Understanding Concurrent Systems: errata

Bill Roscoe

February 25, 2013

This document contains all mistakes and typos etc that might lead to confusion known to the author up to the above date. They are arranged by page number.

Please let me know if you find other issues that should be listed here.

p 69 In the definition of `nhd((i,j))`, `(i-1)/a` should be `i/a` and `(j-1)/b` should be `j/b`.

The definition of `Symbol` is `{1..9}`.

The final `EmptyM` on this page is missing parameter `p`.

p 70 At the end of the fourth program line `EmptyB{p}` should be `EmptyB(p)`.

There is a mistaken double use of the identifier `v` in the defnition of `FullB`, which should be

```
FullB(p,v) = select?q:adj(p)?w:diff(Symbol,{v}) ->FullB(p,v)
             [] done -> FullB(p,v)
```

p 85/6 There are typos in the first line of the definition of $I_{i,j}$ and on the first line of p86 (in the definition of $O'_{i,j}((x,y),p)$).

In the first case the resulting process should be $I'_{i,j}((x,y),p)$.

In the second case the communication should be $up.(i,j-1).((x,y),p)$.

p 110 In the statement of BL1, $[b \Rightarrow c]$ should read $[b \leftrightarrow c]$.

p 111 The definitions of *Mu* and *Md* should each use the same identifier for input and

output:

$$
\begin{array}{rcl}
Mu & = & in.end \rightarrow downto.end \rightarrow upto.end \rightarrow O1 \\
 & & \Box\ in?x : T \rightarrow upto!x \rightarrow Md \\
Md & = & in.end \rightarrow downto.end \rightarrow upto.end \rightarrow O1 \\
 & & \Box\ in?x : T \rightarrow downto!x \rightarrow Mu \\
O1 & = & upfrom?x \rightarrow downfrom?y \rightarrow O2(x,y) \\
O2(x,y) & = & x = end \wedge y = end \,\&\, out!end \rightarrow Mu \\
 & & x = end \wedge y \neq end \,\&\, out!y \rightarrow downin?y' \rightarrow O2(x,y') \\
 & & x \neq end \wedge y = end \,\&\, out!x \rightarrow upin?x' \rightarrow O2(x',y) \\
 & & x \neq end \wedge y \neq end \,\&\, (out!x \rightarrow upin?x' \rightarrow O2(x',y) \\
 & & \qquad\qquad \lhd x < y \rhd \\
 & & \qquad\qquad out!y \rightarrow downin?y' \rightarrow O2(x,y'))
\end{array}
$$

p 112 Exercise 5.15. There is a mysterious reference to a process Q at the end of this exercise. The origin of this is that the Mergesort example on the previous page was a late substitution for Quicksort that appeared in initial drafts. The Q referred to here is the process corresponding to M from that Quicksort, and I did not notice the hanging reference when I changed the example.

This question makes perfect sense as an exercise in creating a recursive enslaved system without reference to the now absent example, or the reader can adapt the coding of Quicksort in Chapter 4 of TPC.

p 113 `selother` is not used correctly in the definition of `EM2(x)`. Should be

```
card(X) > 0 & (selhere?v:X -> FM2
            [] seladj?v -> EM2(diff(X,{v}))
            [] selother?v -> EM2(X)
```

p 131 At the bottom of the page, the $\langle P;\ Q\rangle$ should read

$$traces(P;\ Q) \quad = \quad (traces(P) \cap \Sigma^*) \cup \{s \hat{} t \mid s \hat{} \langle \checkmark \rangle \in \langle P \rangle, t \in traces(Q)\}$$

(i.e. replace P by $traces(P)$ in one place.

