Principles of Model Checking

Solutions to Selected Exercises

Christel Baier
Joost-Pieter Katoen

Appendix B

Solutions to Selected Exercises

This document contains solutions to selected exercises in the book *Principles of Model Checking*, first edition, by Christel Baier (TU Dresden, Germany) and Joost-Pieter Katoen (RWTH Aachen University, Germany & University of Twente, the Netherlands). This booklet is intended for lecturers and instructors for courses that are based on this book.

Most of the solutions in this booklet are adapted from solution sets handed out to students in classes we taught in the last couple of years. We thank our instructors for these classes for their contributions, in particular, Tobias Blechmann, Frank Ciesinski, Markus Grösser, Tingting Han, Joachim Klein, Sascha Klüppelholz, Miriam Nasfi, Martin Neuhäusser, and Ivan S. Zapreev for their contributions.

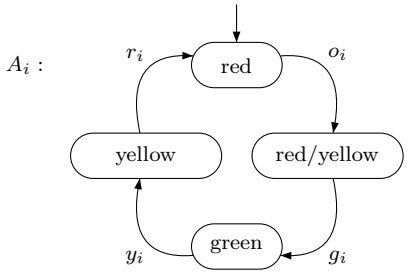
This is a draft version. Solutions are provided without any guarantee of correctness. We welcome suggestions for improvements to the solutions, as well as elegant solutions to exercises of the book that are not covered in this booklet.

List of Covered Exercises

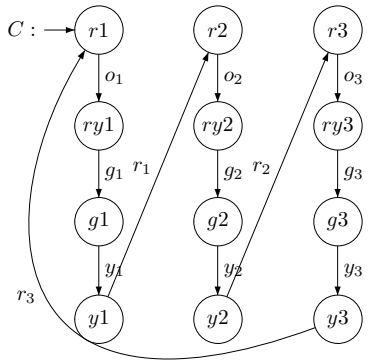
Exercises of Chapter 2

Answer to Exercise 2.3

(a) Let $Act = \bigcup_{0<i\leqslant 5}\{\, r_i, o_i, g_i, y_i \,\}$. Labeling the transition system A_i yields:



(b) The controller has to synchronize with the traffic lights. Note that the actions defined in part (a) uniquely identify the i-th transition system. This is exploited by the controller in the following way. The controller synchronizes with the traffic lights using pairwise handshaking.



(c) Let $TS_1 \| \cdots \| TS_n$ denote the parallel composition of TS_1 through TS_n where TS_i and TS_j $(0 < i < j \leqslant n)$ synchronize over the set of actions $H_{i,j} = Act_i \cap Act_j$ such that $H_{i,j} \cap Act_k = \varnothing$ for $k \notin \{\, i, j \,\}$. The inference rules for the transition relation are:

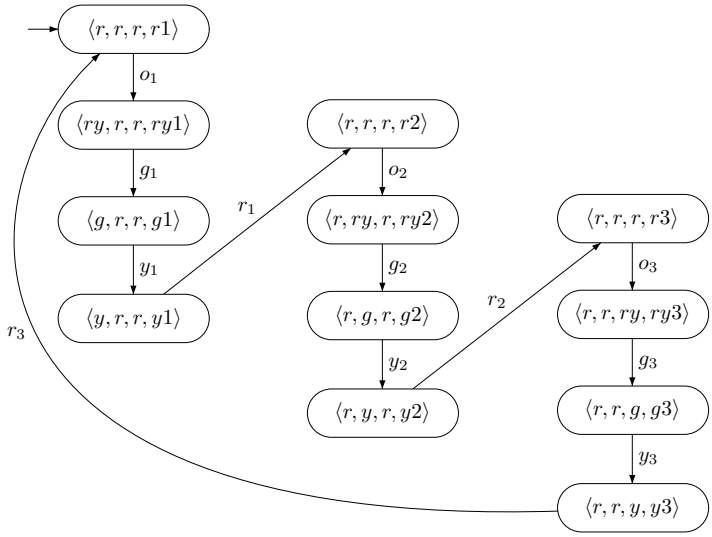
- if $\alpha \in Act_i \setminus \bigcup_{0<j\leqslant n, i\neq j} H_{i,j}$ and $0 < i \leqslant n$:

$$\frac{s_i \xrightarrow{\alpha}_i s_i'}{\langle s_1, \ldots, s_i, \ldots, s_n\rangle \xrightarrow{\alpha} \langle s_1, \ldots, s_i', \ldots, s_n\rangle}$$

- if $\alpha \in H_{i,j}$ and $0 < i < j \leqslant n$:

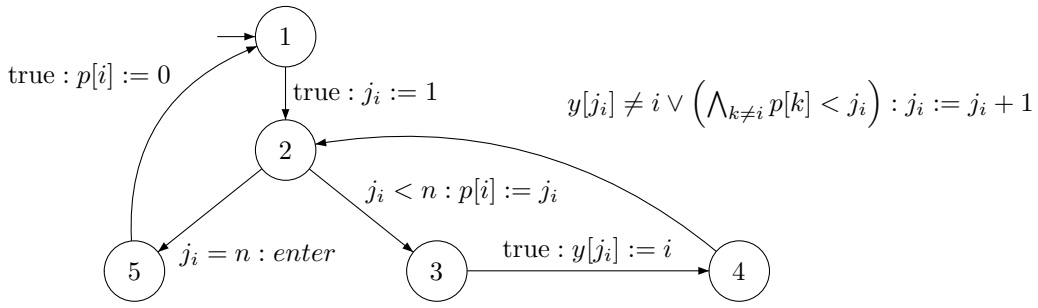
$$\frac{s_i \xrightarrow{\alpha}_i s_i' \qquad s_j \xrightarrow{\alpha}_j s_j'}{\langle s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n\rangle \xrightarrow{\alpha} \langle s_1, \ldots, s_i', \ldots, s_j', \ldots, s_n\rangle}$$

By applying these inference rules, the transition system $A_1 \| A_2 \| A_3 \| C$ becomes:



Answer to Exercise 2.7

(a) The program graph PG_i for process i is given as:



Note that we consider i as a constant here and that the variables j_i are private to process i.

(b) The cardinality of the set of states of $TS(PG_1 ||| \cdots ||| PG_n)$ can be deduced as follows. Let $PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_0, g_0)$ be the formal representation of the program graph from part (a) where:

- $Loc_i = \{\, 1, 2, 3, 4, 5 \,\}$
- $Act_i = \{\, j_i := 1, p[i] := j_i, y[j_i] := i, j_i := j_i{+}1, enter, p[i] := 0 \mid i \in \{1, \ldots, n\} \,\}$

According to the algorithm, we have:

$$\begin{array}{lcll} dom(y[k]) & = & \{\, 1, \ldots, n \,\} & \text{for all } k \in \{0, \ldots, n-1\} \\ dom(j_i) = dom(p[k]) & = & \{\, 0, \ldots, n-1 \,\} & \text{for all } k, i \in \{1, \ldots, n\} \end{array}$$

Therefore it follows $|dom(y[k])| = |dom(j_i)| = |dom(p[k])| = n$. The arrays y and p have capacity n. The state space of the transition system is:

$$S = Loc_1 \times \cdots \times Loc_n \times Eval(\{p[k], y[l], j_i \mid i,k \in \{1,\ldots,n\} \text{ and } l \in \{0,\ldots,n-1\}\}) .$$

Therefore we obtain $|S| = 5^n \cdot n^{3n}$.

(c) We prove a stronger statement that implies mutual exclusion:

$$\text{At level } j \in \{\,0,\ldots,n{-}1\,\}, \text{ at most } n{-}j \text{ processes are at level} \geqslant j$$

By definition, process P_i is at level j iff $p[i] = j$. We proceed by induction over j:

- **basis** ($j = 0$): The statement trivially holds, as $n{-}j = n{-}0 = n$ and there are at most n processes in the system.
- **induction step** ($j \rightsquigarrow j+1$): The induction hypothesis implies that there are at most $n{-}j$ processes at level $\geqslant j$. We show that there is at least one process that cannot move from level j to level $j{+}1$. By contradiction, assume there also were $n-j$ processes at levels $\geqslant j{+}1$ (i.e., no process is stuck at level j). Let i be the last process that writes to $y[j]$. Therefore, the old value of $y[j]$ that corresponds to the previous process k at level j is overwritten and we have $y[j] = i$. Hence the condition $y[j] \neq i$ cannot be true. According to the algorithm,
  - process k writes $p[k]$ before it writes $y[j]$ and
  - process i reads $p[k]$ only after it wrote to $y[j]$.

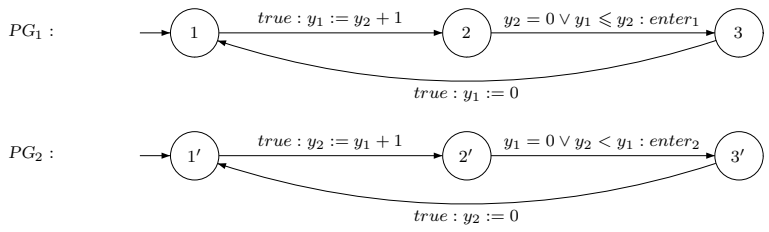
  Therefore every time process i reads $p[k]$, process k already set $p[k] = j$ and for process i, the second condition $p[k] < j$ is not fulfilled either.

  We assumed that process i enters level $j{+}1$. This yields a contradiction since it cannot leave the wait–loop.

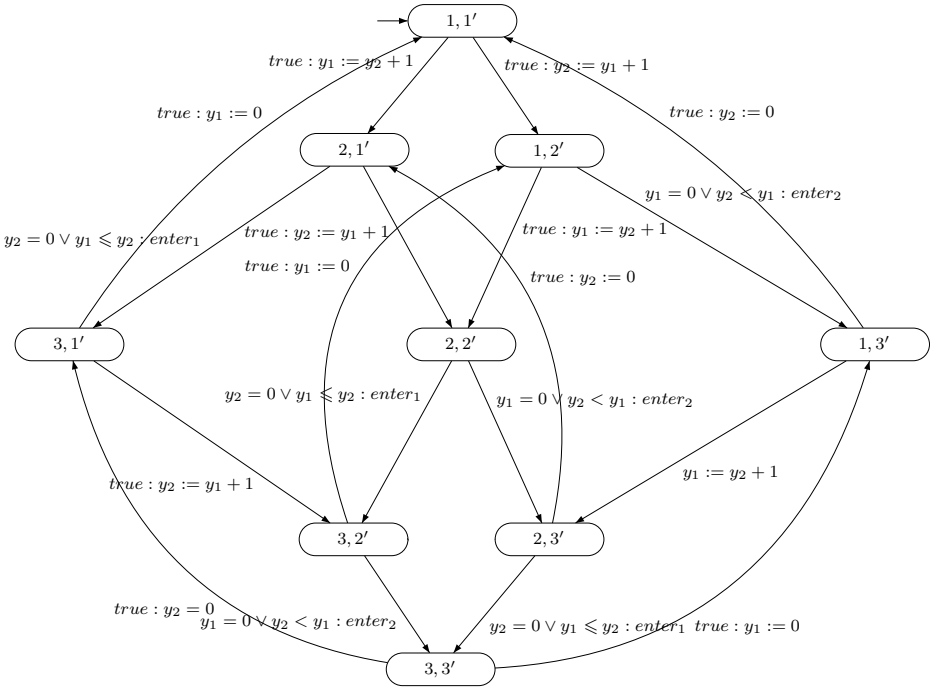
According to the idea of the algorithm, a process enters the critical section when it leaves the wait–loop at level $n{-}1$. As we proved, at level $n{-}1$, there may only be $n{-}(n{-}1) = 1$ processes at level $\geqslant (n{-}1)$. Therefore, the mutual exclusion property holds.

