A CSP solution to the "trains" problem

A.W. Roscoe
Programming Research Group, University of Oxford

The problem as stated leaves one in doubt as to what one must do. Is one meant to abstractly specify the safety and liveness properties one desires of the network? Is one meant to provide some plausible "implementation" which gaurantees the basic principle that two trains must not be simultaneously on the same piece of track? Perhaps we are even meant to enter the realm of inventing scheduling algorithms for the flow of trains. Of course any complete solution to the problem must, in some sense, address all of these issues. The scheduling problem seems, however, to be outside the scope of the workshop.

The solution presented here is a CSP program which addresses the second question above. It can be proved to satisfy certain abstract conditions, and can of course be used as a base (guaranteeing safety) upon which to build any scheduling programs. It is by no means the simplest program which guarantees safety: instead it shows how CSP can be used to create a control structure more realistic than if, say, the actual events of a train entering or leaving a piece of track were "negociable".

The only approach we make towards scheduling is to allow trains to book line sections one step ahead, and to seek an alternative line if the first one they try to book is in use. The element of external control left in the network is in the hands of (notional) train drivers. It would of course be expected that the drivers be controlled and regulated by some signalling system; this higher level of detail is omitted.

Of course the particular control structure chosen here is only one of many which have the desired safety properties. No special merit is claimed for the particular solution given, and the reader is invited to devise his own program to model his own ideas on how the network should be regulated and controlled. The main purpose of this solution is to illustrate the power of CSP in precisely specifying parallel interaction.

As in all CSP programs, communication is achieved by the synchronisation of *events*, or *actions*, of the elements of the network. Each process has its own *alphabet* of events; in the operation of the network, an event can only occur if all the processes in whose alphabets it is are willing to communicate it. This situation is easiest to imagine when no event is shared by more than two processes, since we then avoid the situation where three (or more) processes have to agree on some communication. This convention has therefore been followed in the present solution.

The following solution can be applied to any network of the form described in the problem, except that one simplifying assumption has been made. This is that all lines consist of only one section: of course, this restriction is not serious, as the old section boundaries can be transformed into (rather simple) extra crossing points. There are three types of process in the network: LINE, TRAIN, and CP (crossing point).

Each line has a unique name (typically λ), and two end (crossing-) points (typically α, β) which we assume distinct. In use, a line has a direction and becomes an ordered triple (λ,α,β) (the line with name λ, being used to travel from α to β). We will use the notation l and $\bar{l}$ to denote the two senses of any line, and follow the convention that $\bar{\bar{l}} = l$. If l is any (directed) line, N(l). S(l) and E(l) will denote its name, starting point, and end point respectively.

For each line λ, there will be a process LINE(λ) in our network. It acts as a resource which, when not in use, is prepared to be acquired by either of its end points. When it is in use it tells anyone wishing to use it that it is unavailable. Whichever end it is acquired by, it is willing to be released by either (since a train might have travelled from one end to the other).

Thus, for each line λ, whose endpoints are α,β , we define

```
LINE(λ)  =  get.λ.α → con.λ.α → BUSY(λ)
            ▯ get.λ.β → con.λ.β → BUSY(λ)

BUSY(λ)  =  get.λ.α → dis.λ.α → BUSY(λ)
            ▯ get.λ.β → dis.λ.β → BUSY(λ)
            ▯ rel.λ.α → LINE(λ)
            ▯ rel.λ.β → LINE(λ)
```

The alphabet of the processes LINE(λ), BUSY(λ) are defined to be the set of symbols which they can possibly use. Note that the alphabets of distinct LINEs are disjoint.

Each train has a unique name (typically t). The train runs around the network, controlled by its driver. Before it can reach a crossing point it must have a line booked on which to leave the crossing point (it is assumed for simplicity that trains always leave a crossing point on a different line to the one on which they enter it). When a train has no line booked it is prepared either to negociate with its next crossing point to book one, or to reverse its direction of travel. When a train has a line booked it is allowed either to enter the new line (via the crossing point) or to reverse (which releases its booked line). We denote the set of all train names by T.

