

Concurrent and Real Time Systems

Sample solutions to exercises

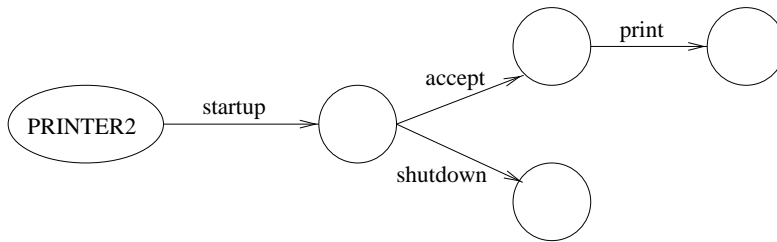
This document contains the generally available sample solutions to selected exercises from the book 'Concurrent and Real Time Systems: the CSP approach' by Steve Schneider.

Answers to the other questions have restricted access.

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

Chapter 1

Question 1.5*



Question 1.10*

$$\begin{aligned} PRESS &= PRESS(0) \\ PRESS(n) &= press \rightarrow PRESS(n+1) \\ &\quad | finish!n \rightarrow STOP \end{aligned}$$

Question 1.13*

TROLLEY = *choc* → *apple* → *cake* → *cake* → *STOP*
 | *cake* → *apple* → *cake* → *STOP*
 | *cake* → *apple* → *STOP*
| *apple* → *choc* → *cake* → *cake* → *STOP*
 | *cake* → *choc* → *cake* → *STOP*
 | *cake* → *choc* → *STOP*
| *cake* → *apple* → *choc* → *cake* → *STOP*
 | *cake* → *choc* → *STOP*
 | *choc* → *apple* → *cake* → *STOP*
 | *cake* → *apple* → *STOP*
 | *cake* → *apple* → *choc* → *STOP*
 | *choc* → *apple* → *STOP*

Question 2.9*

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

$$\begin{aligned}P_1 &= STOP \\P_2 &= STOP \\P_3 &= a \rightarrow STOP \\A &= \{a\} \\B &= \{\} \\P_1 \parallel_A (P_2 \parallel_B P_3) &= STOP\end{aligned}$$

$$(P_1 \parallel_A P_2) \parallel_B P_3 = a \rightarrow STOP$$

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

$$\begin{aligned}f_{i,j}(h_{in}) &= h_{i,j} \\f_{i,j}(v_{in}) &= v_{i,j} \\f_{i,j}(h_{out}) &= h_{i+1,j} \\f_{i,j}(v_{out}) &= v_{i,j+1}\end{aligned}$$

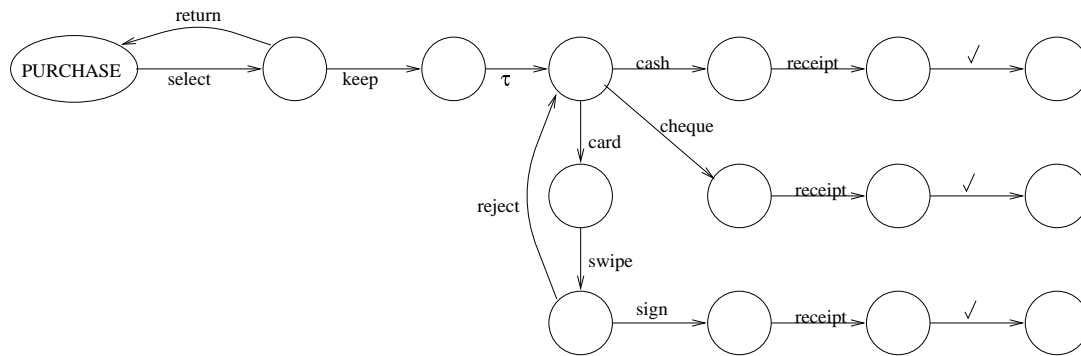
and if $i = j$ then the function $f_{i,j}$ is defined by

$$\begin{aligned}f_{i,j}(h_{in}) &= h_{i,j} \\f_{i,j}(v_{in}) &= v_{i,j} \\f_{i,j}(h_{out}) &= v_{i+1,j+1} \\f_{i,j}(v_{out}) &= v_{i,j+1}\end{aligned}$$