p 140 There is a condition missing in the definition of $traces(P \Theta_A Q)$, which should read:

$$
\begin{array}{rcl}
traces(P \Theta_A Q) & = & \{s \in traces(P) \mid s \in (\Sigma \setminus A)^{*\checkmark}\} \\
 & & \cup \{s \hat{} \langle a \rangle \hat{} t \mid s \in (\Sigma \setminus A)^* \wedge \\
 & & \qquad s \hat{} \langle a \rangle \in traces(P) \wedge a \in A \wedge t \in traces(Q)\}
\end{array}
$$

p 172 The process name `RHSDet(P)` should be replaced by `RHS(P)`, and similarly `DetSpec` should be replaced by `LHS`

p 239 The definition of $failures(P \triangle Q)$ does not handle the termination ($\surd$) of P correctly. It should read:

$$\begin{array}{lcl} failures(P \triangle Q) & = & \{(s,X) \mid (s,X) \in failures(P) \wedge (\langle\rangle, X) \in failures(Q)\} \\ & & \cup \{(s\hat{}t, X) \mid s \in traces(P) \cap \Sigma^* \wedge (t,X) \in failures(Q) \wedge t \neq \langle\rangle\} \\ & & \cup \{(s,X), (s\hat{}\langle\surd\rangle, X) \mid s\hat{}\langle\surd\rangle \in traces(P)\} \end{array}$$

p 435 Chapters 18 and 19:

There are a few cases where the quoted code in these chapters is not, despite what it says on page 435, SVL in the sense that it would be accepted by the SVA parser. This happened because the book was written before the latest version of the front end. There are cases where `signal` has been written out in full rather than `sig`, and ones where the boolean `and` and `or` operators have been written like that rather than the correct `&&` and `||`. On p 466 the wrong comment notation is used:`--` rather than `//`.

Naturally the example files supplied with SVA for these chapters are all in the correct syntax, because they parse and run. The SVA manual is the definitive guide to syntax.

p 462 Despite what it says in the second bullet point, `A` does not refine the other three. This is because it does not change the value `number[i]` until the computation is complete, and so the initial value `0` of this slot can be seen in contexts impossible in the others. Running the tool on the example file will show this.

The result given was obtained on another version of this check, but not the correct one explained in the book. The result quoted on p464 is correct (though inconsistent with p462).

p 488/9 Despite what is claimed here it is not possible to give an accurate definition of $\mathbf{Pri}_{\leq}$ in $\mathscr{RT}$ for general $\leq$.

The refusal testing value of a process P can tell us what traces are possible for $\mathbf{Pri}_{\leq}(P)$: P can only perform an action a that is not maximal in $\leq$ when all greater actions (including τ) are impossible. In other words the trace $\langle a_1, \ldots, a_n\rangle$ is possible for $\mathbf{Pri}_{\leq}(P)$ if and only if

$$\langle X_0, a_1, x_1, \ldots, X_{n-1}, a_n, \bullet\rangle$$

is a refusal testing behaviour, where X_i is $\bullet$ if a_{i-1} is maximal, and $\{a \in \Sigma \mid a > a_{i-1}\}$ if not (even if that set is empty so a_{n-1} is less than only τ and $\surd$).

We have now discovered that sometimes the refusal components of refusal testing behaviours of $\mathbf{Pri}_{\leq}(P)$ can not be computed accurately from the corresponding behaviour of P. This is because $\mathbf{Pri}_{\leq}(P)$ can refuse larger sets than P: notice that if P offers *all* visible events, then the prioritised process refuses all that are not maximal in $\leq$.

Consider the processes

$$DF1(X) = \bigsqcap \{a \rightarrow DF1(X) \mid a \in X\}$$

$$DF2(X) = \bigsqcap \{?x : A \rightarrow DF1(X) \mid A \subseteq X, A \neq \emptyset\}$$

These are equivalent in the refusal testing models: each has all possible behaviours with traces in Σ^* that never refuse the whole alphabet Σ.

Now consider $P1 = DF1(\{a,b\}) \,|||\, CS$ and $P2 = DF2(\{a,b\}) \,|||\, CS$ where $CS = c \rightarrow CS$. Clearly these are also refusal testing equivalent. Now suppose $\leq$ is the order in which $b > c$ and a is incomparable to each of b and c. We ask the question: is $\langle \{c\}, a, \bullet \rangle$ a refusal testing behaviour of $\mathbf{Pri}_{\leq}(Pi)$?

