

◇ Input and Output ◇

So far we have treated all events in the same way, regardless of whether they are thought of as inputs or outputs. It is useful, however, to introduce separate notation for inputs and outputs.

We will use events of the form $c.v$ where c is the name of a *channel* and v is the *value* of a message passing along the channel. Each channel has a *type*, which is simply the set of possible values which can be transmitted along it. If the type of c is T , then the set of events associated with c is $\{c.t \mid t \in T\}$.

We can define two new forms of prefixing. The process $c!v \rightarrow P$ outputs the message v on the channel c and then behaves like P . We require $v \in T$, where T is the type of c . In fact, $c!v \rightarrow P = c.v \rightarrow P$ (using the ordinary prefix notation), but the $c!v$ notation emphasises the fact that c and v are viewed as a channel and a message.

The process $c?x : T \rightarrow P(x)$ is prepared to input any value x of type T , and then behave like $P(x)$. In the ordinary menu choice notation,

$$\begin{aligned} c?x : T \rightarrow P(x) = \\ y : \{c.z \mid z \in T\} \rightarrow P(\text{message}(y)), \\ \text{where, if } y = c.z, \text{ message}(y) = z. \end{aligned}$$

We can define input and output prefixes, using labelled transition rules, as follows.

$$\frac{}{(c!v \rightarrow P) \xrightarrow{c.v} P}$$

$$\frac{}{(c?x : T \rightarrow P(x)) \xrightarrow{c.v} P(v)} \quad v \in T$$

Example:

$$COPYBIT = in?x : \{0, 1\} \rightarrow out!x \rightarrow COPYBIT$$

$$COPY = in?x : \mathbb{N} \rightarrow out!x \rightarrow COPY$$

$$SQUARE = in?x : \mathbb{Z} \rightarrow out!(x*x) \rightarrow SQUARE$$

◇ Specifications ◇

Recall the definitions for the specification of the system consisting of the student and the college.

$$\begin{aligned}
 STUDENT &= year1 \rightarrow (pass \rightarrow YEAR2 \\
 &\quad \mid fail \rightarrow STUDENT) \\
 YEAR2 &= year2 \rightarrow (pass \rightarrow YEAR3 \\
 &\quad \mid fail \rightarrow YEAR2) \\
 YEAR3 &= year3 \rightarrow (pass \rightarrow graduate \rightarrow Stop \\
 &\quad \mid fail \rightarrow YEAR3) \\
 COLLEGE &= fail \rightarrow CF \mid pass \rightarrow C1 \\
 C1 &= fail \rightarrow CF \mid pass \rightarrow C2 \\
 C2 &= fail \rightarrow CF \mid pass \rightarrow prize \rightarrow Stop \\
 CF &= fail \rightarrow CF \mid pass \rightarrow CF
 \end{aligned}$$

$$SYSTEM = STUDENT \parallel_C COLLEGE$$

Initially we defined

$$\begin{aligned}
 SPECF &= pass \rightarrow SPECF \mid fail \rightarrow SPECF \\
 SPEC1 &= pass \rightarrow SPEC1 \mid fail \rightarrow SPECF \\
 SPEC2 &= pass \rightarrow SPEC2 \mid fail \rightarrow SPECF \\
 SPEC2 &= pass \rightarrow prize \rightarrow Stop \mid fail \rightarrow SPECF
 \end{aligned}$$

but the specification

$$SPEC1 \sqsubseteq_t SYSTEM$$

is not quite what we want, because it does not allow *SYSTEM* to do *year1*, *year2*, *year3* or *graduate*.

◇ The Correct Specification ◇

To allow for *year1*, *year2*, *year3* and *graduate* we defined

$$\begin{array}{l} EXTRA = year1 \rightarrow EXTRA \\ \quad | \quad year2 \rightarrow EXTRA \\ \quad | \quad year3 \rightarrow EXTRA \\ \quad | \quad graduate \rightarrow EXTRA \end{array}$$

and then

$$SPEC = SPEC_P \parallel_E EXTRA$$

where

$$\begin{array}{l} SP = \{pass, fail, prize\} \\ E = \{year1, year2, year3, graduate\}. \end{array}$$

In general, to simplify the definition of processes such as *EXTRA*, we can define, for any set *A* of events, the process *Run_A*.

$$Run_A = x : A \rightarrow Run_A$$

Then $EXTRA = Run_E$, and $SPEC_F = Run_{\{pass, fail\}}$.

◇ Hiding ◇

There is an alternative approach to this kind of specification. Instead of putting a process in parallel with the specification to generate the events which we don't care about, we can *hide* those events from the process being specified.

If we define

$NEWSYSTEM =$

$SYSTEM \setminus \{year1, year2, year3, graduate\}$

then the behaviour of $NEWSYSTEM$ is derived from that of $SYSTEM$ by making the listed events invisible. The traces of $NEWSYSTEM$ are the traces of $SYSTEM$ with these events removed.

Now we can simply write

$SPECP \sqsubseteq_t NEWSYSTEM.$

as the specification. $SPECP$ only involves the events which we are interested in, and the hiding in the definition of $NEWSYSTEM$ shows which events we are leaving out of the specification.

◇ The Level Crossing ◇

As an example of writing a specification in CSP, we will look at a railway level crossing. One road and one railway line cross each other, and as usual there is a gate which can be lowered to prevent cars crossing the railway. If the gate is raised, then cars can freely cross the track. Trains can cross the road regardless of whether the gate is up or down.

We will consider the obvious safety property for the level crossing, which is:

There should never be a train and a car on the crossing at the same time.

