Concurrent and Real Time Systems
Sample solutions to exercises

This document contains the generally available sample solutions to selected exercises from the book ‘Concurrent and Real Time Systems: the CSP approach’ by Steve Schneider.

Answers to the other questions have restricted access.

Steve Schneider
S.Schneider@rhbnc.ac.uk
May 1999

Chapter 1

Question 1.5*

Question 1.10*

\[
\begin{align*}
PRESS &= PRESS(0) \\
PRESS(n) &= \text{press} \rightarrow \text{PRESS}(n + 1) \\
& \quad | \text{finish}_n \rightarrow \text{STOP}
\end{align*}
\]
Question 1.13*

\[
TROLLEY = \begin{align*}
    &\text{choc} \to \text{apple} \to \text{cake} \to \text{cake} \to \text{STOP} \\
    &\quad \mid \text{cake} \to \text{apple} \to \text{cake} \to \text{STOP} \\
    &\quad \mid \text{cake} \to \text{apple} \to \text{STOP} \\
    &\mid \text{apple} \to \text{choc} \to \text{cake} \to \text{cake} \to \text{STOP} \\
    &\quad \mid \text{cake} \to \text{choc} \to \text{cake} \to \text{STOP} \\
    &\quad \mid \text{cake} \to \text{choc} \to \text{STOP} \\
    &\mid \text{cake} \to \text{apple} \to \text{choc} \to \text{cake} \to \text{STOP} \\
    &\quad \mid \text{cake} \to \text{choc} \to \text{STOP} \\
    &\mid \text{choc} \to \text{apple} \to \text{cake} \to \text{STOP} \\
    &\quad \mid \text{cake} \to \text{apple} \to \text{STOP} \\
    &\mid \text{cake} \to \text{choc} \to \text{STOP} \\
    &\mid \text{choc} \to \text{apple} \to \text{STOP} \\
\end{align*}
\]
Chapter 2

Question 2.3*

Deadlock can occur whenever one of the girls has the easel, and the other has the box.

Question 2.5*

\[ \text{DCUSTOMER} \parallel \text{SHOP} = \text{enter} \rightarrow \text{select} \rightarrow \text{pay} \rightarrow \text{leave} \rightarrow \text{DCUSTOMER} \parallel \text{SHOP} \]

The SECURITY component of the SHOP process prevents leave from occurring at the point DCUSTOMER is first ready for it.

If the external choice of DCUSTOMER is replaced by an internal choice (resulting in DCUSOMER', say), then the resulting combination may deadlock: the customer might choose to leave without paying, but is prevented from doing so by the process SHOP.

\[ \text{DCUSTOMER'} \parallel \text{SHOP} = \text{enter} \rightarrow \text{select} \rightarrow \]
\[ (\text{STOP} \sqcap \text{pay} \rightarrow \text{leave} \rightarrow \text{DCUSTOMER'} \parallel \text{SHOP}) \]
Question 2.9*

The following gives a counterexample to the claim that interface parallel (with different interfaces) is associative.

\[
\begin{align*}
P_1 &= \text{STOP} \\
P_2 &= \text{STOP} \\
P_3 &= a \rightarrow \text{STOP} \\
A &= \{a\} \\
B &= \{} \\
\bigtriangledown_A (P_2 \bigpar_B P_3) &= \text{STOP} \\
(P_1 \bigpar_A P_2) \bigpar_B P_3 &= a \rightarrow \text{STOP}
\end{align*}
\]

In the first case, \( P_1 \) exercises a veto on all events from \( A \). In the second, \( P_1 \) exercises that veto within the combination with \( P_2 \), but cannot prevent \( P_3 \) from performing events in \( A \setminus B \).