Answer to Exercise 2.9

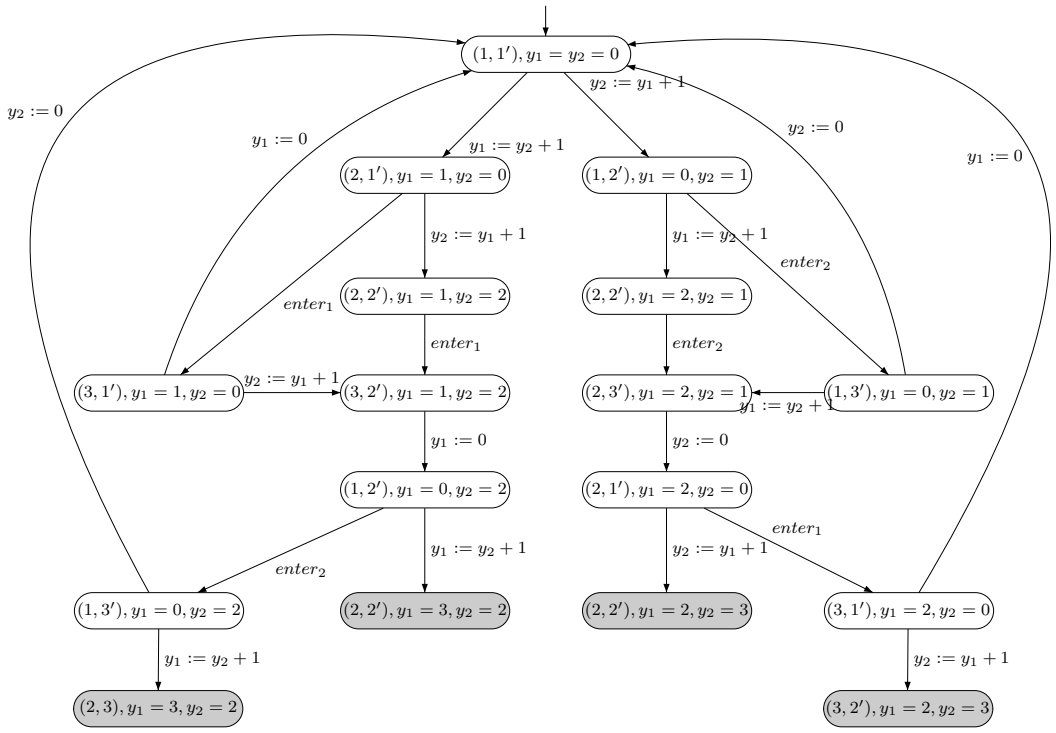
(a) The program graphs of the two processes can be depicted as follows:



(b) First, we provide the program graph $PG_1 \| PG_2$:



The transition system $TS(PG_1|||PG_2)$ for $y_1 \leqslant 2$ and $y_2 \leqslant 2$ becomes:



(c) To show that the complete transition system is infinite, we consider the infinite execution:

$$
\begin{aligned}
\langle 1, 1', y_1 = y_2 = 0 \rangle \quad & \xrightarrow{y_1 := y_2 + 1} \langle 2, 1', y_1 = 1, y_2 = 0 \rangle \\
& \xrightarrow{y_2 := y_1 + 1} \langle 2, 2', y_1 = 1, y_2 = 2 \rangle \\
& \xrightarrow{enter_1} \langle 3, 2', y_1 = 1, y_2 = 2 \rangle \\
& \xrightarrow{y_1 := 0} \langle 1, 2', y_1 = 0, y_2 = 2 \rangle \\
& \xrightarrow{y_1 := y_2 + 1} \langle 2, 2', y_1 = 3, y_2 = 2 \rangle \\
& \xrightarrow{enter_2} \langle 2, 3', y_1 = 3, y_2 = 2 \rangle \\
& \xrightarrow{y_2 := 0} \langle 2, 1', y_1 = 3, y_2 = 0 \rangle \\
& \xrightarrow{y_2 := y_1 + 1} \langle 2, 2', y_1 = 3, y_2 = 4 \rangle \\
& \xrightarrow{enter_1} \langle 3, 2', y_1 = 3, y_2 = 4 \rangle \\
& \ldots
\end{aligned}
$$

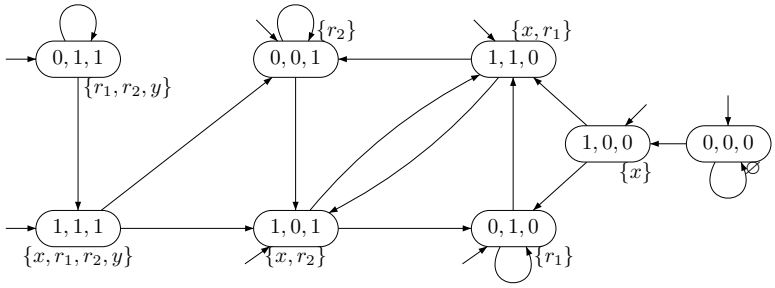
Answer to Exercise 2.11

(a) The output and the circuit control functions for C_1 are as follows:

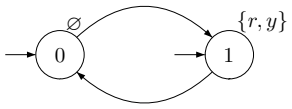
$$
\begin{aligned}
\lambda_y &= r_1 \wedge r_2 \\
\delta_{r_1} &= (x \wedge \neg r_1) \vee (\neg x \wedge r_1) = x \oplus r_1 \\
\delta_{r_2} &= (\neg x \wedge r_2) \vee (x \wedge r_1)
\end{aligned}
$$

The transition system is given by $TS(C_1) = TS_1 = (Eval(\{x, r_1, r_2\}), Act, \rightarrow, I, AP, L)$ where

- $I = \{\, (x, r_1, r_2) \mid x, r_1, r_2 \in \mathbb{B} \,\} = \mathit{Eval}(\{\, x, r_1, r_2 \,\})$
- $AP = \{\, x, r_1, r_2, y \,\}$
- $L : S \to 2^{AP}$ as given in the figure below where evaluations are represented as triples (x, r_1, r_2).



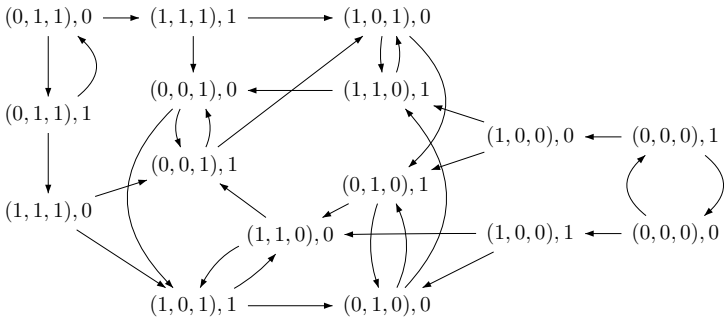
(b) The transition system representation TS_2 of the circuit C_2 is given by:



The synchronous composition of TS_1 and TS_2 is defined as the transition system

$$TS_1 \otimes TS_2 = (S_1 \times S_2, \mathit{Act}, \to, I_1 \times I_2, AP_1 \uplus AP_2, L)$$

It is given by the diagram below. Note that the set of initial states in this case equals the set of states $S_1 \times S_2$, i.e., every state can serve as an initial state. Therefore we do not indicate the initial states. We also omit the atomic propositions. These can be defined analogously to part (a) by renaming the variables of C_2 to r_1' and y', respectively.



Answer to Exercise 2.12

(a) The program graph for process P_i is given by $PG_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_0^i, g_0^i)$ over the set of variables $Var_i = \{ id_i, m_i \}$. To interconnect the processes, we define the channels as

$$Chan := \{c_{ij} \mid i \in \{1, \ldots, n\}, j = (i+1) \bmod n\}$$

where each channel has capacity n, i.e., $cap(c) := n \quad \forall c \in Chan$. The domain of the channel is defined by $dom(c) := \{1, \ldots, n\} \quad \forall c \in Chan$.

The program graph for process P_i is given by:

$$\begin{array}{lcl} Loc_i & := & \{start_i, recv_i, test_i, stop_i\} \\ Act_i & := & \{noop_i\} \\ Effect_i(noop_i, \eta) & := & \eta \\ Loc_0^i & := & \{ start_i \} \\ g_0^i & := & id_i = id \wedge m = 0 \end{array}$$

The transition relation is given as follows:

$$\begin{array}{l} start_i \xrightarrow{c_{i,i+1}!id_i} recv_i \\ recv_i \xrightarrow{c_{i-1,i}?m_i} test_i \\ test_i \xrightarrow{m_i = id_i : noop_i} stop_i \\ test_i \xrightarrow{m_i > id_i : c_{i,i+1}!m_i} recv_i \\ test_i \xrightarrow{m_i < id_i : noop_i} recv_i \end{array}$$

(b) The transition system for $CS = (P_1 | P_2 | \cdots | P_n)$ is given by:

$$TS(CS) = (S, Act, \rightarrow, I, AP, L) \quad \text{where}$$

- $S := Loc_1 \times \ldots \times Loc_n \times Eval(\bigcup_{i=1}^n Var_i) \times Eval(Chan)$
- $Act := \{noop_i \mid i \in \{1, \ldots, n\}\} \cup \{\tau\}$
- $\rightarrow \subseteq S \times Act \times S$
- $I := \{(start_1, \ldots, start_n, \eta, \xi_0) \mid \bigwedge_{i=1}^n \eta \models g_{0,i}\}$ where $\xi_0 : Chan \to dom(c)^*$ denotes the initial channel evaluation
- $AP := \{1, \ldots, n\} \cup \bigcup_{i=1}^n Loc_i$
- $L : S \to 2^{AP} : (l_1, \ldots, l_n, \eta, \xi) \mapsto \{l_1, \ldots, l_n\} \cup \{id_i = n \mid n \in \mathbb{N}\}$

An initial execution fragment is:

$$\begin{array}{ll} & (st, st, st, \overbrace{(id_1/1, id_2/2, id_3/3, m_1/0, m_2/0, m_3/0)}^{\eta}, c_{1,2} = c_{2,3} = c_{3,1} = \varepsilon) \\ \xrightarrow{\tau} & (rc, st, st, \eta, c_{1,2} = 1, c_{2,3} = c_{3,1} = \varepsilon) \\ \xrightarrow{\tau} & (rc, st, rc, \eta, c_{1,2} = 1, c_{2,3} = \varepsilon, c_{3,1} = 3) \\ \xrightarrow{\tau} & (test, st, rc, \eta\,(m_1/3)\,, c_{1,2} = 1, c_{2,3} = c_{3,1} = \varepsilon) \\ \xrightarrow{\tau} & (rc, st, rc, \eta\,(m_1/3)\,, c_{1,2} = 13, c_{2,3} = c_{3,1} = \varepsilon) \\ \ldots & \ldots \end{array}$$

