$\diamond$ Failure Equivalence $\diamond$

A first attempt at a new definition of process equivalence might be to define $P =_r Q$ as

$$traces(P) = traces(Q)$$
$$refusals(P) = refusals(Q)$$

but this is not quite what we want. It would make

$$a \rightarrow ((b \rightarrow Stop) \,\square\, (c \rightarrow Stop))$$

and

$$a \rightarrow ((b \rightarrow Stop) \sqcap (c \rightarrow Stop))$$

equivalent, which is no better than using trace equivalence. The problem is that looking at *refusals* can only detect differences at the first step. As with the definition of determinism, we need to look at events refused after arbitrary traces have been observed.

The solution is to define $failures(P)$ as follows:

$$failures(P) \;=\; \{(s, X) \mid s \in traces(P) \text{ and } X \in refusals(P \,/\, s)\}$$

and then say that $P =_f Q$ means

$$traces(P) = traces(Q)$$
and
$$failures(P) = failures(Q).$$

Recall that $\{\} \in \mathit{refusals}(P)$ for every process P. This means that for every process P and every trace $s \in \mathit{traces}(P)$, $(s, \{\}) \in \mathit{failures}(P)$. So *traces* can be recovered from *failures* by

$$\mathit{traces}(P) = \{s \mid (s, \{\}) \in \mathit{failures}(P)\}.$$

This means that if $\mathit{failures}(P) = \mathit{failures}(Q)$ then $\mathit{traces}(P) = \mathit{traces}(Q)$, so the definition of failure equivalence can be simplified to

$$\mathit{failures}(P) = \mathit{failures}(Q).$$

If P is deterministic, we can analyse $\mathit{failures}(P)$ slightly more.

$$\begin{aligned}
&\mathit{failures}(P)\\
&\quad = \{(s, X) \mid s \in \mathit{traces}(P) \text{ and } X \in \mathit{refusals}(P \ / \ s)\}\\
&\quad = \{(s, X) \mid s \in \mathit{traces}(P)\\
&\qquad\quad \text{and } X \cap \mathit{initials}(P \ / \ s) = \{\}\}\\
&\quad = \{(s, X) \mid s \in \mathit{traces}(P)\\
&\qquad\quad \text{and } X \cap \{x \mid s \frown \langle x \rangle \in \mathit{traces}(P)\} = \{\}\}
\end{aligned}$$

which shows that $\mathit{failures}(P)$ can be defined in terms of $\mathit{traces}(P)$.

So if P and Q are deterministic, and $\mathit{traces}(P) = \mathit{traces}(Q)$, then $\mathit{failures}(P) = \mathit{failures}(Q)$.

Any process defined using just Stop, prefixing, menu choice (or $|$), $\|$ and guarded recursion, is deterministic.

$\diamond$ Failure Refinement $\diamond$

Failure refinement is defined in a similar way to trace refinement.

$$P \sqsubseteq_f Q$$
$$\text{if and only if}$$
$$failures(Q) \subseteq failures(P)$$

It is pronounced "P is failure refined by Q".

To see how failure refinement can be used in specifications, consider a very simple example: the process

$$\mathit{SPEC} = a \to b \to \mathit{SPEC}$$

Recall that if we use *SPEC* as a specification with *trace* refinement, we get a *safety* specification. Processes P satisfying the specification

$$\mathit{SPEC} \sqsubseteq_t P$$

include

$$\begin{aligned} P &= Stop \\ P &= a \to Stop \\ P &= a \to (b \to P \,\Box\, b \to Stop) \\ P &= a \to b \to P \end{aligned}$$

What is the effect of specifying

$$\mathit{SPEC} \sqsubseteq_f P?$$

We need to calculate $failures(\mathit{SPEC})$. In words first: the traces of SPEC are alternating sequences of a and b events, starting with a. After a trace ending in a, SPEC refuses the sets $\varnothing$ and $\{a\}$. After a trace ending in b, it refuses the sets $\varnothing$ and $\{b\}$. So:

$$\begin{array}{rcl} failures(\mathit{SPEC}) & = & \{(\langle a, b\rangle^n \frown \langle a\rangle, \varnothing) \mid n \geqslant 0\} \\ & \cup & \{(\langle a, b\rangle^n \frown \langle a\rangle, \{a\}) \mid n \geqslant 0\} \\ & \cup & \{(\langle a, b\rangle^n, \varnothing) \mid n \geqslant 0\} \\ & \cup & \{(\langle a, b\rangle^n, \{b\}) \mid n \geqslant 0\}. \end{array}$$

To determine whether $\mathit{SPEC} \sqsubseteq_f \mathit{Stop}$ we need to calculate that

$$\begin{array}{rcl} failures(\mathit{Stop}) & = & \{(\langle\rangle, \varnothing), (\langle\rangle, \{a\}), (\langle\rangle, \{b\}), \\ & & (\langle\rangle, \{a, b\})\} \end{array}$$

and then we can see that the failure pairs $(\langle\rangle, \{a\})$ and $(\langle\rangle, \{a, b\})$ are in $failures(\mathit{Stop})$ but not in $failures(\mathit{SPEC})$. Therefore it is not the case that $\mathit{SPEC} \sqsubseteq_f \mathit{Stop}$. We could also write this as

$$\mathit{SPEC} \not\sqsubseteq_f \mathit{Stop}.$$

Now look at $P = a \rightarrow \mathit{Stop}$.

$$\begin{array}{rcl} failures(a \rightarrow \mathit{Stop}) & = & \{(\langle\rangle, \varnothing), (\langle\rangle, \{b\}), (\langle a\rangle, \varnothing), \\ & & (\langle a\rangle, \{a\}), (\langle a\rangle, \{b\}), \\ & & (\langle a\rangle, \{a, b\})\} \end{array}$$

The failure pairs $(\langle a\rangle, \{b\})$ and $(\langle a\rangle, \{a, b\})$ are in $failures(P)$ but not in $failures(\mathit{SPEC})$, so again $\mathit{SPEC} \not\sqsubseteq_f P$.