Chapter 3

Question 3.2*

\[
C = h_{in}?x \rightarrow v_{in}?y \rightarrow v_{out}!\min\{x, y\} \rightarrow h_{out}!\max\{x, y\} \rightarrow C
\]

If \( i < j \) then the function \( f_{i,j} \) is defined by

\[
\begin{align*}
f_{i,j}(h_{in}) &= h_{i,j} \\
f_{i,j}(v_{in}) &= v_{i,j} \\
f_{i,j}(h_{out}) &= h_{i+1,j} \\
f_{i,j}(v_{out}) &= v_{i,j+1}
\end{align*}
\]

and if \( i = j \) then the function \( f_{i,i} \) is defined by

\[
\begin{align*}
f_{i,i}(h_{in}) &= h_{i,i} \\
f_{i,i}(v_{in}) &= v_{i,i} \\
f_{i,i}(h_{out}) &= v_{i+1,i+1} \\
f_{i,i}(v_{out}) &= v_{i,j+1}
\end{align*}
\]

Finally, \( \text{SORTER} \) is defined as follows:

\[
\text{SORTER} = \bigpar_{0 \leq i \leq j \leq n} f_{i,j}(C)
\]

4
Question 3.6*

\[ \text{traces}(P \sqcap \text{RUN}) = \text{traces}(P) \cup \text{traces}(	ext{RUN}) \]
\[ = \text{traces}(P) \cup \text{TRACE} \]
\[ = \text{TRACE} \]
\[ = \text{traces}(	ext{RUN}) \]

Chapter 4

Question 4.4*

1. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{ticket}, \checkmark \rangle \} 

2. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{change}, \text{ticket} \rangle , \langle \text{coin}, \text{change}, \text{ticket}, \checkmark \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{ticket}, \text{change} \rangle , \langle \text{coin}, \text{ticket}, \text{change}, \checkmark \rangle \} 

3. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{coin} \rangle , \langle \text{coin}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{ticket}, \text{coin} \rangle , \langle \text{coin}, \text{ticket}, \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{ticket}, \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{change}, \text{coin} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket}, \checkmark \rangle \} 

4. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{ticket}, \checkmark \rangle , \langle \text{coin}, \text{coin} \rangle , \langle \text{coin}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{coin}, \text{ticket}, \checkmark \rangle , \langle \text{coin}, \text{change}, \text{coin} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket}, \checkmark \rangle \} 

Question 4.8*

1. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{ticket}, \checkmark \rangle \} 

2. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{change}, \text{ticket} \rangle , \langle \text{coin}, \text{change}, \text{ticket}, \checkmark \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{ticket}, \text{change} \rangle , \langle \text{coin}, \text{ticket}, \text{change}, \checkmark \rangle \} 

3. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{coin} \rangle , \langle \text{coin}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{ticket}, \text{coin} \rangle , \langle \text{coin}, \text{ticket}, \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{ticket}, \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{change}, \text{coin} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket}, \checkmark \rangle \} 

4. \{ \langle \_ \rangle , \langle \text{coin} \rangle , \langle \text{coin}, \text{change} \rangle , \langle \text{coin}, \text{change}, \checkmark \rangle , \langle \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{ticket}, \checkmark \rangle , \langle \text{coin}, \text{coin} \rangle , \langle \text{coin}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{coin}, \text{ticket}, \checkmark \rangle , \langle \text{coin}, \text{change}, \text{coin} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket} \rangle , \langle \text{coin}, \text{change}, \text{coin}, \text{ticket}, \checkmark \rangle \} 

5
Question 4.12*

\[ \text{traces}(P) = \{ (up)^n \mid n \in \mathbb{N} \} \]
\[ \cup \{ (up)^n \land (down)^m \mid n \in \mathbb{N} \land m \in \mathbb{N} \land m \leq n \} \]
\[ \cup \{ (up)^n \land (down)^n \land \langle \rangle \mid n \in \mathbb{N} \} \]

Question 4.14*

Yes, \( F \) is guarded. \( \text{traces}(N = F(N)) = \{ a, b \} \). There are no other fixed points: guardedness means that there is a unique fixed point.