Exercises of Chapter 3

Answer to Exercise 3.1
$\mathit{Traces}(TS) = (\{a\}\{a\} + \{a\}\varnothing)\,(\{a,b\} + \{a,b\}\{a\})^\omega$

Answer to Exercise 3.2

(a) Let $TS = (S, Act, \to, I, AP, L)$ be a transition system. Let $TS^* = (S \uplus \{\bot\}, Act, \to', I, AP, L')$ where $\to' := \to \uplus \{s \xrightarrow{\beta} \bot \mid s \in S, \beta \in Act, \forall \alpha \in Act.\ \nexists s' \in S.\ s \xrightarrow{\alpha} s'\} \uplus \{\bot \xrightarrow{\alpha} \bot \mid \alpha \in Act\}$. Extend the labeling function L with respect to the new state $\bot$ as follows:

$$L' : S \uplus \{\bot\} \to 2^{AP} \quad : \quad s \mapsto L(s), \text{ if } s \in S, \text{ and } \varnothing, \text{ otherwise.}$$

(b) For $i \in \{1,2\}$, let $TS_i = (S_i, Act_i, \to_i, I_i, AP, L_i)$ denote two transition systems, possibly with terminal states, over the same set of atomic propositions AP. We have to show:

$$\mathit{Traces}(TS_1) = \mathit{Traces}(TS_2) \implies \mathit{Traces}(TS_1^*) = \mathit{Traces}(TS_2^*)$$

where TS_1^* and TS_2^* are defined according to part (a) of this exercise.

"$\subseteq$" Let $\delta \in \mathit{Traces}(TS_1^*)$. Then there exists a path $\pi \in \mathit{Paths}(TS_1^*)$ such that $\delta = \mathit{trace}(\pi)$. Let $\pi = s_0 s_1 s_2 \cdots$ with $s_i \in S \uplus \{\bot\}$ for all $i \in \mathbb{N}$.

- **Case 1:**

$$\begin{aligned}
\pi \in S_1^\omega \quad &\xrightarrow{(a)} (s_i, s_{i+1}) \in \to \quad \text{for all } i \in \mathbb{N} \\
&\implies \pi \in \mathit{Paths}(TS_1) \\
&\implies \delta \in \mathit{Traces}(TS_1) \\
&\implies \delta \in \mathit{Traces}(TS_2) \\
&\implies \exists \pi' \in \mathit{Paths}(TS_2).\quad \delta = \mathit{trace}(\pi') \\
&\xrightarrow{(a)} \pi' \in \mathit{Paths}(TS_2^*) \\
&\implies \mathit{trace}(\pi') \in \mathit{Traces}(TS_2^*) \\
&\implies \delta \in \mathit{Traces}(TS_2^*)
\end{aligned}$$

- **Case 2:**

$$\begin{aligned}
\pi \notin S_1^\omega \quad &\implies \pi \in (S_1 \cup \{\bot\})^\omega \\
&\xrightarrow{(a)} \pi = \overbrace{s_0 s_1 s_2 \ldots s_k}^{\hat{\pi}}\, s_\bot^\omega \text{ and } \hat{\pi} \in \mathit{Paths}_{fin}(TS_1) \\
&\xrightarrow{(a)} s_k \text{ is a terminal state in } TS_1 \\
&\implies \hat{\delta} \in \mathit{Traces}_{fin}(TS_1) \text{ with } \hat{\delta} = \mathit{trace}(\hat{\pi}) \\
&\implies \hat{\delta} \in \mathit{Traces}_{fin}(TS_2) \text{ and there exists } \hat{\pi}' \in \mathit{Paths}_{fin}(TS_2) \\
&\qquad \text{s.t. } \mathit{trace}(\hat{\pi}') = \hat{\delta} \text{ and } \mathit{last}(\hat{\pi}') \text{ is a terminal state in } TS_2 \\
&\xrightarrow{(a)} \hat{\pi}' s_\bot^\omega \in \mathit{Paths}(TS_2^*) \\
&\xrightarrow{(a)} \mathit{trace}(\hat{\pi}' s_\bot^\omega) = \hat{\delta}\varnothing^\omega = \delta \in \mathit{Traces}(TS_2^*)
\end{aligned}$$

"$\supseteq$" follows by symmetry.

Answer to Exercise 3.3
The algorithm is indicated in Algorithm B.

Algorithm 51 Invariant Checking using Breadth–First Search

Input: finite transition system TS and propositional formula Φ
Output: **true** or the shortest counterexample

---

queue of states $Q = \varepsilon$;
finite trace $\hat{\sigma} = \varepsilon$;
set of states R;
set of tuples $P \subseteq S \times S$;

---

procedure bfs(state s)
enqueue(Q, s);
$P := \{(s, \bot)\}$;
$R := \{s\}$;
while $(Q \neq \varepsilon) \wedge (\text{first}(Q) \models \Phi)$ **do**
  let $p := \text{dequeue}(Q)$;
  for all $p' \in \text{Post}(p) \setminus R$ **do**
    enqueue(Q, p');
    $R := R \cup \{p'\}$;
    $P := P \cup \{(p', p)\}$;
  od
od
if $Q \neq \varepsilon$ **then**
  let $p := \text{first}(Q)$;
  while $p \neq \bot$ **do**
    $\hat{\sigma} := p.\hat{\sigma}$;
    let $(p, p') \in P$;
    $p := p'$;
  od
  return false; shortest counterexample $\hat{\sigma}$;
else
  return true;
fi

Answer to Exercise 3.5

(a) $P_{\text{false}} = \varnothing$

(b) initially, x is equal to zero: $\{x = 0\}(2^{AP})^{\omega}$.

(c) initially, x differs from zero: $(\varnothing + \{x > 1\})(2^{AP})^{\omega}$

(d) initially, x is equal to zero, but at some point x exceeds one: $\{x = 0\}(2^{AP})^{*}\{x > 1\}(2^{AP})^{\omega}$.

(e) x exceeds one only finitely often: $((2^{AP})^*\{\{x=0\},\varnothing\}^\omega$.

(f) x exceeds one infinitely often: $(2^{AP})^*\{x>1\}^\omega$.

(g) the value of x alternates between zero and one: $(\{x=0\}\varnothing)^\omega + (\varnothing\{x=0\})^\omega$

(h) $P_{\text{true}} = \left(2^{AP}\right)^\omega$

Let us check which of the above LT properties are safety properties.

(a)+(h) $P_{\text{false}} = \varnothing$ and $P_{\text{true}} = \left(2^{AP}\right)^\omega$ are both safety properties. Since $P_{\text{false}} = \varnothing$ and $\mathit{closure}(\varnothing) = \varnothing$, it follows that P_{false} is a safety property. As $\mathit{closure}(P_{\text{true}}) = P_{\text{true}}$, P_{true} is a safety property.

(b) initially, x is equal to zero: $P_2 = \mathcal{L}_\omega\big(\{x=0\}.\left(2^{AP}\right)^\omega\big)$. P_2 is a safety property with $\mathit{MinBadPref}(P_2)\,\{\{x>0\},\varnothing\}$.

(c) initially, x differs from zero: $P_3 = \mathcal{L}_\omega\big((\varnothing + \{x>1\})\,.\left(2^{AP}\right)^\omega\big)$. P_3 is a safety property with $\mathit{MinBadPref}(P_3) = \{\{x=0\}\}$

(d) initially, x is equal to zero, but at some point, x exceeds one:

$$P_4 = \mathcal{L}_\omega\big(\{x=0\}.\left(2^{AP}\right)^*.\{x>1\}.\left(2^{AP}\right)^\omega\big).$$

It follows that $\mathit{closure}(P_4) = \mathcal{L}_\omega\big(\{x=0\}.\left(2^{AP}\right)^\omega\big)$. Therefore $\mathit{closure}(P_4) \neq P_4$ and P_4 is not a safety property.

(e) x exceeds one only finitely many times: $P_5 = \mathcal{L}_\omega\big(\left(2^{AP}\right)^*.(\{x=0\}+\varnothing)^\omega\big)$. We have that $\mathit{closure}(P_5) = \left(2^{AP}\right)^\omega$ and $\mathit{closure}(P_5) \neq P_5$. Therefore P_5 is not a safety property.

(f) x should exceed one infinitely often. This is not a safety property.

(g) the value of x alternates between zero and one: $P_7 = \mathcal{L}_\omega\big((\{x=0\}.\varnothing)^\omega + (\varnothing.\{x=0\})^\omega\big)$: P_7 is a safety property with:

$$\mathit{MinBadPref}(P_7) = \mathcal{L}\big(\left(2^{AP}\right)^*.\{x>1\} \;+\; \left(2^{AP}\right)^*.\varnothing.\varnothing \;+\; \left(2^{AP}\right)^*.\{x=0\}.\{x=0\}\big)$$

The first summand refers to the situation in which at some point x exceeds 1, the second summand to the situation in which x equals 1 at successive positions, and finally the last summand does the same for x being zero.

Answer to Exercise 3.8
The proof obligation is: $\mathit{pref}(P) = \mathit{pref}(P')$ iff $\mathit{closure}(P) = \mathit{closure}(P')$. The "only if" direction is straightforward:

$$\mathit{closure}(P) = \left\{\sigma \in \left(2^{AP}\right)^\omega \mid \mathit{pref}(\sigma) \subseteq \mathit{pref}(P)\right\} = \left\{\sigma \in \left(2^{AP}\right)^\omega \mid \mathit{pref}(\sigma) \subseteq \mathit{pref}(P')\right\} = \mathit{closure}(P').$$

For the "if" direction, we obtain using the solution of Exercise 3.10 (see below):

$$\mathit{pref}(P) = \mathit{pref}(\mathit{closure}(P)) = \mathit{pref}(\mathit{closure}(P')) = \mathit{pref}(P').$$

Answer to Exercise 3.9
Let TS be a transition system. We have to show that

1. $\mathit{closure}(\mathit{Traces}(TS))$ is a safety property and
2. $TS \models \mathit{closure}(\mathit{Traces}(TS))$

Proof.

1. We have to show that $P = \mathit{closure}(\mathit{Traces}(TS))$ is a safety property. LT-property P is a safety property if and only if $P = \mathit{closure}(P)$ holds. With $P = \mathit{closure}(\mathit{Traces}(TS))$, this boils down to show that $\mathit{closure}(\mathit{closure}(\mathit{Traces}(TS))) = \mathit{closure}(\mathit{Traces}(TS))$. This goes as follows:

$$\begin{aligned}\mathit{closure}(\mathit{closure}(\mathit{Traces}(TS))) &= \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid \mathit{pref}(\sigma) \subseteq \mathit{pref}(\mathit{closure}(\mathit{Traces}(TS)))\right\} \\ &\overset{(*)}{=} \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid \mathit{pref}(\sigma) \subseteq \mathit{pref}(\mathit{Traces}(TS))\right\} \\ &= \mathit{closure}(\mathit{Traces}(TS))\end{aligned}$$

   where (*) refers to the answer to Exercise 3.10. Therefore $\mathit{closure}(\mathit{Traces}(TS))$ is a safety property.

