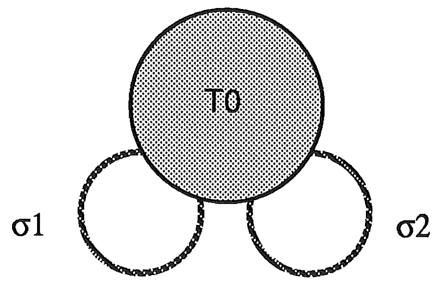


Fig.3 State diagram for Crossing2 (arcs without arrows are directed towards S0)



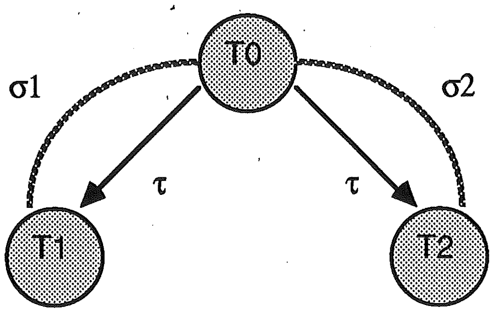


Fig 4a State diagram for SafeCrossing (S1)

Fig 4b Revised state diagram (S1')