Finally, *SORTER* is defined as follows:

$$SORTER = \parallel_{0 \leq i \leq j \leq n} f_{i,j}(C)$$

Question 3.6*



Chapter 4

Question 4.4*

$$\begin{aligned}
 \text{traces}(P \square \text{RUN}) &= \text{traces}(P) \cup \text{traces}(\text{RUN}) \\
 &= \text{traces}(P) \cup \text{TRACE} \\
 &= \text{TRACE} \\
 &= \text{traces}(\text{RUN})
 \end{aligned}$$

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. $\{\langle \rangle, \langle \text{coin} \rangle, \langle \text{coin}, \text{change} \rangle, \langle \text{coin}, \text{ticket} \rangle, \langle \text{coin}, \text{coin} \rangle, \langle \text{coin}, \text{coin}, \text{ticket} \rangle, \langle \text{coin}, \text{coin}, \text{change} \rangle, \langle \text{coin}, \text{ticket}, \text{coin} \rangle, \langle \text{coin}, \text{change}, \text{coin} \rangle, \langle \text{coin}, \text{coin}, \text{ticket}, \text{change} \rangle, \langle \text{coin}, \text{coin}, \text{ticket}, \text{change}, \checkmark \rangle, \langle \text{coin}, \text{coin}, \text{change}, \text{ticket} \rangle, \langle \text{coin}, \text{coin}, \text{change}, \text{ticket}, \checkmark \rangle, \langle \text{coin}, \text{ticket}, \text{coin}, \text{change} \rangle, \langle \text{coin}, \text{ticket}, \text{coin}, \text{change}, \checkmark \rangle, \langle \text{coin}, \text{change}, \text{coin}, \text{ticket} \rangle, \langle \text{coin}, \text{change}, \text{coin}, \text{ticket}, \checkmark \rangle\}$
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 4.12*

$$\begin{aligned} \text{traces}(P) &= \{\langle \text{up} \rangle^n \mid n \in \mathbb{N}\} \\ &\cup \{\langle \text{up} \rangle^n \wedge \langle \text{down} \rangle^m \mid n \in \mathbb{N} \wedge m \in \mathbb{N} \wedge m \leq n\} \\ &\cup \{\langle \text{up} \rangle^n \wedge \langle \text{down} \rangle^n \wedge \langle \checkmark \rangle \mid n \in \mathbb{N}\} \end{aligned}$$

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*

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

where

$$\begin{aligned} (SPY \parallel \text{log!}x \rightarrow MASTER) \setminus \text{relay} &= \text{listen}?y \rightarrow \text{log!}x \rightarrow (SPY \parallel \text{log!}y \rightarrow MASTER) \setminus \text{relay} \\ &\mid \text{log!}x \rightarrow \text{listen}?y \rightarrow (SPY \parallel \text{log!}y \rightarrow MASTER) \setminus \text{relay} \end{aligned}$$

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

Chapter 5

Question 5.2*

The specification is

$$tr = tr_0 \wedge \langle \text{raw} \rangle \wedge tr_1 \wedge \langle \text{cooked} \rangle \wedge tr_2 \Rightarrow \text{wash in } tr_1$$

The combination $RAW \parallel_{\{\text{wash}\}} COOKED$ 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

$$F(Y) = (\textit{open} \rightarrow \textit{close} \rightarrow Y \square \textit{locked} \rightarrow \textit{STOP})$$

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 $\textit{foot}(tr) = \textit{open} \Rightarrow \textit{foot}(\textit{init}(tr)) \neq \textit{open}$
2. The specification that \textit{close} does not appear twice consecutively is not preserved by F (consider $F(\textit{close} \rightarrow \textit{STOP})$). It is necessary to find a stronger specification that is preserved by F . One that suffices is:

$$(\textit{foot}(tr) = \textit{close} \Rightarrow \textit{foot}(\textit{init}(tr)) \neq \textit{close}) \wedge \textit{head}(tr) \neq \textit{close}$$