2. We show, that $TS \models \mathit{closure}(\mathit{Traces}(TS))$. From the definition of *closure*, we can infer for any LT-property P: $P \subseteq \mathit{closure}(P)$. Therefore we have

   $$\mathit{Traces}(TS) \subseteq \mathit{closure}(\mathit{Traces}(TS)).$$

   Furthermore for each transition system TS, it holds $TS \models \mathit{Traces}(TS)$.
   As $\mathit{Traces}(TS) \subseteq \mathit{closure}(\mathit{Traces}(TS))$, we can directly infer $TS \models \mathit{closure}(\mathit{Traces}(TS))$.

Answer to Exercise 3.10
We have to show that $\mathit{pref}(\mathit{closure}(P)) = \mathit{pref}(P)$ for any LT-property P.

"$\subseteq$":

$$\begin{aligned}\text{Let } \hat{\sigma} \in \mathit{pref}(\mathit{closure}(P)) &\Longrightarrow \exists \sigma \in \left(2^{AP}\right)^{\omega} \text{ with } \sigma \in \mathit{closure}(P) \text{ and } \hat{\sigma} \in \mathit{pref}(\sigma) \\ &\xrightarrow{p.D.} \mathit{pref}(\sigma) \subseteq \mathit{pref}(P) \\ &\Longrightarrow \hat{\sigma} \in \mathit{pref}(P).\end{aligned}$$

$\supseteq$:

$$\begin{aligned}\text{Let } \hat{\sigma} \in \mathit{pref}(P) &\Longrightarrow \exists \sigma \in P \text{ with } \hat{\sigma} \in \mathit{pref}(\sigma) \\ &\Longrightarrow \text{since } \sigma \in P \text{ we have } \mathit{pref}(\sigma) \subseteq \mathit{pref}(P) \\ &\Longrightarrow \sigma \in \mathit{closure}(P) \\ &\Longrightarrow \hat{\sigma} \in \mathit{pref}(\mathit{closure}(P)).\end{aligned}$$

Answer to Exercise 3.11
Assume P and P' are liveness properties.

- $P \cup P'$ is a liveness property. This can be seen as follows. $\mathit{pref}(P) = \mathit{pref}(P') = \left(2^{AP}\right)^*$. Thus $\mathit{pref}(P \cup P') = \left(2^{AP}\right)^*$ and $P \cup P'$ is a liveness property.
- $P \cap P'$ is not a liveness property. A counterexample can be given as follows. Let $AP = \{ a, b \}$ and define
$$\begin{aligned} P &= \mathcal{L}_\omega \left(\left(2^{AP}\right)^* . a^\omega \right) \\ P' &= \mathcal{L}_\omega \left(\left(2^{AP}\right)^* . b^\omega \right). \end{aligned}$$
Then $P \cap P' = \varnothing$; thus $P \cap P'$ is not a liveness property.

Now let P and P' be safety properties.

- $P \cup P'$ is a safety property. By Lemma 3.36 it follows that $\mathit{closure}(P \cup P') = \mathit{closure}(P) \cup \mathit{closure}(P')$. Given that P and P' are safety properties, we have $\mathit{closure}(P) \cup \mathit{closure}(P') = P \cup P'$. Thus $P \cup P'$ is a safety property.
- $P \cap P'$ is a safety property. Let $\sigma \in \left(2^{AP}\right)^\omega \setminus (P \cap P')$; either $\sigma \notin P$ or $\sigma \notin P'$; assume wlog. $\sigma \notin P$. Since P is a safety property, there exists $\hat{\sigma} \in \mathit{pref}(\sigma)$ such that
$$P \cap \left\{ \sigma' \in \left(2^{AP}\right)^\omega \mid \hat{\sigma} \in \mathit{pref}(\sigma') \right\} = \varnothing.$$
Since $P \cap P' \subseteq P$, we infer
$$(P \cap P') \cap \left\{ \sigma' \in \left(2^{AP}\right)^\omega \mid \hat{\sigma} \in \mathit{pref}(\sigma') \right\} = \varnothing.$$
Thus $P \cap P'$ is a safety property.

Answer to Exercise 3.12

(a) We prove $\mathit{closure}(P) \subseteq P_{safe}$:
$$\begin{aligned} P = P_{safe} \cap P_{live} &\implies P \subseteq P_{safe} \\ &\iff \mathit{closure}(P) \subseteq \mathit{closure}(P_{safe}) \\ &\iff \mathit{closure}(P) \subseteq P_{safe} \end{aligned}$$

(b) We have $\mathit{closure}(P) \subseteq P_{safe}$ from part (a) and $P = P_{safe} \cap P_{live}$.
$$\begin{aligned} &\implies \mathit{closure}(P) \cap P_{live} \subseteq P \\ &\implies (\mathit{closure}(P) \cap P_{live}) \cup \left(2^{AP}\right)^\omega \setminus \mathit{closure}(P) \subseteq P \cup \left(2^{AP}\right)^\omega \setminus \mathit{closure}(P) \\ &\iff (*) P_{live} \cup \left(2^{AP}\right)^\omega \setminus \mathit{closure}(P) \subseteq P \cup \left(2^{AP}\right)^\omega \setminus \mathit{closure}(P) \\ &\implies P_{live} \subseteq P \cup \left(2^{AP}\right) \setminus \mathit{closure}(P) \end{aligned}$$

For the equivalence $(*)$, consider $\sigma \in P_{live} \setminus closure(P)$. Then σ is in the set $\left(2^{AP}\right)^{\omega} \setminus closure(P)$.

An alternative proof is the following:

(a) We prove $closure(P) \subseteq P_{safe}$:

$$\begin{aligned} \sigma \in closure(P) \quad &\iff pref(\sigma) \subseteq pref(P) \\ &\Rightarrow pref(\sigma) \subseteq pref(P) \subseteq pref(P_{safe}) \text{ since } P \subseteq P_{safe} \\ &\Rightarrow \sigma \in closure(P_{safe}) \\ &\Rightarrow \sigma \in P_{safe} \text{ since } P_{safe} = closure(P_{safe}) \text{ (safety property)} \end{aligned}$$

(b) We prove $P_{live} \subseteq P \cup \left(\left(2^{AP}\right)^{\omega} \setminus closure(P)\right)$:

Let $\sigma \in P_{live}$. To show: $\sigma \in \left(P \cup \left(\left(2^{AP}\right)^{\omega} \setminus closure(P)\right)\right)$.
We have $P \subseteq P_{live}$ since $P = P_{safe} \cap P_{live}$.

1st case: $\sigma \in P$. Then the proposition holds trivially.

2nd case: $\sigma \in P_{live} \setminus P$.

To show that $\sigma \in \left(\left(2^{AP}\right)^{\omega} \setminus closure(P)\right)$, it suffices to show that $\sigma \notin closure(P)$. We have $\sigma \in P_{live} \setminus P$. Therefore $\sigma \in P_{live}$ and $\sigma \notin P_{safe}$. By $closure(P) \subseteq P_{safe}$ (compare part (a)), we deduce $\sigma \notin closure(P)$.
But then, $\sigma \in \left(2^{AP}\right)^{\omega} \setminus closure(P)$.

Answer to Exercise 3.13
According to the decomposition theorem (see Theorem 3.37, every LT property can be decomposed into a safety and a liveness property in the following way:

$$P = \underbrace{closure(P)}_{P_{safe}} \cap \underbrace{\left(P \cup \left(\left(2^{AP}\right)^{\omega} \setminus closure(P)\right)\right)}_{P_{live}}$$

The linear time property P over $AP = \{a, b\}$ can be characterized by the following ω-regular expression:

$$P = \mathcal{L}_{\omega}\left(\{a\}^*\{a,b\}\left(\left(2^{AP}\right)^* \{\{b\},\{a,b\}\}\right)^{\omega}\right)$$

In our case, this yields the following safety property:

$$\begin{aligned} P_{safe} &= closure(P) \\ &= \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid pref(\sigma) \subseteq pref(P)\right\} \\ &= \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid pref(\sigma) \subseteq \mathcal{L}\left(\{a\}^*\{a,b\}\left(2^{AP}\right)^* + \{a\}^*\right)\right\} \\ &= \mathcal{L}_{\omega}\left(\{a\}^*\{a,b\}\left(2^{AP}\right)^{\omega} + \{a\}^{\omega}\right) \end{aligned}$$

The liveness property $P_{live} = \Big(P \cup \Big(\Big(2^{AP}\Big)^{\omega} \setminus \mathit{closure}(P)\Big)\Big)$ can be deduced as follows:

$$\begin{aligned} P_{live} &= P \cup \Big(\Big(2^{AP}\Big)^{\omega} \setminus P_{safe}\Big) \\ &= P \cup \Big(\Big(2^{AP}\Big)^{\omega} \setminus \mathcal{L}_{\omega}\Big(\{a\}^*\{a,b\}\Big(2^{AP}\Big)^{\omega} + \{a\}^{\omega}\Big)\Big) \\ &= P \cup \mathcal{L}_{\omega}\Big(\{a\}^*\{\varnothing,\{b\}\}\Big(2^{AP}\Big)^{\omega}\Big) \\ &= \mathcal{L}_{\omega}\Big(\{a\}^*\{a,b\}\Big(\Big(2^{AP}\Big)^{*}\{\{b\},\{a,b\}\}\Big)^{\omega} + \{a\}^*\{\varnothing,\{b\}\}\Big(2^{AP}\Big)^{\omega}\Big) \end{aligned}$$

Answer to Exercise 3.14
The semaphore-based mutual exclusion algorithm violates the absence of starvation property. Consider the following path:

$$\pi = (nc_1, nc_2, 1)(w_1, nc_2, 1)\Big((w_1, w_2, 1)(c_1, w_2, 0)(nc_1, w_2, 1)\Big)^{\omega}$$

The corresponding trace is

$$\varnothing\{\mathit{wait}_1\}\Big(\{\mathit{wait}_1, \mathit{wait}_2\}\{\mathit{crit}_1, \mathit{wait}_2\}\{\mathit{wait}_2\}\Big)^{\omega}$$

Process 2 thus waits infinitely long to acquire access to the critical tion.

There is no corresponding path in the transition system TS_{Pet}, and thus does starvation does not occur in TS_{Pet}.