The process TRAIN(t,l) represents train t running along the directed line l, with no other line booked. The process TRAIN(t,l,l') (where $E(l) = S(l')$) represents train t running along line l with line l' booked at E(l).

Thus, when $\beta = E(l)$, $E(l) = S(l')$ we define

$$\begin{aligned}
\mathrm{TRAIN}(t,l) \;=\;& \mathrm{reverse}.t \rightarrow \mathrm{TRAIN}(t,\overline{l}) \\
& \square \; (\underset{S(m)=\beta}{\square} \; \mathrm{book}.t.m \rightarrow \mathrm{req}.t.m \rightarrow (\mathrm{con}.t.m \rightarrow \mathrm{booked}.t.m \rightarrow \mathrm{TRAIN}(t,l,m) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \square \; \mathrm{ref}.t.m \rightarrow \mathrm{refused}.t.m \rightarrow \mathrm{TRAIN}(t,l)))
\end{aligned}$$

$$\begin{aligned}
\mathrm{TRAIN}(t,l,l') \;=\;& \mathrm{reverse}.t \rightarrow \mathrm{rel}.t.l' \rightarrow \mathrm{TRAIN}(t,\overline{l}) \\
& \square \; (\mathrm{goto}.t.\beta \rightarrow \mathrm{arrive}.t.l \rightarrow \mathrm{enter}.t.l' \rightarrow \mathrm{leave}.t.l \rightarrow \\
& \qquad\qquad\qquad\qquad\qquad \mathrm{rel}.t.l \rightarrow \mathrm{TRAIN}(t,l') \;)
\end{aligned}$$

Once again, the alphabets of TRAIN(t,l) and TRAIN(t,l,l') are the set of all events which they can possibly ever use. Communication with the driver has one of the forms "reverse", "book", "booked", "refused", "goto". All other events are communications with a crossing point. The alphabets of distinct trains are disjoint, and the alphabets of TRAINs are disjoint from those of LINEs.

Crossing points also have unique names (typically α, β). They have two roles. Firstly they act as intermediaries between TRAINs and LINEs. Secondly they control the timings of transitions from one line to another - specifically they only allow one train to be using them at any time (to avoid crashes). The solution given here treats these two aspects of their behaviour as independent; a more sophisticated solution might include a third parallel process representing a "register of bookings" to prevent "rogue" trains from entering, and perhaps to assist in the changing of points. (The solution here is good enough for the well-behaved TRAINs defined above.)

$$\mathrm{CP}(\alpha) \;=\; \mathrm{CP1}(\alpha) \parallel \mathrm{CP2}(\alpha) \;, \quad \text{where}$$

$$\begin{aligned}
\mathrm{CP1}(\alpha) \;=\;& \underset{\substack{S(l)=\alpha \\ t \in T}}{\square} \mathrm{req}.t.l \rightarrow \mathrm{get}.(N(l)).\alpha \rightarrow (\mathrm{con}.(N(l)).\alpha \rightarrow \mathrm{con}.t.l \rightarrow \mathrm{CP1}(\alpha) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad \square \; \mathrm{dis}.(N(l)).\alpha \rightarrow \mathrm{ref}.t.l \rightarrow \mathrm{CP1}(\alpha) \;) \\
& \square \; (\underset{\substack{S(l)\neq\alpha \\ t \in T}}{\square} \mathrm{rel}.t.l \rightarrow \mathrm{rel}.N(l).\alpha \rightarrow \mathrm{CP1}(\alpha) \;)
\end{aligned}$$

$$\mathrm{CP2}(\alpha) \;=\; \underset{\substack{E(l)=\alpha \\ t \in T}}{\square} \mathrm{arrive}.t.l \rightarrow (\underset{S(m)\neq\alpha}{\square} \mathrm{enter}.t.m \rightarrow \mathrm{leave}.t.l \rightarrow \mathrm{CP2}(\alpha) \;)$$

The alphabets of $\mathrm{CP}(\alpha)$, $\mathrm{CP1}(\alpha)$ and $\mathrm{CP2}(\alpha)$ again consist of precisely the symbols which could ever be used by them. Note that the alphabets of $\mathrm{CP1}(\alpha)$ and $\mathrm{CP2}(\alpha)$ are disjoint, as are the alphabets of distinct CPs. Note further that with the exception of the "train-driver" commands described above, every symbol in the alphabet of a LINE or TRAIN is in the alphabet of some CP, and that every symbol in the alphabet of each CP is contained in the alphabet of some LINE or TRAIN.