3. This specification is preserved by F .

Chapter 6

Question 6.7*

$$\frac{}{DIV \xrightarrow{\tau} DIV}$$

Question 6.8*

$$\frac{}{CHAOS \xrightarrow{a} CHAOS} \quad [a \in \Sigma]$$

$$\frac{}{CHAOS \xrightarrow{\tau} STOP}$$

$$\frac{}{CHAOS \xrightarrow{\zeta} STOP}$$

Chapter 7

Question 7.3*

Both P_1 and P_2 satisfy $X \neq \Sigma$. Any refusal X of $P_1 \parallel P_2$ 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 $\exists b \notin X_1, b \neq a$ and so $b \notin X$. Thus $X \neq \Sigma$. Similarly if $a \in X_2$ then $X \neq \Sigma$.

Finally, if $a \notin X_1$ and $a \notin X_2$, then $a \notin X$, 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

$$\forall Y \bullet (SPEC \sqsubseteq_{SF} Y \Rightarrow SPEC \sqsubseteq_{SF} F(Y))$$

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

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

$$\begin{aligned} SPEC \sqsubseteq_{SF} Y &\Rightarrow F(SPEC) \sqsubseteq_{SF} F(Y) \\ &\Rightarrow SPEC \sqsubseteq_{SF} F(Y) \end{aligned}$$

by transitivity of refinement.

Question 7.8*

1.

$$\begin{aligned} STACK(\langle \rangle) &= push?x : T \rightarrow STACK(\langle x \rangle) \\ STACK(\langle x \rangle \wedge s) &= pop!x \rightarrow STACK(s) \\ &\quad \square (STOP \sqcap push?y : T \rightarrow STACK(\langle y, x \rangle \wedge s)) \end{aligned}$$

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

$$\begin{aligned} STACK_1(\langle \rangle) &= push?x : T \rightarrow STACK(\langle x \rangle) \\ STACK_1(\langle x \rangle) &= pop!x \rightarrow STACK(\langle \rangle) \end{aligned}$$

and $STACK_1(\langle \rangle)$ is a refinement of $STACK(\langle \rangle)$, 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:

$$\begin{aligned} STACK_2(\langle \rangle) &= push?x : T \rightarrow STACK(\langle x \rangle) \\ STACK_2(\langle x \rangle) &= pop!x \rightarrow STACK(\langle \rangle) \\ &\quad \square push?y : T \rightarrow STACK(\langle y, x \rangle) \\ STACK_2(\langle y, x \rangle) &= pop!x \rightarrow STACK(\langle x \rangle) \end{aligned}$$

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

Chapter 8

Question 8.5*

1. $P_1 = a \rightarrow STOP \sqcap b \rightarrow STOP$
2. Yes: $P_2 = (a \rightarrow b \rightarrow STOP) \sqcap (b \rightarrow a \rightarrow STOP)$
3. P_2 above has $P \parallel P \neq_{FDI} P$
4. Define $AS = a \rightarrow AS$ and $BS = b \rightarrow BS$. Then

$$\begin{aligned} P_1 &= AS \parallel \parallel (STOP \sqcap b \rightarrow STOP) \\ P_2 &= BS \parallel \parallel (STOP \sqcap a \rightarrow STOP) \end{aligned}$$

P_1 and P_2 are both nondeterministic, but their interleaved combination is deterministic.

5. No, if P is nondeterministic then so too is $P \parallel \parallel P$.
6. The deterministic process $P = a \rightarrow a \rightarrow b \rightarrow STOP$ has $P \parallel \parallel P$ 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:
 $S_D(tr) = \exists tr' \leq tr. tr' \downarrow choc > tr' \downarrow coin$
2. $S_D(tr) = \exists tr_0, tr_1. tr = tr_0 \wedge \langle in, in, in \rangle \wedge tr_1$
3. $S_D(tr) = \exists n \geq 2_{32}. in.n \mathbf{i}ntr$
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. $S_I(u) = tr \upharpoonright \{a\} \neq \langle \rangle$
6. Assuming all requests are unique (so any request can appear at most once in the trace), the requirement can be expressed as follows:

$$S_I(u) = (req.i \mathbf{i}n u) \Rightarrow \langle req.i, service.i \rangle \preceq u$$

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:

$$\begin{aligned} S_F(tr, X) &= \checkmark \notin \sigma(tr) \Rightarrow X \neq \Sigma' \\ S_I(u) &= false \end{aligned}$$

S_I here states that no infinite trace is possible—that the process should not have any infinite traces.