Of course there are many other properties which we might like to specify, for example a liveness property:

Whenever a car approaches the crossing, it should eventually be able to cross.

but for the moment we will stick to safety.

We will use the following events to represent the interesting aspects of the behaviour of the system.

car.approach, car.enter, car.leave, train.approach, train.enter, train.leave, gate.lower, gate.raise, crash

The processes *CARS* and *TRAINS* supply streams of cars and trains.

$$\begin{aligned} \text{CARS} &= \text{car.approach} \rightarrow \text{car.enter} \rightarrow \\ &\quad \text{car.leave} \rightarrow \text{CARS} \end{aligned}$$
$$\begin{aligned} \text{TRAINS} &= \text{train.approach} \rightarrow \text{train.enter} \rightarrow \\ &\quad \text{train.leave} \rightarrow \text{TRAINS} \end{aligned}$$

The following processes model the behaviour of the crossing. This is a complete description of all possibilities, including a car and a train simultaneously using the crossing. Later we will add a control process which uses the gate to restrict access by cars.

C, *T*, *CT* and *CR* correspond to the crossing when there is a car, train, both or neither present.

GATE models the gate; notice that a car cannot enter the crossing when the gate is down.

$$\begin{aligned} \text{CR} &= \text{car.approach} \rightarrow \text{car.enter} \rightarrow C \\ &\quad | \text{train.approach} \rightarrow \text{train.enter} \rightarrow T \\ C &= \text{car.leave} \rightarrow \text{CR} \\ &\quad | \text{train.approach} \rightarrow \text{train.enter} \rightarrow \text{CT} \\ T &= \text{train.leave} \rightarrow \text{CR} \\ &\quad | \text{car.approach} \rightarrow \text{car.enter} \rightarrow \text{CT} \\ \text{CT} &= \text{crash} \rightarrow \text{Stop} \\ \text{GATE} &= \text{gate.lower} \rightarrow \text{gate.raise} \rightarrow \text{GATE} \\ &\quad | \text{car.enter} \rightarrow \text{GATE} \end{aligned}$$

Defining some sets of events:

$$E_T = \{train.approach, train.enter, train.leave\}$$

$$E_C = \{car.approach, car.enter, car.leave\}$$

$$E_{GC} = \{gate.raise, gate.lower, car.enter\}$$

$$E_X = \{crash\}$$

$$E_S = E_T \cup E_C \cup E_G \cup E_X$$

$$E_{TCC} = E_T \cup E_C \cup E_X$$

$$E_{TCG} = E_T \cup E_C \cup E_G$$

allows us to define the whole system as

$$SYSTEM = ((CR_{E_{TCC}} \parallel_{E_G} GATE)_{E_S} \parallel_{E_C} CARS)_{E_S} \parallel_{E_T} TRAINS.$$

To specify that no crashes occur, we need a process *SPEC* which can do any events except for *crash*.

$$SPEC = Run_{E_{TCG}}.$$

The requirement that the crossing satisfies this specification is expressed by

$$SPEC \sqsubseteq_t SYSTEM.$$

FDR can be used to show that the specification is not satisfied.

◇ Controlling the Crossing ◇

Now we will define a process *CONTROL* which, when placed in parallel with *SYSTEM*, will constrain the behaviour so that whenever a train approaches the gate must be lowered. This will be achieved by making *CONTROL* and *SYSTEM* synchronise on certain events. We hope that the result will be a system which satisfies the safety specification.

$$\begin{aligned} \text{CONTROL} = & \text{train.approach} \rightarrow \text{gate.lower} \rightarrow \\ & \text{train.enter} \rightarrow \text{train.leave} \rightarrow \\ & \text{gate.raise} \rightarrow \text{CONTROL} \\ & | \text{car.approach} \rightarrow \text{car.enter} \rightarrow \\ & \text{car.leave} \rightarrow \text{CONTROL} \end{aligned}$$

$$\text{SAFE_SYSTEM} = \text{SYSTEM} \parallel_{E_S \parallel E_{TCG}} \text{CONTROL}$$

Again, FDR can be used to test whether

$$\text{SPEC} \sqsubseteq_t \text{SAFE_SYSTEM}$$

and this time we will find that it does.

◇ Using Hiding ◇

Alternatively, we can use hiding to avoid specifying the events which we don't care about. In this case, all we want to do is specify that *crash* never occurs.

If we hide all the events except *crash* from *SYSTEM* (or *SAFE_SYSTEM*) then all we need for the specification is a process which never does *crash*:

$$Stop \sqsubseteq_t SYSTEM \setminus (E_{TCG})$$

◇ Another Level Crossing ◇

Here is another way of modelling the level crossing.
Remove the *crash* event, and change the definition of CT to

$$CT = \begin{array}{l} car.leave \rightarrow T \\ | \quad train.leave \rightarrow C. \end{array}$$

Also change the definition of E_S to

$$E_S = E_C \cup E_T \cup E_G.$$

As before,

$$SYSTEM = (CR_UP \parallel_{E_S} CARS) \parallel_{E_T} TRAINS.$$

The specification process is

$$SPEC = \begin{array}{l} train.approach \rightarrow train.enter \rightarrow \\ \quad train.leave \rightarrow SPEC \\ | \quad car.approach \rightarrow car.enter \rightarrow \\ \quad car.leave \rightarrow SPEC \end{array}$$

and we can either write

$$SPEC \parallel_{E_C \cup E_T} Run_{E_G} \sqsubseteq_t SYSTEM$$

or

$$SPEC \sqsubseteq_t (SYSTEM \setminus E_G)$$