When $i = 1$ the answer is "no", since whenever $P1$ performs the event a the set of events it offers is precisely $\{a,c\}$ (it can also offer $\{b,c\}$). On the other hand, $P2$ can choose to offer $\{a,b,c\}$: in this state the priority operator prevents c from being offered to the outside, meaning that $\mathbf{Pri}_{\leq}(P2)$ can be in a stable state where a is possible but c is not: so in this case the answer is "yes". This demonstrates that we need more information than refusal testing of Pi to calculate the refusal testing behaviours of $\mathbf{Pri}_{\leq}(Pi)$.

On close inspection this example tells us that $\mathbf{Pri}_{\leq}(\cdot)$ is only compositional for refusal testing when the structure of $\leq$ is such that whenever a and b are incomparable events in Σ and $c < b$ then also $c < a$. This means that the order has to take one of two forms:

- A linearly ordered list of collections of equally prioritised events, the first of which contains $\{\tau, \surd\}$.
- A linearly ordered list of collections of equally prioritised events, the first of which is exactly $\{\tau, \surd\}$, together with a further collection of events that are incomparable to the members of the first two of these collections and greater than the rest.

The second of these includes the order used for the timed priority model (see Chapter 14), in which the only prioritisation is that $\{\tau, \surd\}$ have greater priority than the time event(s), typically $\{tock\}$. One can[1] give a compositional definition over refusal testing models in these circumstances.

These issues disappear for the acceptance traces model $\mathscr{FL}$ and its variants, which are therefore the *only* CSP models with respect to which our priority operator can be defined in general.

With respect to this model, the semantics of $\mathbf{Pri}_{\leq}(P)$ are the behaviours

$$\{\langle A_0, a_1, A_1, \ldots, A_{n-1}, a_n, A_n \rangle \mid \langle Z_0, a_1, Z_1, \ldots, Z_{n-1}, a_n, Z_n \rangle \in P\} \cup$$
$$\{\langle A_0, a_1, A_1, \ldots, A_{n-1}, a_n, \bullet, \surd \rangle \mid \langle Z_0, a_1, Z_1, \ldots, Z_{n-1}, a_n, \bullet, \surd \rangle \in P\}$$

where in every case one of the following holds:

[1] The principles that apply here are, that when calculating the refusal testing behaviours of $\mathbf{Pri}_{\leq}(P)$ for such a $\leq$, the behaviours of P considered cannot have $\bullet$ refusals unless the following event is incomparable with $\{\tau, \surd\}$, and if the following event is not maximal amongst Σ under $\leq$, a refusal set must contain all greater members of Σ. If an event a follows which is not minimal, then non-$\bullet$ refusals X in P can give rise to one where any subset of $\{b \in \Sigma \mid b < a\}$ is added to X. The conditions on $\leq$ have the effect that any other events that might be simultaneously avaliable to a in $\mathbf{Pri}_{\leq}(P)$ do not cause any further additional refusals that a does not. Adding additional members of refusals based on the subsequent action means that, to calculate the final refusals in behaviours of length in $\mathbf{Pri}_{\leq}(P)$, you need to examine those with an additional event on the end in P.

- a_i is maximal under $\leq$ and $A_i = \bullet$ (so there is no condition on Z_i except that it exists).
- a_i is not maximal under $\leq$ and $A_i = \bullet$ and Z_i is not $\bullet$ and neither does Z_i contain any $b > a_i$.
- Neither A_i nor Z_i is $\bullet$, and $A_i = \{a \in Z_i \mid \neg \exists b \in Z_i . b > a\}$,
- and in each case where $A_{i-1} \neq \bullet$, $a_i \in A_{i-1}$.

Notice how we are able, when P offers the set Z, to calculate the set that $\mathbf{Pri}_{\leq}(P)$ offers: the set $\{a \in Z \mid \neg \exists b \in Z . b > a\}$.