Chapter 9

Question 9.5*

$$\begin{aligned} CON1 &= in.kate \rightarrow out \rightarrow CON1 \\ &| in.eleanor \rightarrow out \rightarrow CON1 \end{aligned}$$

Question 9.6*

$$\begin{aligned} CON2 &= (in.kate \rightarrow (out \rightarrow RUN_{EKO} \triangleright^{10} RUN_{EKO})) \Delta_{60} CON2 \\ &| in.eleanor \rightarrow out \rightarrow CON2 \\ &| out \rightarrow CON2 \end{aligned}$$

Chapter 10

Question 10.2*

$N = n : \mathbb{N} \xrightarrow{2^{-n}} N$ has zeno executions but no spin executions.

Question 10.3*

If $N(i) = a \xrightarrow{i^{-1}} N(i+1)$, then $N(0)$ has no infinitely fast executions—all infinite executions take infinitely long to unwind. However, the $N(i)$ are not t -guarded 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 $(a \xrightarrow{2} STOP) \stackrel{2}{\succ} STOP$ 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 $\langle\langle(1, a)\rangle, [0, 1) \times \{a\}\rangle$ 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

$$(STOP \sqcap b \rightarrow SUCCESS) \sqcap (c \rightarrow STOP \sqcap a \rightarrow SUCCESS)$$

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*

$$shower \text{ at } t \Rightarrow eat \text{ live from } t + 10 \text{ until } \{eat\}$$

It is straightforward to show that

$$leave \rightarrow SKIP \quad \mathbf{sat} \quad \mathbf{no shower}$$

So the rule for prefix yields that

$$eat \xrightarrow{5} leave \rightarrow SKIP \quad \mathbf{sat} \quad eat \text{ live from } 0 \text{ until } \{eat\} \\ \wedge \mathbf{no shower}$$

so a further application of the rule for prefix yields that

$$\begin{aligned} \text{shower} \xrightarrow{10} \text{eat} \xrightarrow{5} \text{leave} \rightarrow \text{SKIP} \quad \text{sat} \quad s \neq \langle \rangle \Rightarrow \\ \text{shower at } \text{begin}(s) \wedge \text{eat live from } \text{begin}(s) + 10 \text{ until } \{\text{eat}\} \\ \wedge \text{no shower}[\text{tail}(s)/s] \end{aligned}$$

which implies

$$\text{shower} \xrightarrow{10} \text{eat} \xrightarrow{5} \text{leave} \rightarrow \text{SKIP} \quad \text{sat} \quad \text{shower at } t \Rightarrow \text{eat live from } t + 10 \text{ until } \{\text{eat}\}$$

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

$$\text{wake} \xrightarrow{10} \text{shower} \xrightarrow{10} \text{eat} \xrightarrow{5} \text{leave} \rightarrow \text{SKIP} \quad \text{sat} \quad \text{shower at } t \Rightarrow \text{eat live from } t + 10 \text{ until } \{\text{eat}\}$$

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*

$$\begin{aligned} a \rightarrow b \rightarrow \text{STOP} \parallel c \rightarrow \text{STOP} &=_{\mathcal{T}} c \rightarrow a \rightarrow b \rightarrow \text{STOP} \\ | a \rightarrow (b \rightarrow c \rightarrow \text{STOP} & \\ | c \rightarrow b \rightarrow \text{STOP}) & \\ \sqsubseteq_{\mathcal{T}} a \rightarrow (b \rightarrow \text{STOP} \sqcap c \rightarrow \text{STOP}) & \\ \mathcal{T} \sqsubseteq_{\mathcal{TF}} a \xrightarrow{2} (b \rightarrow \text{STOP} \sqcap c \rightarrow \text{STOP}) & \end{aligned}$$