Answer to Exercise 3.15

	E_b	E_a	E'
$\mathcal{F}_1 = (\varnothing, \{\{\alpha\}\}, \varnothing)$		yes	
$\mathcal{F}_2 = (\varnothing, \{\{\alpha, \beta\}\}, \varnothing)$			
$\mathcal{F}_3 = (\varnothing, \{\{\beta\}\}, \varnothing)$	yes		
$\mathcal{F}'_1 = (\varnothing, \varnothing, \{\{\alpha\}\})$		yes	
$\mathcal{F}'_2 = (\varnothing, \varnothing, \{\{\alpha, \beta\}\})$			
$\mathcal{F}'_3 = (\varnothing, \varnothing, \{\{\beta\}\})$			

(a) Argumentation for some of the cases of strong fairness constraints:

- $\mathcal{F}_1 = (\varnothing, \{\{\alpha\}\}, \varnothing)$ and E_a: as s_1 is visited infinitely often, α is enabled infinitely often. $\Longrightarrow$ On each trace $\sigma \in \mathit{FairTraces}_{\mathcal{F}_1}(TS)$, α is executed infinitely often. For each α transition it holds: Either the current state or its α-successor state has a in its atomic proposition. Therefore on each fair trace, a appears infinitely often. $\Longrightarrow$ $TS \models_{\mathcal{F}_1} E_a$
- $\mathcal{F}_2 = (\varnothing, \{\{\alpha, \beta\}\}, \varnothing)$ and E_a: we have $TS \not\models_{\mathcal{F}_2} E_a$. This can be seen as follows. Consider $\pi = (s_1 s_3)^{\omega}$. π is a $\mathcal{F}_2$ fair path in TS. But $\mathit{trace}(\pi) = (\varnothing\{b\})^{\omega} \notin E_a$. Therefore $\mathit{FairTraces}_{\mathcal{F}_2}(TS) \not\subseteq E_a$.

- $\mathcal{F}_3 = (\varnothing, \{\{\beta\}\}, \varnothing)$ and E_b. As every infinite path passes infinitely often through s_1, β is enabled infinitely often. Thus, β is executed infinitely often on paths in $\mathit{FairPaths}_{\mathcal{F}_3}(TS)$. $\Longrightarrow$ we visit s_3 infinitely often (as there is only one β transition in TS). $\Longrightarrow$ we see $\{b\}$ infinitely often. $\Longrightarrow \mathit{FairTraces}_{\mathcal{F}_3}(TS) \subseteq E_b$

(b) Argumentation for some of the cases when imposing weak fairness constraints:

- $\mathcal{F}_1' = (\varnothing, \varnothing, \{\{\alpha\}\})$ and E_b. We have $(s_1 s_2)^\omega \in \mathit{FairTraces}_{\mathcal{F}_1}(TS)$ since a is not continuously enabled in state s_1. Further, the corresponding trace σ is defined by $\mathit{trace}\,((s_1 s_2)^\omega) = (\varnothing\{a\})^\omega$. But then, $\sigma \notin E_b$. Therefore $\mathit{FairTraces}_{\mathcal{F}_1'}(TS) \not\subseteq E_b$ and $TS \not\models_{\mathcal{F}_1'} E_b$.
- $\mathcal{F}_1' = (\varnothing, \varnothing, \{\{\alpha\}\})$ and E_a: To falsify $TS \models_{\mathcal{F}_1'} E_a$, there must be a trace $\sigma \in \mathit{FairTraces}_{\mathcal{F}_1'}(TS)$ (with corresponding path π) such that there are only finitely many a's in σ. Thus, π, s_2 and s_4 may be visited only finitely often. $\Longrightarrow$ π loops from some point onwards only between states s_1 and s_3. But then, α is infinitely often (in a row) enabled (s_1 and s_3 have outgoing α transitions). According to the weak fairness constraint $\{\alpha\}$, s_2 or s_4 must be visited infinitely often. This yields a contradiction.

It remains to consider the LT-property E' which is a safety property with

$$\mathit{BadPref}(E') = \left(2^{AP}\right)^* \{a\}\{a,b\}\varnothing \left(2^{AP}\right)^*.$$

TS is realizable fair for $\mathcal{F}_i$ and $\mathcal{F}_i'$ for $1 \leqslant i \leqslant 3$. Therefore

$$TS \models E' \quad \Longleftrightarrow \quad TS \models_{\mathcal{F}} E' \qquad \text{for } \mathcal{F} \in \{\mathcal{F}_i, \mathcal{F}_i' \mid 1 \leqslant i \leqslant 3\}$$

But because of the path $s_1 s_2 s_4 s_1 \ldots$ with trace $\varnothing\{a\}\{a,b\}\varnothing \ldots$, $TS \not\models E'$. Thus, $TS \not\models_{\mathcal{F}} E'$.

Answer to Exercise 3.17
Let $\mathcal{F} = (\varnothing, \{\{\alpha\}, \{\beta\}\}, \{\{\beta\}\})$ be the fairness constraint and $P =$ "eventually a" be the LT-property under consideration. Then $TS \not\models_{\mathcal{F}} P$:

Argument:

$$TS \models_{\mathcal{F}} P \quad \Longleftrightarrow \quad \mathit{FairTraces}_{\mathcal{F}}(TS) \subseteq P.$$

Consider the path $\pi = (s_3 s_4 s_5 s_4 s_6)^\omega$. It is $\mathcal{F}$-fair since α and β are enabled and triggered infinitely often. As β is not always enabled (s_5 has no outgoing β transition), the weak fairness constraint imposes no restriction on this path. Thus, $(s_3 s_4 s_5 s_4 s_6)^\omega \in \mathit{FairTraces}_{\mathcal{F}}(TS)$. On the other hand, $\mathit{trace}((s_3 s_4 s_5 s_4 s_6)^\omega) = \varnothing^\omega$. Therefore $\mathit{trace}(\pi)$ is not in P:

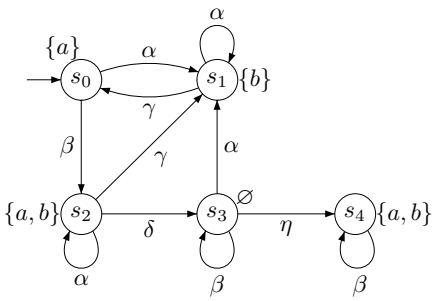
$$\begin{array}{ll} \Longrightarrow & \mathit{FairTraces}_{\mathcal{F}}(TS) \not\subseteq P \\ \Longrightarrow & TS \not\models_{\mathcal{F}} P. \end{array}$$

Answer to Exercise 3.18
Realizable fairness assumptions:

1. $\mathcal{F}_1 = (\{\{\alpha\}\}, \{\{\delta\}\}, \{\{\alpha, \beta\}\})$ is not realizable fair. Consider the states s_1 and s_4. There are no $\mathcal{F}_1$ fair path fragments starting from s_1 or s_4, as on each such path fragment, α transitions never occur. This violates the unconditional fairness constraint $\{\{\alpha\}\}$.
2. $\mathcal{F}_2 = (\{\{\alpha, \delta\}\}, \{\{\alpha, \beta\}\}, \{\{\delta\}\})$ is realizable fair, as the SCC $\{s_1, s_4\}$ is reachable from every state and $(s_1 s_4)^\omega$ is a $\mathcal{F}_2$ fair path fragment.
3. $\mathcal{F}_3 = (\{\{\alpha, \delta\}, \{\beta\}\}, \{\{\alpha, \beta\}\}, \{\{\delta\}\})$ is realizable fair. Consider the same SCC $\{s_1, s_4\}$ and again the path fragment $(s_1 s_4)^\omega$.

Answer to Exercise 3.19(b)
We consider each of the fairness assumptions $\mathcal{F}_i$ for $i \in \{1, 2\}$:

We have $TS \models_{\mathcal{F}_i} P$ iff $\mathit{FairTraces}_{\mathcal{F}_i}(TS) \subseteq P$. Because of $\overset{\infty}{\exists}\ k.\ A_k = \{a, b\}$, each trace has to visit at least one of s_2 or s_4 infinitely many times. Additionally, from some point onwards, each a-state must be followed by a state that is annotated with (at least) b.



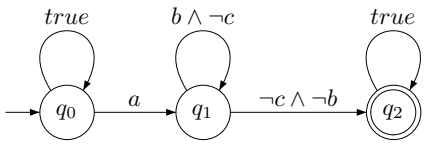
1. $TS \models_{\mathcal{F}_1} P_2$:
   - Any trace that reaches s_4 is not $\mathcal{F}_1$-fair as α is executed only finitely many times. This is in contradiction to $\mathcal{F}_{1,ucond} = \big\{\{\alpha\}\big\}$.
   - Therefore $s_3 \xrightarrow{\eta} s_4$ is never taken.
   - Because of $\{\eta\} \in \mathcal{F}_{1,strong}$ and because η actions cannot be executed infinitely often (in fact, only once from s_3 to s_4), the state s_3 must not be visited infinitely often.
   - We cannot stay in states s_1 or s_2 by only taking transitions $s_1 \xrightarrow{\alpha} s_1$ and $s_2 \xrightarrow{\alpha} s_2$ because of the enabled γ transitions to s_0 or s_1, respectively.
   - As β is enabled in s_0, all $\mathcal{F}_1$–fair paths visit exactly s_0, s_1 and s_2 infinitely often.

   Therefore $\mathit{FairTraces}_{\mathcal{F}_1}(TS) \subseteq P_2$ and $TS \models_{\mathcal{F}_1} P_2$.
2. $TS \not\models_{\mathcal{F}_2} P_2$:
   Consider the path $\pi = (s_0 s_2 s_3 s_1)^\omega$ with its corresponding trace $\sigma = (\{a\}\{a, b\}\varnothing\{b\})^\omega$.
   We have $\pi \in \mathit{FairPaths}_{\mathcal{F}_2}(TS)$, but $\sigma \notin P_2$.
   $\Longrightarrow \mathit{FairTraces}_{\mathcal{F}_2}(TS) \not\subseteq P_2$.

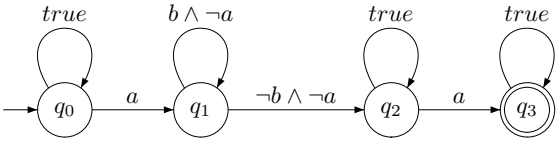
Exercises of Chapter 4

Answer to Exercise 4.1

(a) If a becomes valid, afterwards b stays valid ad infinitum or until c holds. This is a regular safety property. Its set of bad prefixes is accepted by the following NFA $\mathcal{A}_1$:



(b) Between two neighbouring occurrences of a, b always holds. Again, this is a regular safety property. The bad prefixes are accepted by the finite automaton $\mathcal{A}_2$:



(c) Between two neighbouring occurrences of a, b occurs more often than c. This is a safety property, but not a regular one: assume there exists a DFA $\mathcal{A}_3 = \big(Q, 2^{\{a,b,c\}}, \delta, Q_0, F\big)$ with $\mathcal{L}(\mathcal{A}_3) = \mathit{BadPref}(P_3)$. Let $|Q| = n$. Consider the bad prefix

$$w = \{a\}\{b\}^n\{c\}^n\{a\} \in \mathit{BadPref}(P_3)$$

Then, $w \in \mathcal{L}(\mathcal{A}_3)$ with an accepting run $\rho = q_0 q_1 q_2 \dots q_{n+1}\rho'$ which is uniquely determined (we consider an DFA). There are $n+1$ states visited during the run on infix $\{b\}^n$, namely the states $q_1, \dots, q_{n+1}$.