Question 4.18*

\[(SPY \parallel MASTER) \setminus \text{relay} = \text{listen}\? x : T \rightarrow (SPY \parallel \log! x \rightarrow MASTER) \setminus \text{relay}\]

where

\[(SPY \parallel \log! x \rightarrow MASTER) \setminus \text{relay} = \text{listen}\? y \rightarrow \log! x \rightarrow (SPY \parallel \log! y \rightarrow MASTER) \setminus \text{relay} \]
\[\mid \log! x \rightarrow \text{listen}\? y \rightarrow (SPY \parallel \log! y \rightarrow MASTER) \setminus \text{relay} \]

Thus \((SPY \parallel \log! y \rightarrow MASTER) \setminus \text{relay}\) is a fixed point of the guarded mutual recursive equation defining the \textit{RECORD} processes, and hence must be equal (since the fixed point is unique).

Chapter 5

Question 5.2*

The specification is

\[ tr = tr_0 \land \langle \text{raw} \rangle \land tr_1 \land \langle \text{cooked} \rangle \land tr_2 \Rightarrow \text{wash in } tr_1 \]

The combination \( RAW \parallel \text{wash} \) \( \textit{COOKED} \) does not meet this specification, since it allows the trace \( \langle \text{raw}, \text{wash}, \text{raw}, \text{cooked} \rangle \).

Question 5.7*

The first two proof rules are sound.

The third proof rule is not sound. The processes \( P_1 \) and \( P_2 \) of Exercise 5.6 provide a counterexample, since \( P_1 \parallel P_2 \) has \( \langle b, a \rangle \) as a possible trace, and this does not satisfy \( tr \downarrow a \leq tr \downarrow c \).
Question 5.9*
It is necessary to prove that the recursive function

\[ F(Y) = (\text{open} \rightarrow \text{close} \rightarrow Y \ \square \ \text{locked} \rightarrow \text{STOP}) \]

preserves the specification, and that the specification is satisfiable. These proofs can be carried out by applying the rules for prefix and for external choice.

1. Prove that \( F \) preserves \( \text{foot(tr)} = \text{open} \Rightarrow \text{foot(init(tr)} \neq \text{open} \)

2. The specification that \( \text{close} \) does not appear twice consecutively is not preserved by \( F \) (consider \( F(\text{close} \rightarrow \text{STOP}) \)). It is necessary to find a stronger specification that is preserved by \( F \). One that suffices is:

\[(\text{foot(tr)} = \text{close} \Rightarrow \text{foot(init(tr)} \neq \text{close}) \wedge \text{head(tr)} \neq \text{close}\]

3. This specification is preserved by \( F \).

Chapter 6

Question 6.7*

\[ \text{DIV} \Downarrow \text{DIV} \]

Question 6.8*

\[ \text{CHAOS} \Downarrow \text{CHAOS} \quad [a \in \Sigma] \]

\[ \text{CHAOS} \Downarrow \text{STOP} \]

\[ \text{CHAOS} \not\Downarrow \text{STOP} \]

Chapter 7

Question 7.3*

Both \( P_1 \) and \( P_2 \) satisfy \( X \neq \Sigma \). Any refusal \( X \) of \( P_1 \ || \ P_2 \) will be made up of a refusal \( X_1 \) of \( P_1 \), and \( X_2 \) of \( P_2 \), such that \( X \setminus \{a\} = X_1 \setminus \{a\} \cap X_2 \setminus \{a\} \), and \( X \cap \{a\} = (X_1 \cup X_2) \cap \{a\} \).

If \( a \in X_1 \) then \( \exists b \not\in X_1, b \neq a \) and so \( b \not\in X \). Thus \( X \neq \Sigma \). Similarly if \( a \in X_2 \) then \( X \neq \Sigma \).

Finally, if \( a \not\in X_1 \) and \( a \not\in X_2 \), then \( a \not\in X \), and so again \( X \neq \Sigma \).
In every case, $X \neq \Sigma$, and so the process is strongly deadlock-free.
If the interface contains more than one event, then strong deadlock-freedom need not be preserved. For example, if $P_1 = a \rightarrow P_1$ and $P_2 = b \rightarrow P_2$, then both processes are strongly deadlock-free, but $P_1 \parallel_{\{a, b\}} P_2$ deadlocks.

