Concurrent and Real Time Systems
Sample solutions to exercises

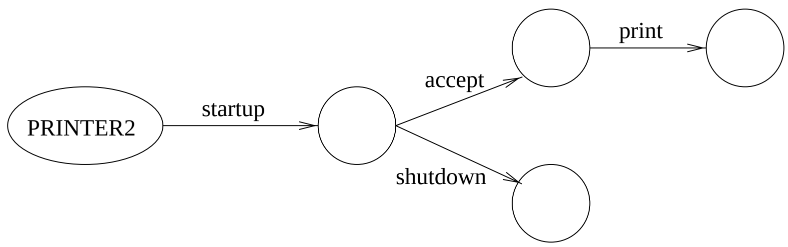
This document contains the generally available sample solutions to selected exercises from the book 'Concurrent and Real Time Systems: the CSP approach' by Steve Schneider.

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*

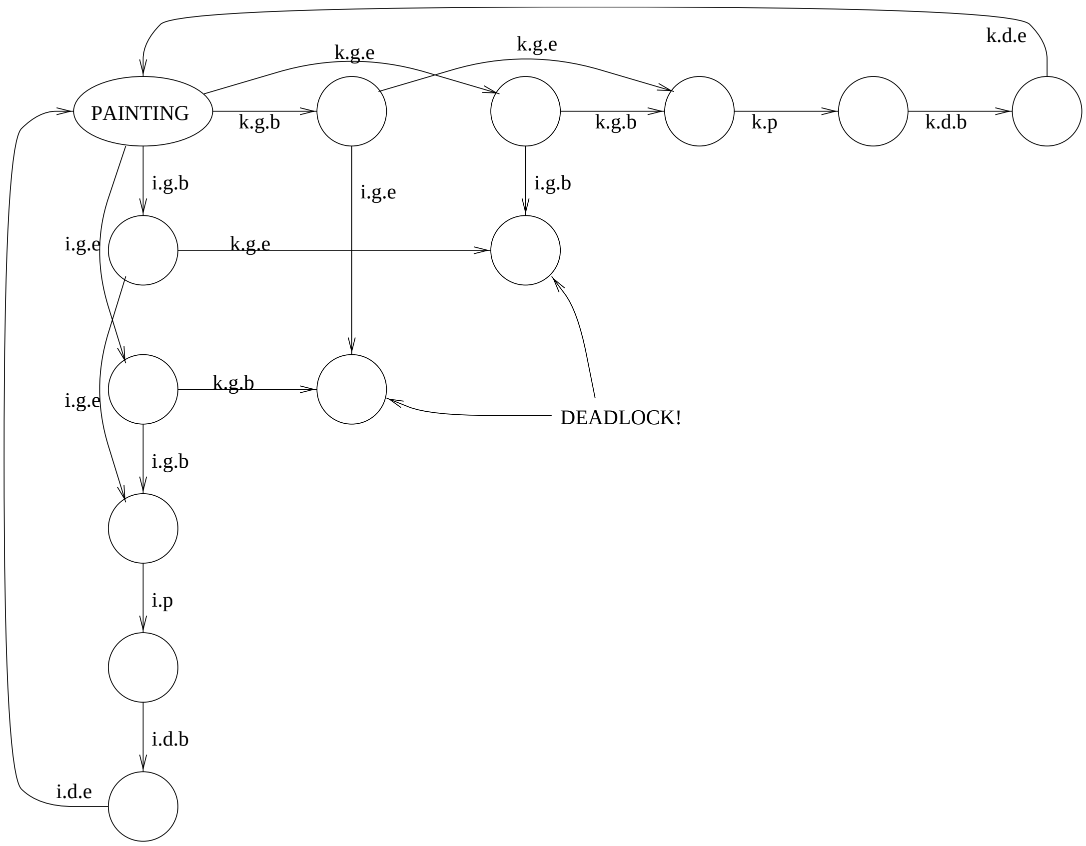
$$\begin{array}{rcl} PRESS & = & PRESS(0) \\ PRESS(n) & = & press \rightarrow PRESS(n+1) \\ & & \mid finish!n \rightarrow STOP \end{array}$$