Because $|Q| = n$, there exists $1 \leqslant i < j \leqslant n{+}1$ such that $q_i = q_j$. Therefore, we can construct the run:

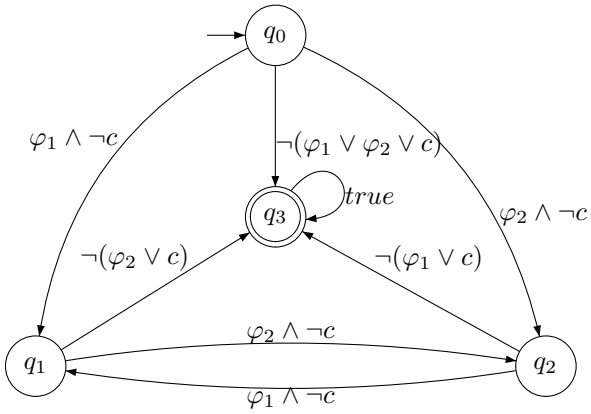
$$q_0 q_1 \dots q_i \underbrace{(q_{i+1} \dots q_j = q_i)}_{\text{inserted}} q_{i+1} \dots q_j q_{j+1} \dots q_{n+1}\rho'$$

which is also accepting (as the suffix ρ' leads into a final state). The corresponding input word has the form

$$w' = \{a\}\{b\}^{n+\overbrace{j-i}^{\geqslant 1}}\{c\}^n\{a\} \in \mathcal{L}(\mathcal{A}_3)$$

This yields a contradiction since w' is not a bad prefix (between the two a-occurrences, b occurs more often than c) but $w' \in \mathcal{L}(\mathcal{A}_3) = \mathit{BadPref}(P_3)$.

(d) $a \wedge \neg b$ and $b \wedge \neg a$ are valid in alternation or until c becomes valid. This is a regular safety property. Its set of bad prefixes is accepted by the following NFA $\mathcal{A}_4$. Let $\varphi_1 = a \wedge \neg b$ and $\varphi_2 = b \wedge \neg a$:



Answer to Exercise 4.2

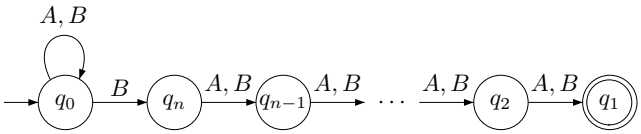
(a) Formally, we define the NFA $\mathcal{A}_n = (Q_n, \Sigma, \delta_n, Q_0, F)$ where

- $Q_n = \{q_0, q_n, q_{n-1}, \ldots, q_1\}$
- transition relation defined by δ_n:

$$\begin{array}{llll} \delta_n(q_0, A) & = \{q_0\} & \delta_n(q_0, B) & = \{q_0, q_n\} \\ \delta_n(q_i, A) & = \{q_{i-1}\} \text{ for } 1 < i \leqslant n & \delta_n(q_i, B) & = \{q_{i-1}\} \text{ for } 1 < i \leqslant n \end{array}$$

- the set of initial states: $Q_0 = \{q_0\}$
- $F = \{q_1\}$

This can also be outlined as follows:



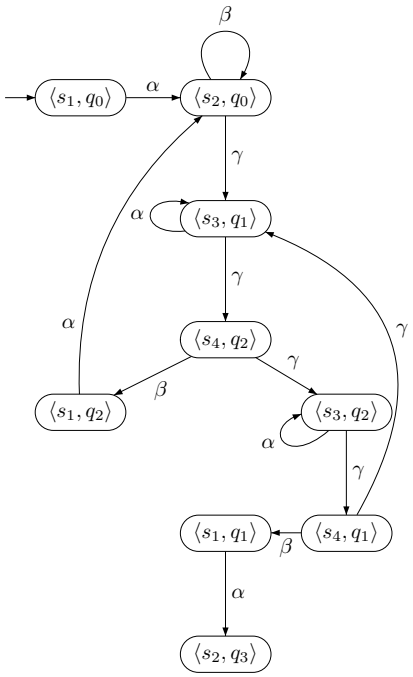
(b) Determinization of $\mathcal{A}_n$ yields the DFA $\mathcal{A}'_n = \big(2^{Q_n}, \Sigma, \delta'_n, \{q_0\}, F'_n\big)$ where

- the transition function δ'_n is defined (for $k \in \{0, \ldots, n\}$) as follows:

$$\begin{array}{ll} \delta'_n(\{q_0, q_{i_1}, \ldots, q_{i_k}\}, A) & = \big\{q_{i_j - 1} \mid i_j > 1 \text{ where } j \in \{1, \ldots, k\}\big\} \uplus \{q_0\} \\ \delta'_n(\{q_0, q_{i_1}, \ldots, q_{i_k}\}, B) & = \big\{q_{i_j - 1} \mid i_j > 1 \text{ where } j \in \{1, \ldots, k\}\big\} \uplus \{q_0, q_n\} \end{array}$$

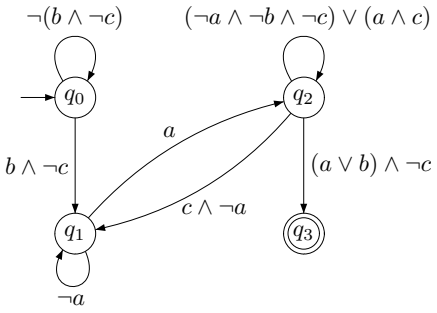
- The acceptance set is given by $F'_n = \{Q' \in 2^{Q_n} \mid q_1 \in Q'\}$

Answer to Exercise 4.5
The product $TS \otimes \mathcal{A}$ is the following transition system:

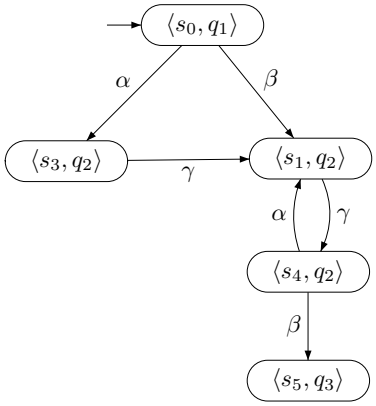


Answer to Exercise 4.6

(a) An NFA that accepts the set of minimal bad prefixes of P_{safe} is:



(b) First we apply the $TS \otimes \mathcal{A}$ construction which yields:



A counterexample to $TS \models P_{safe}$ is given by the following initial path fragment in $TS \otimes \mathcal{A}$:

$$\pi_{\otimes} = \langle s_0, q_1 \rangle \langle s_3, q_2 \rangle \langle s_1, q_2 \rangle \langle s_4, q_2 \rangle \langle s_5, q_3 \rangle$$

By projection on the state components, we get a path in the underlying transition system:

$$\pi = s_0 s_3 s_1 s_4 s_5 \quad \text{with} \quad \mathit{trace}(\pi) = \{a,b\}\{a,c\}\{a,b,c\}\{a,c\}\{a,b\}$$

Obviously, $\mathit{trace}(\pi) \in \mathit{BadPref}(P_{safe})$, so we have $\mathit{Traces}_{fin}(TS) \cap \mathit{BadPref}(P_{safe}) \neq \varnothing$. By Lemma 3.25, this yields $TS \not\models P_{safe}$.

Answer to Exercise 4.7
Prove or refute the following equivalences for ω-regular expressions:

(a) $(E_1 + E_2).F^\omega \equiv E_1.F^\omega + E_2.F^\omega$
True, since:

$$\begin{aligned}
\mathcal{L}_\omega((E_1 + E_2).F^\omega) &= \mathcal{L}(E_1 + E_2).\mathcal{L}(F)^\omega \\
&= \big(\mathcal{L}(E_1) \cup \mathcal{L}(E_2)\big).\mathcal{L}(F)^\omega \\
&= \mathcal{L}(E_1).\mathcal{L}(F)^\omega \cup \mathcal{L}(E_2).\mathcal{L}(F)^\omega \\
&= \mathcal{L}_\omega(E_1.F^\omega) \cup \mathcal{L}_\omega(E_2.F^\omega) \\
&= \mathcal{L}_\omega(E_1.F^\omega + E_2.F^\omega)
\end{aligned}$$

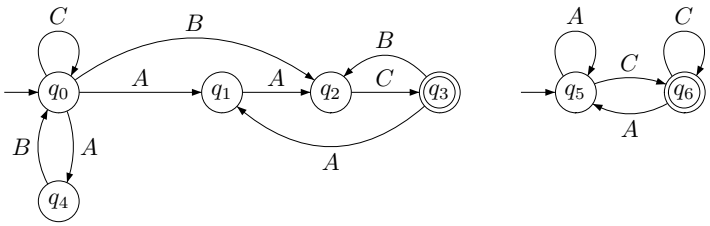
(b) $E.(F_1 + F_2)^\omega \equiv E.F_1^\omega + E.F_2^\omega$
False: Consider $E = \underline{\varepsilon}$ and $F_1 = A$, $F_2 = B$ where $\underline{\varepsilon}$ denotes the language consisting of the empty word only, i.e. $\{\varepsilon\}$.
Then $\mathcal{L}_\omega(E.(F_1 + F_2)^\omega) = \{A, B\}^\omega$, but $(AB)^\omega \notin \mathcal{L}_\omega(E.F_1^\omega + E.F_2^\omega) = \{A^\omega, B^\omega\}$.

(c) $E.(F.F^*)^\omega \equiv E.F^\omega$, since:

$$\begin{aligned}
\mathcal{L}_\omega(E.(F.F^*)^\omega) &= \mathcal{L}(E).\mathcal{L}(F.F^*)^\omega \\
&= \mathcal{L}(E).\mathcal{L}(F^+)^\omega \\
&= \mathcal{L}(E).\left(\{w_0w_1w_2\dots w_k \mid k>0 \wedge w_i \in \mathcal{L}(F) \text{ for all } i \in \{0,\dots,k\}\}\right)^\omega \\
&= \mathcal{L}(E).\{v_1v_2\dots \mid v_i \in \mathcal{L}(F^+)\} \\
&= \mathcal{L}(E).\{w_{1,1}w_{1,2}\dots w_{1,k_1}w_{2,1}\dots w_{2,k_2}w_{3,1}\dots \mid w_{i,j_i} \in \mathcal{L}(F)\ \forall i \geqslant 1 \wedge \forall j_i \in \{1,\dots,k_i\}\} \\
&= \mathcal{L}(E).\mathcal{L}(F)^\omega \\
&= \mathcal{L}_\omega(E.F^\omega).
\end{aligned}$$

(d) $(E^*.F)^\omega \not\equiv E^*.F^\omega$. As a counterxample, consider $E = A$, $F = B$. Then, $(AB)^\omega \in \mathcal{L}_\omega((E^*.F)^\omega)$ but $(AB)^\omega \notin \mathcal{L}_\omega(E^*.F^\omega)$.

Answer to Exercise 4.11



Note: We allow more than one initial state! Formally, the automaton outlined above is given by

$\mathcal{A}_2 = (\{q_0,\dots,q_6\},\{A,B,C\},\delta,\{q_0,q_5\},\{q_3,q_6\})$ where δ is defined as above.