**Question 7.5**

The assertion

$$\forall Y \bullet (SPEC \subseteqSF Y \Rightarrow SPEC \subseteqSF F(Y))$$

implies the assertion that $SPEC \subseteqSF F(SPEC)$, since $SPEC$ is one possible instantiation of $Y$. Thus we have to prove that $SPEC \subseteqSF F(SPEC)$ means that $SPEC \subseteqSF F(Y)$ for any refinement $Y$ of $SPEC$.

By monotonicity of $F$ (made up of CSP operators, which are all monotonic),

$$SPEC \subseteqSF Y \Rightarrow F(SPEC) \subseteqSF F(Y)$$

$$\Rightarrow SPEC \subseteqSF F(Y)$$

by transitivity of refinement.

**Question 7.8**

1.

$$STACK(\emptyset) = push?x : T \rightarrow STACK(\langle x \rangle)$$

$$STACK(\langle x \rangle \cap s) = pop!x \rightarrow STACK(s)$$

$$\square (STOP \sqcap push?y : T \rightarrow STACK(\langle y, x \rangle \cap s))$$

2. By resolving the internal choice of the specification in favour of $STOP$, we obtain

$$STACK_1(\emptyset) = push?x : T \rightarrow STACK(\langle x \rangle)$$

$$STACK_1(\langle x \rangle) = pop!x \rightarrow STACK(\emptyset)$$

and $STACK_1(\emptyset)$ is a refinement of $STACK(\emptyset)$, and hence is a stack.

3. By resolving the internal choice against $STOP$ for singleton sequences, and in favour of $STOP$ for sequences of length 2, the following refinement of $STACK$ is obtained:

$$STACK_2(\emptyset) = push?x : T \rightarrow STACK(\langle x \rangle)$$

$$STACK_2(\langle x \rangle) = pop!x \rightarrow STACK(\emptyset)$$

$$\square push?y : T \rightarrow STACK(\langle y, x \rangle)$$

$$STACK_2(\langle y, x \rangle) = pop!x \rightarrow STACK(\langle x \rangle)$$

Thus $STACK_2(\emptyset)$ describes a stack, since it is a refinement of $STACK(\emptyset)$.  

8
Chapter 8

Question 8.5*

1. \( P_1 = a \rightarrow \text{STOP} \quad \square b \rightarrow \text{STOP} \)
2. Yes: \( P_2 = (a \rightarrow b \rightarrow \text{STOP}) \cap (b \rightarrow a \rightarrow \text{STOP}) \)
3. \( P_2 \) above has \( P \| P \neq_{FDI} P \)
4. Define \( AS = a \rightarrow AS \) and \( BS = b \rightarrow BS \). Then
   \[
   \begin{align*}
   P_1 &= AS \| (STOP \cap b \rightarrow STOP) \\
   P_2 &= BS \| (STOP \cap a \rightarrow STOP)
   \end{align*}
   \]
   \( P_1 \) and \( P_2 \) are both nondeterministic, but their interleaved combination is deterministic.
5. No, if \( P \) is nondeterministic then so too is \( P \| P \).
6. The deterministic process \( P = a \rightarrow a \rightarrow b \rightarrow \text{STOP} \) has \( P \| P \) nondeterministic: after the trace \( \langle a, a \rangle \), the \( b \) might be performed or refused.

Question 8.6*

1. Any divergence must at some point have given out more chocolates than received coins:
   \[
   S_D(tr) = \exists tr' \leq tr. tr' \downarrow \text{choc} > tr' \downarrow \text{coin}
   \]
2. \( S_D(tr) = \exists tr_0, tr_1. tr = tr_0 \cap (in, in, in) \cap tr_1 \)
3. \( S_D(tr) = \exists n \geq 2^{32} in. n \mid tr \)
4. This is a specification on the infinite behaviour—that in the limit there should be at least two outputs for every three inputs. This may be expressed as follows:
   \[
   S_I(u) = \exists N. \forall tr < u. (\# tr > N \Rightarrow (tr \downarrow out/tr \downarrow in) \geq \frac{2}{3})
   \]
5. \( S_I(u) = tr \uparrow \{a\} \neq \emptyset \)
6. Assuming all requests are unique (so any request can appear at most once in the trace),
   the requirement can be expressed as follows:
   \[
   S_I(u) = (req, inu) \Rightarrow (req, service, i) \not\leq u
   \]
7. For every execution to eventually terminate, the process must be deadlock-free and have no infinite traces. This is a condition upon the failures and the infinite traces of the process:
   \[
   \begin{align*}
   S_F(tr, X) &= \checkmark \not\in \sigma(tr) \Rightarrow X \neq \Sigma' \\
   S_I(u) &= \text{false}
   \end{align*}
   \]
   \( S_I \) here states that no infinite trace is possible—that the process should not have any infinite traces.