Question 1.13*

$$
\begin{aligned}
TROLLEY \;=\;& choc \rightarrow (apple \rightarrow cake \rightarrow cake \rightarrow STOP \\
&\qquad\qquad \mid cake \rightarrow (apple \rightarrow cake \rightarrow STOP \\
&\qquad\qquad\qquad\qquad \mid cake \rightarrow apple \rightarrow STOP)) \\
&\mid apple \rightarrow (choc \rightarrow cake \rightarrow cake \rightarrow STOP \\
&\qquad\qquad \mid cake \rightarrow (choc \rightarrow cake \rightarrow STOP \\
&\qquad\qquad\qquad\qquad \mid cake \rightarrow choc \rightarrow STOP)) \\
&\mid cake \rightarrow (apple \rightarrow (choc \rightarrow cake \rightarrow STOP \\
&\qquad\qquad\qquad\qquad \mid cake \rightarrow choc \rightarrow STOP) \\
&\qquad\qquad \mid choc \rightarrow (apple \rightarrow cake \rightarrow STOP \\
&\qquad\qquad\qquad\qquad \mid cake \rightarrow apple \rightarrow STOP) \\
&\qquad\qquad \mid cake \rightarrow (apple \rightarrow choc \rightarrow STOP \\
&\qquad\qquad\qquad\qquad \mid choc \rightarrow apple \rightarrow STOP))
\end{aligned}
$$

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*

$$DCUSTOMER \parallel SHOP \quad = \quad enter \rightarrow select \rightarrow pay \rightarrow leave \rightarrow DCUSTOMER \parallel SHOP$$

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*.

$$\begin{aligned} DCUSTOMER' \parallel SHOP \quad = \quad & enter \rightarrow select \rightarrow \\ & (STOP \sqcap pay \rightarrow leave \rightarrow DCUSTOMER' \parallel SHOP) \end{aligned}$$

Question 2.9*

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

$$\begin{aligned} P_1 &= STOP \\ P_2 &= STOP \\ P_3 &= a \to STOP \\ A &= \{a\} \\ B &= \{\} \end{aligned}$$

$$P_1 \underset{A}{\|} (P_2 \underset{B}{\|} P_3) = STOP$$

$$(P_1 \underset{A}{\|} P_2) \underset{B}{\|} P_3 = a \to STOP$$

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*

$$C \quad = \quad h_{in}?x \to v_{in}?y \to v_{out}!\min\{x, y\} \to h_{out}!\max\{x, y\} \to C$$

If $i < j$ then the function $f_{i,j}$ is defined by

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

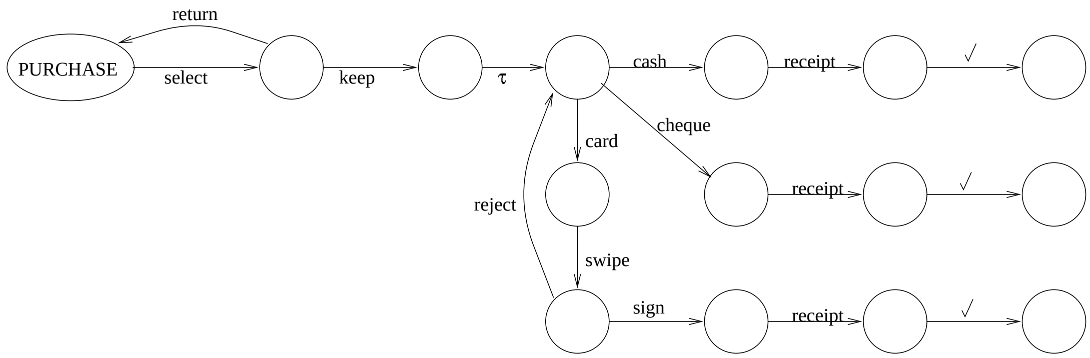
and if $i = j$ then the function $f_{i,j}$ is defined by

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

Finally, $SORTER$ is defined as follows:

$$SORTER \quad = \quad \|^{0 \leqslant i \leqslant j \leqslant n} f_{i,j}(C)$$

Question 3.6*



Chapter 4

Question 4.4*

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

Question 4.8*

1. $\{\langle\rangle, \langle coin\rangle, \langle coin, change\rangle, \langle coin, change, \checkmark\rangle, \langle coin, ticket\rangle, \langle coin, ticket, \checkmark\rangle\}$
2. $\{\ \langle\rangle, \langle coin\rangle, \langle coin, change\rangle, \langle coin, change, ticket\rangle, \langle coin, change, ticket, \checkmark\rangle,$
   $\langle coin, ticket\rangle, \langle coin, ticket, change\rangle, \langle coin, ticket, change, \checkmark\rangle\}$
3. $\{\ \langle\rangle, \langle coin\rangle, \langle coin, change\rangle, \langle coin, ticket\rangle, \langle coin, coin\rangle, \langle coin, coin, ticket\rangle, \langle coin, coin, change\rangle,$
   $\langle coin, ticket, coin\rangle, \langle coin, change, coin\rangle, \langle coin, coin, ticket, change\rangle,$
   $\langle coin, coin, ticket, change, \checkmark\rangle, \langle coin, coin, change, ticket\rangle, \langle coin, coin, change, ticket, \checkmark\rangle,$
   $\langle coin, ticket, coin, change\rangle, \langle coin, ticket, coin, change, \checkmark\rangle, \langle coin, change, coin, ticket\rangle,$
   $\langle coin, change, coin, ticket, \checkmark\rangle\}$
4. $\{\ \langle\rangle, \langle coin\rangle, \langle coin, change\rangle, \langle coin, change, \checkmark\rangle, \langle coin, ticket\rangle, \langle coin, ticket, \checkmark\rangle,$
   $\langle coin, coin\rangle, \langle coin, coin, ticket\rangle, \langle coin, coin, ticket, \checkmark\rangle, \langle coin, change, coin\rangle,$
   $\langle coin, change, coin, ticket\rangle, \langle coin, change, coin, ticket, \checkmark\rangle\}$