Answer to Exercise 4.12

(a) $\mathcal{L}_\omega(\mathcal{A}_1) = L_{q_0q_0}.L_{q_0q_0}^\omega = \mathcal{L}\left(C(A+B+C)^*C + A(A+B+C)^*A\right)^\omega$

(b) For $\mathcal{A}_2$, we have $F = \{\, q_1, q_3 \,\}$:

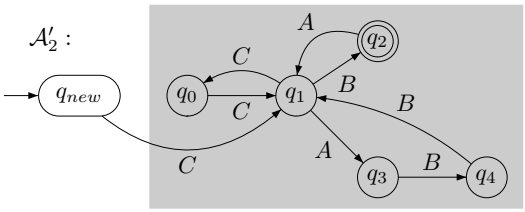
$$\begin{aligned}
L_{q_0q_1} &= (B+C)^*A(BC)^* \\
L_{q_0q_3} &= (B+C)^*(B + A(BC)^*B)A^* \\
L_{q_1q_1} &= (BC)^* \\
L_{q_3q_3} &= A^*
\end{aligned}$$

The language accepted by $\mathcal{A}_2$ then is

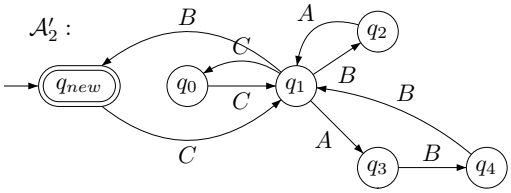
$$\begin{aligned}
\mathcal{L}_\omega(\mathcal{A}_2) &= \textstyle\bigcup_{q\in F, q_0\in Q_0} L_{q_0q}.(L_{qq} \setminus \{\varepsilon\})^\omega \\
&= L_{q_0q_1}.(L_{q_1q_1} \setminus \{\varepsilon\})^\omega \ \cup\ L_{q_0q_3}.(L_{q_3q_3} \setminus \{\varepsilon\})^\omega \\
&= \mathcal{L}_\omega\big(((B+C)^*A(BC)^*).((BC)^+)^\omega + ((B+C)^*(B+A(BC)^*B)A^*).(A^+)^\omega\big) \\
&= \mathcal{L}_\omega\big((B+C)^*A(BC)^\omega + (B+C)^*(B+A(BC)^*B)A^\omega\big)
\end{aligned}$$

Answer to Exercise 4.13
We construct the corresponding automaton in two steps:

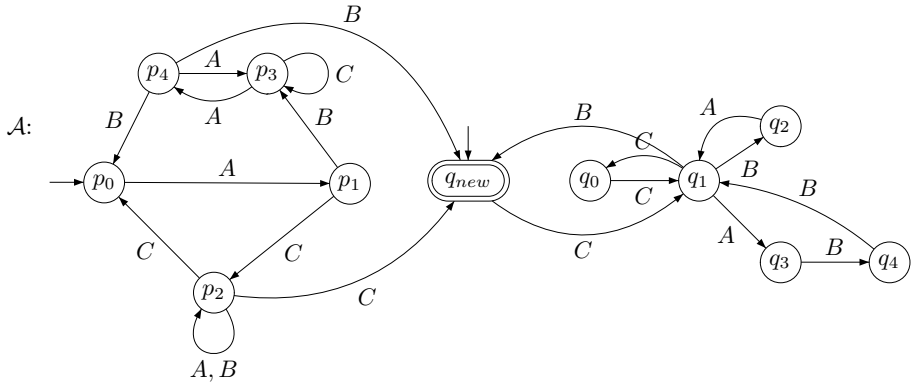
- First, we construct an NBA that accepts the language $\mathcal{L}(\mathcal{A}_2)^\omega$: As there exists an incoming transition into the initial state, we have to preprocess the NFA $\mathcal{A}_2$ by introducing a new initial state q_{new} which is non-final:



  Starting from $\mathcal{A}_2'$, we now construct the NBA $\mathcal{A}_2''$ that accepts $\mathcal{L}(\mathcal{A}_2)^\omega$:
  Therefore, for each state which has an outgoing transition into a final state, we add a corresponding transition that goes back to the initial state. The only accepting states are the initial states of this new automaton:



- In the second step, we construct the NBA $\mathcal{A}$ that accepts the concatenation $\mathcal{L}(\mathcal{A}_1).\mathcal{L}(\mathcal{A}_2)^\omega$:

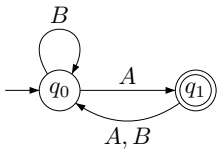


Answer to Exercise 4.22
To show that the class of DBA-acceptable languages is not closed under complementation, consider

the following ω-regular language over $\Sigma = \{A, B\}$:

$$L = \mathcal{L}_\omega\big(((A + B)^* A)^\omega\big)$$

It is recognizable by the following DBA:



The fact that its complement language $\bar{L} = \{A, B\}^\omega \setminus L = \mathcal{L}_\omega\big((A + B)^* B^\omega\big)$ is not recognizable by a DBA directly follows from Theorem 4.50.

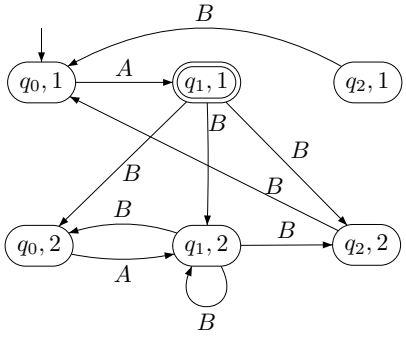
Answer to Exercise 4.24
The acceptance condition for GNBA $\mathcal{A} = (Q, \Sigma, \delta, Q_0, \mathcal{F})$ with $\mathcal{F} = \{F_1, \ldots, F_n\}$ and $F_i \subseteq Q$ for $(1 \leqslant i \leqslant n)$:

$$\mathcal{A} \text{ accepts } \alpha \in \Sigma^\omega \iff \text{ex. infinite run } \rho \text{ of } \mathcal{A} \text{ on } \alpha \text{ s.t. } \forall F \in \mathcal{F}.\ \Big(\overset{\infty}{\exists}\ j \geqslant 0.\ \rho[j] \in F\Big)$$

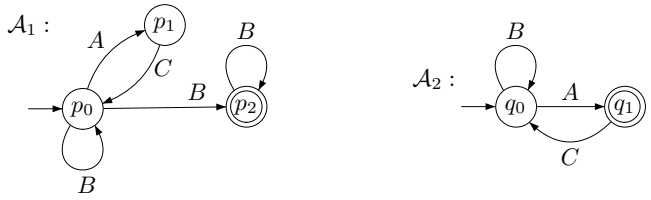
Using the construction from Theorem 4.56, we infer the NBA $\mathcal{A}' = (Q', \Sigma, \delta', Q'_0, F)$ where

- $Q' = Q \times \{1, 2\}$
- $\delta'((q, i), A) = \begin{cases} \{(q', i) \mid q' \in \delta(q, A)\} & \text{if } q \notin F_i \\ \{(q', (i \bmod 2) + 1) \mid q' \in \delta(q, A)\} & \text{otherwise} \end{cases}$
- $Q'_0 = \{(q_0, 1)\}$
- $F = \{(q_1, 1)\}$

The NBA for the given GNBA can be outlined as follows:



Answer to Exercise 4.25
NBA $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, Q_{0,1}, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, Q_{0,2}, F_2)$ for the languages $(AC{+}B)^*B^\omega$ and $(B^*AC)^\omega$, respectively are:



The corresponding GNBA are given by:

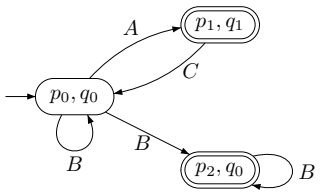
$$\mathcal{G}_1 = (Q_1, \Sigma, \delta_1, Q_{0,1}, \{F_1\}) \quad \text{and} \quad \mathcal{G}_2 = (Q_2, \Sigma, \delta_2, Q_{0,2}, \{F_2\})\,.$$

Applying the product construction (cf. Lemma 4.59) yields the GNBA

$$\mathcal{G} = (Q_1 \times Q_2, \Sigma, \delta, Q_{0,1} \times Q_{0,2}, \mathcal{F}) \quad \text{where}$$

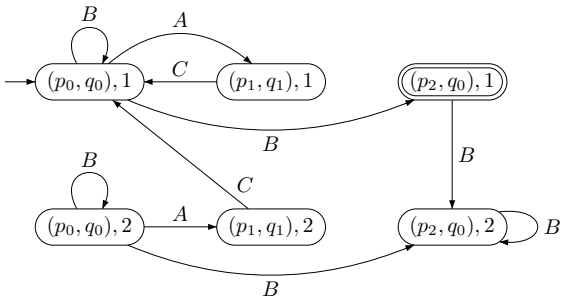
- $\delta((p,q), A) = \delta_1(p, A) \times \delta_2(q, A)$, and
- $\mathcal{F} = \{F_1 \times Q_2\} \cup \{Q_1 \times F_2\} = \{\{(p_2, q_0), (p_2, q_1)\}, \{(p_0, q_1), (p_1, q_1), (p_2, q_1)\}\}$

The GNBA $\mathcal{G}$ can be outlined as follows (only reachable states are indicated):



Let us argue why $\mathcal{L}_\omega(\mathcal{G}) = \varnothing$. GNBA $\mathcal{G}$ accepts an input word α if and only if for each $F \in \mathcal{F}$ some states are visited infinitely often. But as soon as (p_2, q_0) is visited —the only state in F_1— the accept set F_1 is not reachable any longer, and thus cannot be visited infinitely often. Therefore $\mathcal{G}$ accepts the empty language.

Given $\mathcal{G}$, we can construct an equivalent NBA $\mathcal{A}$. This yields:



Again, on each possible run, the state $((p_2, q_0), 2)$ of $\mathcal{A}$ can be visited only once. Therefore also $\mathcal{L}_\omega(\mathcal{A}) = \varnothing$ holds.