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*

$$\begin{aligned}
 \Psi(PRINT = mid?y \xrightarrow{1} print!y \rightarrow PRINT) &= \\
 &= PRINT0 \\
 PRINT0 &= mid?y \rightarrow tock \rightarrow PRINT1(y) \\
 &\quad \square tock \rightarrow PRINT0 \\
 PRINT1(y) &= print!y \rightarrow PRINT0 \\
 &\quad \square tock \rightarrow PRINT1(y)
 \end{aligned}$$

Question A.4*

With $PRINT0$ as given in the previous question, and $SPOOL0$ as follows:

$$\begin{aligned}
 SPOOL0 &= in?x \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow SPOOL1(x) \\
 &\quad \square tock \rightarrow SPOOL0 \\
 SPOOL1(x) &= mid!x \rightarrow SPOOL0 \\
 &\quad \square tock \rightarrow SPOOL1
 \end{aligned}$$

we have to reduce

$$\begin{aligned}
 \Psi(PRINTER) &= (SPOOL0 \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T \\
 &= in?x \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow (SPOOL1(x) \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T \\
 &\quad \square tock \rightarrow (SPOOL0 \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T
 \end{aligned}$$

$$(SPOOL1(x) \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T = (SPOOL0 \parallel_{mid.T} PRINT1(x)) \setminus_{tock} mid.T$$

$$\begin{aligned}
 (SPOOL0 \parallel_{mid.T} PRINT1(x)) \setminus_{tock} mid.T &= \\
 out!x \rightarrow (SPOOL0 \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T & \\
 \square tock \rightarrow (SPOOL0 \parallel_{mid.T} PRINT1(x)) \setminus_{tock} mid.T & \\
 \square in?y \rightarrow out!x \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow (SPOOL1(x) \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T & \\
 \square tock \rightarrow out!x \rightarrow tock \rightarrow tock \rightarrow (SPOOL1(x) \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T & \\
 \square tock \rightarrow out!x \rightarrow tock \rightarrow (SPOOL1(x) \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T & \\
 \square tock \rightarrow (SPOOL1(x) \parallel_{mid.T} PRINT1(y)) \setminus_{tock} mid.T &
 \end{aligned}$$

$$\begin{aligned}
& (SPOOL1(x) \parallel_{mid.T} PRINT1(y)) \setminus_{tock} mid.T = \\
& \quad tock \rightarrow (SPOOL1(x) \parallel_{mid.T} PRINT1(y)) \setminus_{tock} mid.T \\
& \quad \square out!y \rightarrow (SPOOL1(x) \parallel_{mid.T} PRINT0) \setminus_{tock} mid.T
\end{aligned}$$

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

$$\begin{aligned}
\Psi(PRINTER) &= PR00 \\
&= in?x \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow PR01(x) \\
&\quad \square tock \rightarrow PR00 \\
PR01(x) &= out!x \rightarrow PR00 \\
&\quad \square tock \rightarrow PR01(x) \\
&\quad \square in?y \rightarrow out!x \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow PR01(x) \\
&\quad \quad \square tock \rightarrow out!x \rightarrow tock \rightarrow tock \rightarrow PR01(x) \\
&\quad \quad \quad \square tock \rightarrow out!x \rightarrow tock \rightarrow PR01(x) \\
&\quad \quad \quad \quad \square tock \rightarrow PR11(x, y) \\
PR11(x, y) &= tock \rightarrow PR11(x, y) \\
&\quad \square out!y \rightarrow PR10(x)
\end{aligned}$$

Appendix B

Question B.1*

-- Exercise B.1

```

datatype Status = on | off

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

ATT = coat.off -> store -> ATT
      [] retrieve -> coat.on -> ATT

CUST = enter -> coat.off -> eat -> coat.on -> CUST

MEALS = ATT [|{|coat|}|] CUST

```