Question 4.12*

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

Question 4.14*

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

Question 4.18*

$$
(SPY \parallel MASTER) \setminus relay \quad = \quad listen?x : T \to (SPY \parallel log!x \to MASTER) \setminus relay
$$

where

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

Thus $(SPY \parallel log!y \to MASTER) \setminus relay$ 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

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

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

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 $P_1 \mid\mid\mid P_2$ has $\langle b, a\rangle$ as a possible trace, and this does not satisfy $tr \downarrow a \leqslant tr \downarrow c$.

Question 5.9*

It is necessary to prove that the recursive function

$$F(Y) = (open \to close \to Y \,\Box\, locked \to STOP)$$

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

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

3. This specification is preserved by F.

Chapter 6

Question 6.7*

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

Question 6.8*

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

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

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

Chapter 7

Question 7.3*

Both P_1 and P_2 satisfy $X \neq \Sigma$. Any refusal X of $P_1 \underset{\{a\}}{\|} P_2$ 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 $P_1 = a \to P_1$ and $P_2 = b \to P_2$, then both processes are strongly deadlock-free, but $P_1 \underset{\{a,b\}}{\|} P_2$ deadlocks.

Question 7.5*

The assertion

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

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

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

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

by transitivity of refinement.

Question 7.8*

1.

$$\begin{aligned} \mathit{STACK}(\langle\rangle) &= \mathit{push}?x : T \to \mathit{STACK}(\langle x\rangle) \\ \mathit{STACK}(\langle x\rangle \frown s) &= \mathit{pop}!x \to \mathit{STACK}(s) \\ &\quad \Box\ (\mathit{STOP} \sqcap \mathit{push}?y : T \to \mathit{STACK}(\langle y, x\rangle \frown s)) \end{aligned}$$

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

$$\begin{aligned} \mathit{STACK}_1(\langle\rangle) &= \mathit{push}?x : T \to \mathit{STACK}(\langle x\rangle) \\ \mathit{STACK}_1(\langle x\rangle) &= \mathit{pop}!x \to \mathit{STACK}(\langle\rangle) \end{aligned}$$

   and $\mathit{STACK}_1(\langle\rangle)$ is a refinement of $\mathit{STACK}(\langle\rangle)$, 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:

$$\begin{aligned} \mathit{STACK}_2(\langle\rangle) &= \mathit{push}?x : T \to \mathit{STACK}(\langle x\rangle) \\ \mathit{STACK}_2(\langle x\rangle) &= \mathit{pop}!x \to \mathit{STACK}(\langle\rangle) \\ &\quad \Box\ \mathit{push}?y : T \to \mathit{STACK}(\langle y, x\rangle) \\ \mathit{STACK}_2(\langle y, x\rangle) &= \mathit{pop}!x \to \mathit{STACK}(\langle x\rangle) \end{aligned}$$

   Thus $\mathit{STACK}_2(\langle\rangle)$ describes a stack, since it is a refinement of $\mathit{STACK}(\langle\rangle)$.

Chapter 8

Question 8.5*

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

$$\begin{array}{rcl} P_1 & = & AS \,|||\, (STOP \sqcap b \to STOP) \\ P_2 & = & BS \,|||\, (STOP \sqcap a \to STOP) \end{array}$$

   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 $P = a \to a \to b \to STOP$ 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: $S_D(tr) = \exists\, tr' \leqslant tr . tr' \downarrow choc > tr' \downarrow coin$
2. $S_D(tr) = \exists\, tr_0, tr_1 . tr = tr_0 \frown \langle in, in, in \rangle \frown tr_1$
3. $S_D(tr) = \exists\, n \geqslant 2_{32} . in.n \mathbf{in} tr$
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:

$$S_I(u) = \exists\, N . \forall\, tr < u . (\#tr > N \Rightarrow (tr \downarrow out / tr \downarrow in) \geqslant \frac{2}{3}$$

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:

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

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:

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

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

Chapter 9

Question 9.5*

$$
\begin{aligned}
CON1 &= in.kate \to out \to CON1 \\
&\quad \mid in.eleanor \to out \to CON1
\end{aligned}
$$

Question 9.6*

$$
\begin{aligned}
CON2 &= (in.kate \to (out \to RUN_{EKO} \overset{10}{\triangleright} RUN_{EKO})) \triangle_{60} CON2 \\
&\quad \mid in.eleanor \to out \to CON2 \\
&\quad \mid out \to CON2
\end{aligned}
$$

Chapter 10

Question 10.2*

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

Question 10.3*

If $N(i) = a \overset{i^{-1}}{\to} N(i+1)$, 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 $(a \xrightarrow{2} STOP) \overset{2}{\triangleright} STOP$ 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 $(\langle (1, a) \rangle, [0, 1) \times \{a\}$ 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*

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

It is straightforward to show that

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

So the rule for prefix yields that

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

so a further application of the rule for prefix yields that

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

which implies

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

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

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

as required.

Question 12.6*

$$\forall\, t . s \uparrow \{t\} \downarrow \Sigma \leqslant 1$$

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

Chapter 13

Question 13.4*

$$\begin{array}{lll}
a \rightarrow b \rightarrow STOP \;|||\; c \rightarrow STOP & =_T & c \rightarrow a \rightarrow b \rightarrow STOP \\
\quad \mid a \rightarrow \; (b \rightarrow c \rightarrow STOP & & \\
\quad\quad\quad \mid c \rightarrow b \rightarrow STOP) & & \\
 & \sqsubseteq_T & a \rightarrow (b \rightarrow STOP \;\Box\; c \rightarrow STOP) \\
 & {}_T\!\sqsubseteq_{TF} & a \xrightarrow{2} (b \rightarrow STOP \;\Box\; c \rightarrow STOP)
\end{array}$$

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*

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

Question A.4*

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

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

we have to reduce

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

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

$$
\begin{aligned}
&(SPOOL0 \underset{mid.T}{\|} PRINT1(x)) \setminus_{tock} mid.T = \\
&\quad out!x \rightarrow (SPOOL0 \underset{mid.T}{\|} PRINT0) \setminus_{tock} mid.T \\
&\quad \Box\ tock \rightarrow (SPOOL0 \underset{mid.T}{\|} PRINT1(x)) \setminus_{tock} mid.T \\
&\quad \Box\ in?y \rightarrow \quad out!x \rightarrow tock \rightarrow tock \rightarrow tock \rightarrow (SPOOL1(x) \underset{mid.T}{\|} PRINT0) \setminus_{tock} mid.T \\
&\qquad\qquad \Box\ tock \rightarrow \quad out!x \rightarrow tock \rightarrow tock \rightarrow (SPOOL1(x) \underset{mid.T}{\|} PRINT0) \setminus_{tock} mid.T \\
&\qquad\qquad\qquad \Box\ tock \rightarrow \quad out!x \rightarrow tock \rightarrow (SPOOL1(x) \underset{mid.T}{\|} PRINT0) \setminus_{tock} mid.T \\
&\qquad\qquad\qquad\qquad \Box\ tock \rightarrow (SPOOL1(x) \underset{mid.T}{\|} PRINT1(y)) \setminus_{tock} mid.T
\end{aligned}
$$

$$(SPOOL1(x) \underset{mid.T}{\|} PRINT1(y)) \setminus_{tock}\, mid.T =$$

$$\begin{array}{l} tock \to (SPOOL1(x) \underset{mid.T}{\|} PRINT1(y)) \setminus_{tock}\, mid.T \\ \Box\ out!y \to (SPOOL1(x) \underset{mid.T}{\|} PRINT0) \setminus_{tock}\, mid.T \end{array}$$

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

$$\begin{array}{rcl} \Psi(PRINTER) & = & PR00 \\ & = & in?x \to tock \to tock \to tock \to PR01(x) \\ & & \Box\ tock \to PR00 \\ PR01(x) & = & out!x \to PR00 \\ & & \Box\ tock \to PR01(x) \\ & & \Box\ in?y \to \quad out!x \to tock \to tock \to tock \to PR01(x) \\ & & \qquad\qquad \Box\ tock \to \quad out!x \to tock \to tock \to PR01(x) \\ & & \qquad\qquad\qquad\qquad \Box\ tock \to \quad out!x \to tock \to PR01(x) \\ & & \qquad\qquad\qquad\qquad\qquad\qquad \Box\ tock \to PR11(x,y) \\ PR11(x,y) & = & tock \to PR11(x,y) \\ & & \Box\ out!y \to PR10(x) \end{array}$$

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
```