$\diamond$ Exercise $\diamond$

If we define $P = a \to (b \to P \,\Box\, b \to Stop)$, is it true that $SPEC \sqsubseteq_f P$? Either show that all the failure pairs of P are also failure pairs of $SPEC$, or find a failure pair of P which is not a failure pair of $SPEC$.

$\diamond$ Liveness $\diamond$

$SPEC \sqsubseteq_f P$ is a *liveness* specification which requires P to do certain events. Which definitions of P satisfy the specification? Obviously

$$P = a \to b \to P$$

does, because that is the same process as $SPEC$. In fact this is the only process satisfying this specification. So in this example, the specification is very restrictive indeed: it pins down the implementation precisely.

$\diamond$ Safety and Liveness $\diamond$

Saying that $t \in traces(P)$ is a *positive* statement: it describes something that P can do. A specification of the form

$$\mathit{SPEC} \sqsubseteq_t P$$

puts a limit on the traces that P can do, so it is a specification which restricts behaviour.

Saying that $(t, X) \in failures(P)$ is a *negative* statement: it describes something that P cannot do. A specification of the form

$$\mathit{SPEC} \sqsubseteq_f P$$

puts a limit on what P can fail to do, so it requires P to accept at least a certain range of behaviours.

Alternatively: P fails a safety (trace) specification by doing too much. P fails a liveness (failure) specification by refusing too much, i.e. by not doing enough.

$\diamond$ Another Example $\diamond$

Process P will have alphabet $\{a, b, c\}$, and we want to specify that P must be able to do an infinite sequence of alternating a and b events, starting with a; we do not care when c events occur.

We can use the process

$$\mathit{ALT} = a \rightarrow b \rightarrow \mathit{ALT}$$

as a specification for the a and b events, as before. To allow the c events to occur freely we use hiding, and express the specification as

$$\mathit{ALT} \sqsubseteq_f (P \setminus \{c\})$$

Definitions of P satisfying this specification include

$$\begin{aligned}
P &= a \rightarrow b \rightarrow P \\
P &= c \rightarrow a \rightarrow c \rightarrow c \rightarrow b \rightarrow P \\
P &= a \rightarrow b \rightarrow c \rightarrow P \\
P &= a \rightarrow c \rightarrow b \rightarrow a \rightarrow b \rightarrow P
\end{aligned}$$

because in each case, $P \setminus \{c\}$ is the same process as ALT.

Definitions of P not satisfying the specification include

$$\begin{aligned}
Q &= c \rightarrow b \rightarrow Q \\
P &= a \rightarrow (b \rightarrow P \;\Box\; b \rightarrow Q) \\
P &= a \rightarrow b \rightarrow (P \;\Box\; a \rightarrow c \rightarrow \mathit{Stop}).
\end{aligned}$$

◇ Level Crossing Liveness ◇

In our model of the level crossing, there is an infinite stream of cars trying to cross, and also an infinite stream of trains. We can specify liveness (the requirement that whenever a car approaches it should eventually be allowed to cross, and similarly for the trains) as follows.

$$\begin{aligned} \mathit{CARSPEC} &= \mathit{car.approach} \rightarrow \mathit{car.enter} \rightarrow \\ &\quad \mathit{car.leave} \rightarrow \mathit{CARSPEC} \\ \mathit{TRAINSPEC} &= \mathit{train.approach} \rightarrow \mathit{train.enter} \rightarrow \\ &\quad \mathit{train.leave} \rightarrow \mathit{TRAINSPEC} \end{aligned}$$

The specifications are

$$\mathit{CARSPEC} \sqsubseteq_f (\mathit{SAFE\_SYSTEM} \setminus \{\mathit{train}, \mathit{gate}\})$$

$$\mathit{TRAINSPEC} \sqsubseteq_f (\mathit{SAFE\_SYSTEM} \setminus \{\mathit{car}, \mathit{gate}\})$$

(all the *gate.???* events are hidden, etc.)

These specifications can be checked using FDR.

$\diamond$ Scheduler Liveness $\diamond$

A liveness specification for the cyclic scheduler is that the processes continue to be started, in turn, forever. This can be written

$$CYCLE_0 \sqsubseteq_f (SCHED \setminus \{finish\})$$

where $CYCLE_0$ is the process which was used for the safety specification, and all the *finish.i* events are hidden. This specification can be checked with FDR.

Another liveness specification might be to pick a particular process i and specify that *start.i* and *finish.i* keep happening alternately forever. This can be done with a specification process in which *start.i* and *finish.i* alternate, by hiding all the other *start* and *finish* events in *SCHED*.