9
Chapter 9

Question 9.5*

\[ \text{CON1} = \ \text{in\_kate} \rightarrow \text{out} \rightarrow \text{CON1} \]
\[ \ | \ \text{in\_eleanor} \rightarrow \text{out} \rightarrow \text{CON1} \]

Question 9.6*

\[ \text{CON2} = (\text{in\_kate} \rightarrow (\text{out} \rightarrow \text{RUN}_{\text{EKO}} \uplus \text{RUN}_{\text{EKO}})) \triangle \text{CON2} \]
\[ \ | \ \text{in\_eleanor} \rightarrow \text{out} \rightarrow \text{CON2} \]
\[ \ | \ \text{out} \rightarrow \text{CON2} \]

Chapter 10

Question 10.2*

\[ N = n : \mathbb{N}^{2n} \] has zero executions but no spin executions.

Question 10.3*

If \( N(i) = a \xrightarrow{t} N(i+1) \), then \( N(0) \) has no infinitely fast executions—all infinite executions take infinitely long to unwind. However, the \( N(i) \) are not \( t \)-guaranteed for any \( t \).

Chapter 11

Question 11.1*

1. Yes
2. Yes
3. No. This is a single timed event.
4. No. It does not have the correct structure, since it is not half-open at time 7.
5. Yes
6. No. This is a set of timed events, but it does not have the structure of a refusal set.
Question 11.2*

1. No: \( a \) is not refused over the interval \([0, 3)\)
2. Yes: \( a \) is refused over the interval \([3, 6)\)
3. No: \( b \) is not refused over the entire interval \([3, 6)\) (though it is refused for part of it)
4. Yes: both \( a \) and \( b \) are refused over \([3, 5)\)
5. Yes: \( a \) is refused after time 7 for ever
6. No: \( b \) is not refused after time 7
7. Yes: the empty refusal (no refusal observed) is consistent with any execution.

Question 11.9*

1. Yes, this is a valid refinement. Any behaviour of \((a \xrightarrow{2} STOP) \div^2 STOP\) is either a behaviour of \(STOP\) (if the trace is empty), or a behaviour of \(a \rightarrow STOP\) (if the trace contains \(a\)—since the \(a\) cannot be refused before it occurs).
2. No, this is not a valid refinement. The failure \(((1, a)], [0, 1) \times \{a\}\) is possible for the right hand side, but not for the left.

Question 11.13*

The refusal of \(b\) followed by the performance of \(c\) can no longer be detected by the resulting test

\[ (STOP \sqcap b \rightarrow SUCCESS) \Box (c \rightarrow STOP \Box a \rightarrow SUCCESS) \]

In fact, the resulting test cannot distinguish between \(Q_1\) and \(Q_2\), since it has lost the ability to do refusal testing.