The solution to the problem is thus

$$\text{TRAINS} \parallel \text{CPS} \parallel \text{LINES}$$

where TRAINS, CPS and LINES are respectively the parallel composition of all TRAINs, CPs and LINEs, correctly initalised to reflect their starting position.

The system as it stands has two levels of "user". Firstly, the TRAINs can be thought of as using the "network" which consists of the CPs and LINEs. To these users the communications between CPs and LINEs are irrelevant and should be hidden from them. We should thus form the process

$$\text{NETWORK} = (\text{CPS} \parallel \text{LINES})/L,$$

where L is the union of the alphabets of the LINEs. NETWORK is a process which interacts correctly with TRAINs without unnecessary communication. The second level of user is provided by the train drivers. To these users the "booking" communications between TRAINs and CPs are irrelevant and should be hidden. (The remaining communications are interesting since they reveal the train's position.) Thus to the drivers the system appears as

$$\text{SYSTEM} = (\text{TRAINS} \parallel \text{NETWORK})/B,$$

where B is the set of all symbols of the form req.t.m, con.t.m, ref.t.m or rel.t.m.

We thus see that CSP allows us to construct systems in a structured fashion, progressively hiding internal communication.

CSP also allows us to analyse our programs by abstract methods. We can, for example, specify and prove correctness properties of programs by using mathematical models such as "traces" and "refusal-sets". The traces model [H] lets us prove partial correctness properties of processes, by studying their possible sequences of communications (traces). Such a property one could prove of our present process SYSTEM is (for each line l)

$$s \in \text{traces(SYSTEM)} \Rightarrow \text{length}(s \,|\, \{\text{leave.t.l}, \text{leave.t.}\bar{l} : t \in T\}) \geqslant \text{length}(s \,|\, \{\text{enter.t.l}, \text{enter.t.}\bar{l} : t \in T\}) - C,$$

where C is 0 or 1 depending on whether there is or is not a train initially on l (or $\bar{l}$). (This property simply says that at no time can there be more than one train on any line.)

The slightly more complex refusal-sets model ([HBR], improved in [BR], [B] [R]) allows us to specify total correctness properties, for we study not only the traces of processes, but also the sets of communications which a process can refuse at each point in its history. A typical (and weak) property one could prove of SYSTEM is "freedom from deadlock" (i.e. at all times there is at least something SYSTEM can do. This model would allow one to precisely analyse the causes of local deadlock and related conditions in SYSTEM. In either model it is easy to show that the structured hiding used to construct SYSTEM has no effect on the external behaviour of the system: all hiding can be moved to the outside.

$$\text{SYSTEM} = (\text{TRAINS} \parallel \text{CPS} \parallel \text{LINES})/(L \cup B)$$

References

[B] S.D. Brookes, A Model for Communicating Sequential Processes, Oxford D.Phil Thesis (1983) (available as a Carnegie-Mellon Tech. Report).

[BR] S.D. Brookes and A.W. Roscoe, An Improved Failure-set Model for Communicating Processes, To appear.

[H] C.A.R. Hoare, A Model for Communicating Sequential Processes, Tech. Report PRG-22, Oxford University Programming Research Group.

[HBR] C.A.R. Hoare, S.D. Brookes and A.W. Roscoe, A Theory of Communicating Sequential Processes, Tech. Report PRG-16, Oxford University Programming Research Group. (To appear in an extended form in JACM, also presently available as a CMU Tech. Report.)

[R] A.W. Roscoe, A Mathematical Theory of Communicating Processes, Oxford D. Phil Thesis (1982).