Chapter 12

Question 12.2*

\( shower \) at \( t \) \( \Rightarrow \) \( eat \) live from \( t + 10 \) until \( \{eat\} \)

It is straightforward to show that

\( leave \rightarrow SKIP \) \( sat \) \( no\ shower \)

So the rule for prefix yields that

\( eat \xrightarrow{5} leave \rightarrow SKIP \) \( sat \) \( eat \) live from 0 until \( \{eat\} \)

\( \land no\ shower \)
so a further application of the rule for prefix yields that

\[
\begin{align*}
\text{shower} & \rightarrow \text{eat} \rightarrow \text{leave} \rightarrow \text{SKIP} & \text{sat} & s \neq \emptyset \Rightarrow \\
& \quad \text{shower at } \text{begin}(s) \land \text{eat live from } \text{begin}(s) + 10 \text{ until } \{\text{eat}\} \\
& \quad \land \neg \text{shower}[\text{tail}(s)/s]
\end{align*}
\]

which implies

\[
\begin{align*}
\text{shower} \rightarrow \text{eat} \rightarrow \text{leave} \rightarrow \text{SKIP} & \text{ sat } & \text{shower at } t \Rightarrow & \text{eat live from } t + 10 \text{ until } \{\text{eat}\}
\end{align*}
\]

A final application of the rule for prefix, with some judicious simplification, yields again that

\[
\begin{align*}
\text{wake} \rightarrow \text{shower} \rightarrow \text{eat} \rightarrow \text{leave} \rightarrow \text{SKIP} & \text{ sat } & \text{shower at } t \Rightarrow & \text{eat live from } t + 10 \text{ until } \{\text{eat}\}
\end{align*}
\]

as required.

**Question 12.6***

\[
\forall t . s \uparrow \{t\} \downarrow \Sigma \leq 1
\]

The process TSWITCH does meet this specification. This can be established by a recursion induction.

**Chapter 13**

**Question 13.4***

\[
\begin{align*}
\text{a} & \rightarrow \text{b} \rightarrow \text{STOP} \parallel \text{c} \rightarrow \text{STOP} & \equiv_{T} & \text{c} \rightarrow \text{a} \rightarrow \text{b} \rightarrow \text{STOP} \\
& \mid \text{a} \rightarrow (\text{b} \rightarrow \text{c} \rightarrow \text{STOP} & \mid \text{c} \rightarrow \text{b} \rightarrow \text{STOP}) & \subseteq_{T} & \text{a} \rightarrow (\text{b} \rightarrow \text{STOP} \circ \text{c} \rightarrow \text{STOP}) \\
& & & \subseteq_{TP} & \text{a} \xrightarrow{2} (\text{b} \rightarrow \text{STOP} \circ \text{c} \rightarrow \text{STOP})
\end{align*}
\]

The processes are not related by a failures timewise refinement. For example, the timed process can refuse \(c\) for ever after a trace with \(a\) and \(b\), whereas the untimed process cannot.
Appendix A

Question A.3*

\[ \Psi(\text{PRINT} = \text{mid} ? y \xrightarrow{\text{print}} y \rightarrow \text{PRINT}) = \]
\[ = \text{PRINT0} \]
\[ \text{PRINT0} = \text{mid} ? y \rightarrow \text{tok} \rightarrow \text{PRINT1}(y) \]
\[ \quad \square \text{tok} \rightarrow \text{PRINT0} \]
\[ \text{PRINT1}(y) = \text{print} ! y \rightarrow \text{PRINT0} \]
\[ \quad \square \text{tok} \rightarrow \text{PRINT1}(y) \]

Question A.4*

With \text{PRINT0} as given in the previous question, and \text{SPOOL0} as follows:

\[ \text{SPOOL0} = \text{in} ? x \rightarrow \text{tok} \rightarrow \text{tok} \rightarrow \text{tok} \rightarrow \text{SPOOL1}(x) \]
\[ \quad \square \text{tok} \rightarrow \text{SPOOL0} \]
\[ \text{SPOOL1}(x) = \text{mid} ! x \rightarrow \text{SPOOL0} \]
\[ \quad \square \text{tok} \rightarrow \text{SPOOL1} \]

we have to reduce

\[ \Psi(\text{PRINTER}) = (\text{SPOOL0} \mid_{\text{mid} \cdot T} \text{PRINT0}) \setminus_{\text{tok}} \text{mid} \cdot T \]
\[ = \text{in} ? x \rightarrow \text{tok} \rightarrow \text{tok} \rightarrow \text{tok} \rightarrow \text{SPOOL1}(x) \mid_{\text{mid} \cdot T} \text{PRINT0} \setminus_{\text{tok}} \text{mid} \cdot T \]
\[ \quad \square \text{tok} \rightarrow (\text{SPOOL0} \mid_{\text{mid} \cdot T} \text{PRINT0}) \setminus_{\text{tok}} \text{mid} \cdot T \]

\[ (\text{SPOOL1}(x) \mid_{\text{mid} \cdot T} \text{PRINT0}) \setminus_{\text{tok}} \text{mid} \cdot T = (\text{SPOOL0} \mid_{\text{mid} \cdot T} \text{PRINT1}(x)) \setminus_{\text{tok}} \text{mid} \cdot T \]

\[ (\text{SPOOL0} \mid_{\text{mid} \cdot T} \text{PRINT1}(x)) \setminus_{\text{tok}} \text{mid} \cdot T = \]
\[ \text{out} ! x \rightarrow (\text{SPOOL0} \mid_{\text{mid} \cdot T} \text{PRINT0}) \setminus_{\text{tok}} \text{mid} \cdot T \]
\[ \quad \square \text{tok} \rightarrow (\text{SPOOL0} \mid_{\text{mid} \cdot T} \text{PRINT1}(x)) \setminus_{\text{tok}} \text{mid} \cdot T \]
\[ \quad \square \text{in} ? y \rightarrow \text{out} ! x \rightarrow \text{tok} \rightarrow \text{tok} \rightarrow \text{tok} \rightarrow \text{SPOOL1}(x) \mid_{\text{mid} \cdot T} \text{PRINT0} \setminus_{\text{tok}} \text{mid} \cdot T \]
\[ \quad \square \text{tok} \rightarrow \text{out} ! x \rightarrow \text{tok} \rightarrow \text{tok} \rightarrow \text{SPOOL1}(x) \mid_{\text{mid} \cdot T} \text{PRINT0} \setminus_{\text{tok}} \text{mid} \cdot T \]
\[ \quad \square \text{tok} \rightarrow \text{out} ! x \rightarrow \text{tok} \rightarrow \text{SPOOL1}(x) \mid_{\text{mid} \cdot T} \text{PRINT1}(y) \setminus_{\text{tok}} \text{mid} \cdot T \]
\[(SPOOL(x) \parallel PRINT(y)) \downarrow_{\text{lock mid.} T =}
\]
\[tack \to (SPOOL(x) \parallel PRINT(y)) \downarrow_{\text{lock mid.} T}
\]
\[\Box out!y \to (SPOOL(x) \parallel PRINT) \downarrow_{\text{lock mid.} T}
\]

This has given \(\Psi(PRINTER)\) as the fixed point of a function (on a family of processes) which does not contain parallelism or hiding:

\[
\Psi(PRINTER) = PR00
\]
\[
= \text{in}?x \rightarrow tack \rightarrow tack \rightarrow tack \rightarrow PR01(x)
\]
\[\Box tack \rightarrow PR00
\]
\[
PR01(x) = out!x \rightarrow PR00
\]
\[\Box tack \rightarrow PR01(x)
\]
\[\Box in?y \rightarrow out!x \rightarrow tack \rightarrow tack \rightarrow tack \rightarrow PR01(x)
\]
\[\Box tack \rightarrow out!x \rightarrow tack \rightarrow tack \rightarrow PR01(x)
\]
\[\Box tack \rightarrow out!x \rightarrow tack \rightarrow PR01(x)
\]
\[\Box tack \rightarrow PR11(x, y)
\]
\[
PR11(x, y) = tack \rightarrow PR11(x, y)
\]
\[\Box out!y \rightarrow PR10(x)
\]

Appendix B

Question B.1*

Exercise B.1

datatype Status = on | off

channel coat: Status
channel store, retrieve, enter, eat

ATT = coat.off \rightarrow store \rightarrow ATT
\[\Box retrieve \rightarrow coat.on \rightarrow ATT
\]

CUST = enter \rightarrow coat.off \rightarrow eat \rightarrow coat.on \rightarrow CUST

MEALS = ATT \{\{coat\}\} CUST