Answer to Exercise 4.26

(a) The language accepted by the Muller automaton is given by the following ω-regular expression:

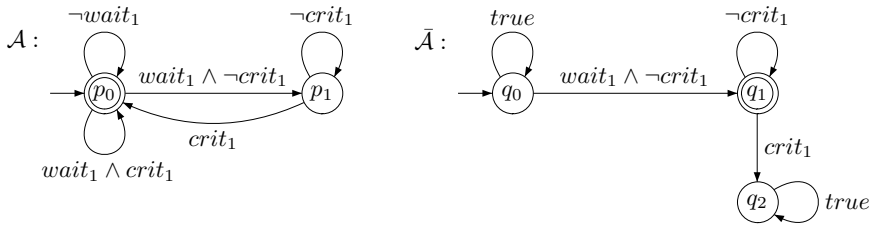
$$\begin{aligned} E &= \big(BB + (BB + CC)^* C(AA + CC)^* C\big)^* \\ \mathcal{L}_\omega(\mathcal{A}) &= \mathcal{L}_\omega\left(E.C^\omega + E.CA^\omega\right) \end{aligned}$$

(b) Let $\mathcal{G} = (Q, \Sigma, \delta, Q_0, \mathcal{F})$ with $\mathcal{F} = \{F_1, \ldots, F_k\}, F_i \subseteq Q, 1 \leqslant i \leqslant k$ be an GNBA. Construct the equivalent nondeterministic Muller automaton $\mathcal{A} = (Q, \Sigma, \delta, Q_0, \mathcal{F}')$ with

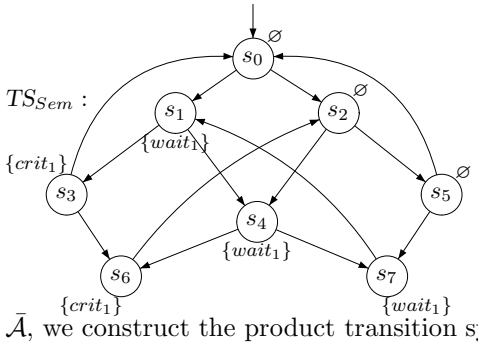
$$\mathcal{F}' := \left\{ F \subseteq Q \mid \bigwedge_{i=1}^{k} F_i \cap F \neq \varnothing \right\}$$

Answer to Exercise 4.27

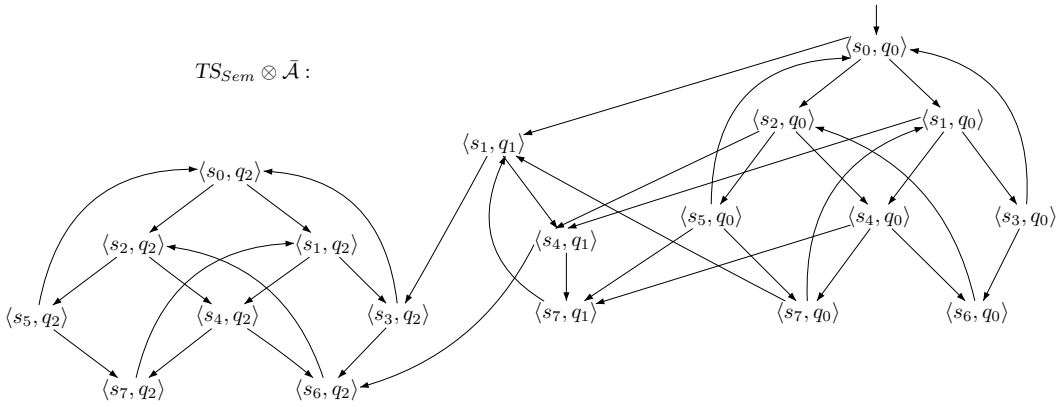
(a) The NBA $\mathcal{A}$ and $\bar{\mathcal{A}}$ for P_{live} and $\bar{P}_{live} = \left(2^{AP}\right)^\omega \setminus P_{live}$, respectively are:



(b) (I) Construct the transition system $TS_{Sem} \otimes \bar{\mathcal{A}}$. The transition system TS_{Sem} can be outlined as follows:



Based on TS_{Sem} and $\bar{\mathcal{A}}$, we construct the product transition system:



(II) To prove $TS_{Sem} \models P_{live}$, we check the persistence property P_{pers} = "eventually forever Φ" (where $F = \{q_1\}$ and $\Phi = \neg F$) for the transition system $TS_{Sem} \otimes \bar{\mathcal{A}}$. Using the nested depth-first search algorithm, we search for a reachable cycle in $TS_{Sem} \otimes \bar{\mathcal{A}}$ containing at least one $\neg\Phi$-state (i.e., a state from F). The algorithm yields, denoting the stack content from left to right, the top element on the left:

- Initial state: $\langle s_0, q_0 \rangle$
- 1st descent:

$$U = \langle s_0, q_2 \rangle \langle s_3, q_2 \rangle \langle s_1, q_2 \rangle \langle s_7, q_2 \rangle \langle s_4, q_2 \rangle \langle s_2, q_2 \rangle \langle s_6, q_2 \rangle \langle s_4, q_1 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$$

- $\langle s_0, q_2 \rangle, \langle s_3, q_2 \rangle, \langle s_1, q_2 \rangle, \langle s_7, q_2 \rangle, \langle s_4, q_2 \rangle$ are popped from the stack as they have no successor states that are not visited yet and their state component is not a final state of $\bar{\mathcal{A}}$. This yields:

$$U = \langle s_2, q_2 \rangle \langle s_6, q_2 \rangle \langle s_4, q_1 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$$

- 2nd descent: $U = \langle s_5, q_2 \rangle \langle s_2, q_2 \rangle \langle s_6, q_2 \rangle \langle s_4, q_1 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$. Again, all successor states of $\langle s_5, q_2 \rangle$ are already visited ($\in R$), therefore $\langle s_5, q_2 \rangle, \langle s_2, q_2 \rangle$ and $\langle s_6, q_2 \rangle$ are popped. This results in:

$$U = \langle s_4, q_1 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$$

- 3rd descent: The successor state $\langle s_7, q_1 \rangle$ of $\langle s_4, q_1 \rangle$ is not visited yet:

$$U = \langle s_1, q_1 \rangle \langle s_7, q_1 \rangle \langle s_4, q_1 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$$

- Now, all successor states of $\langle s_1, q_1 \rangle$ are already visited; however, since $\langle s_1, q_1 \rangle \not\models \Phi$ ($q_1 \in F$), we start a nested depth first search from here looking for a backward edge to $\langle s_1, q_1 \rangle$ and pop $\langle s_1, q_1 \rangle$ from U:

$$U = \langle s_7, q_1 \rangle \langle s_4, q_1 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$$

  - **cycle_check(**$\langle s_1, q_1 \rangle$**)** with $V = \varepsilon$ and $T = \varnothing$:

$$V = \langle s_7, q_1 \rangle \langle s_4, q_1 \rangle \langle s_1, q_1 \rangle$$

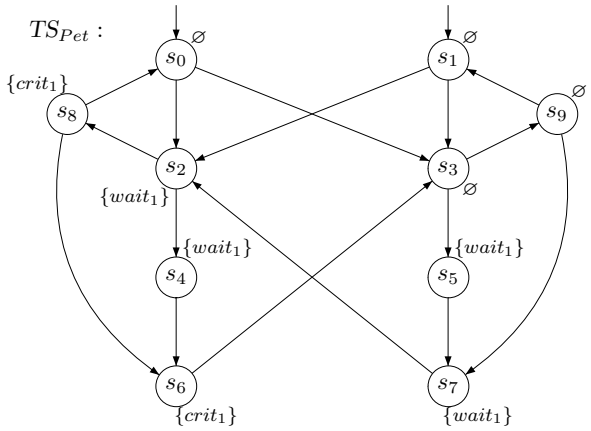
  - $Post(\langle s_7, q_1 \rangle) = \{\langle s_1, q_1 \rangle\}$ and therefore we found a backward edge to $\langle s_1, q_1 \rangle$:

$$V = \langle s_1, q_1 \rangle \langle s_7, q_1 \rangle \langle s_4, q_1 \rangle \langle s_1, q_1 \rangle$$

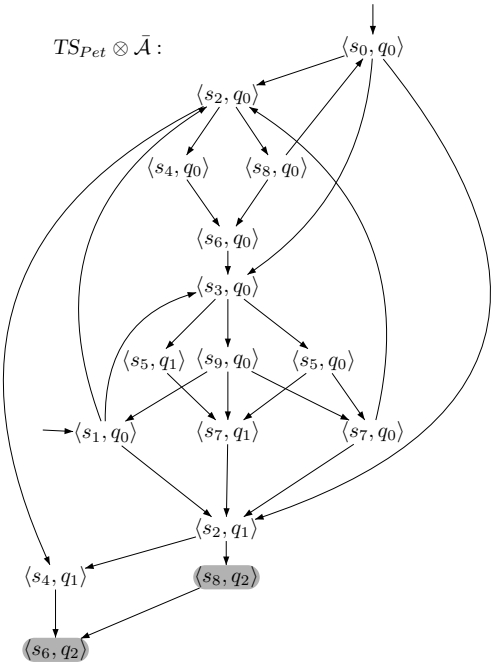
The generated counterexample now is:

$$Reverse(V \cdot U) = \langle s_0, q_0 \rangle \langle s_2, q_0 \rangle \langle s_4, q_1 \rangle \langle s_7, q_1 \rangle \langle s_1, q_1 \rangle \langle s_4, q_1 \rangle \langle s_7, q_1 \rangle \langle s_1, q_1 \rangle \ldots$$

(c) The Peterson mutual exclusion protocol is described by the following transition system (only over the atomic propositions $AP = \{wait_1, crit_1\}$):



Again, we outline the product transition system $TS_{Pet} \otimes \bar{\mathcal{A}}$:



Here, the gray states represent the part of $TS_{Pet} \otimes \bar{\mathcal{A}}$ that is induced by the loop-state q_2 of $\bar{\mathcal{A}}$. As the only outgoing transition of q_2 is a loop back to q_2 and q_2 is non-final, we cannot find a cycle in this part of the transition system where we visit q_1 (i.e. a final state) infinitely often.

The nested depth-first search yields:

- 1st initial state: $\langle s_0, q_0 \rangle$:
- 1st descent yields:

$$\begin{aligned} U = \quad & \langle s_0, q_2 \rangle \langle s_8, q_2 \rangle \langle s_2, q_2 \rangle \langle s_7, q_2 \rangle \langle s_5, q_2 \rangle \langle s_3, q_2 \rangle \langle s_6, q_2 \rangle \langle s_4, q_1 \rangle \\ & \langle s_2, q_1 \rangle \langle s_7, q_1 \rangle \langle s_5, q_1 \rangle \langle s_3, q_0 \rangle \langle s_6, q_0 \rangle \langle s_4, q_0 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle \end{aligned}$$

- As all successor states ($\langle s_2, q_2 \rangle$ and $\langle s_3, q_2 \rangle$) of $\langle s_0, q_2 \rangle$ are already visited (*i.e.* $\in R$), we check whether $\langle s_0, q_2 \rangle \models \neg\Phi$. This is not the case, therefore we skip the nested DFS and track back:

$$\begin{aligned} U = \quad & \langle s_2, q_2 \rangle \langle s_7, q_2 \rangle \langle s_5, q_2 \rangle \langle s_3, q_2 \rangle \langle s_6, q_2 \rangle \langle s_4, q_1 \rangle \langle s_2, q_1 \rangle \\ & \langle s_7, q_1 \rangle \langle s_5, q_1 \rangle \langle s_3, q_0 \rangle \langle s_6, q_0 \rangle \langle s_4, q_0 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle \end{aligned}$$

- The successor $\langle s_4, q_2 \rangle$ of $\langle s_2, q_2 \rangle$ has not been visited yet:

$$\begin{aligned} U = \quad & \langle s_4, q_2 \rangle \langle s_2, q_2 \rangle \langle s_7, q_2 \rangle \langle s_5, q_2 \rangle \langle s_3, q_2 \rangle \langle s_6, q_2 \rangle \langle s_4, q_1 \rangle \langle s_2, q_1 \rangle \\ & \langle s_7, q_1 \rangle \langle s_5, q_1 \rangle \langle s_3, q_0 \rangle \langle s_6, q_0 \rangle \langle s_4, q_0 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle \end{aligned}$$

– Again by backtracking, we visit the state $\langle s_3, q_2 \rangle$ the 2nd time and continue with its successor state $\langle s_9, q_2 \rangle$:

$$\begin{aligned} U = \quad & \langle s_1, q_2 \rangle \langle s_9, q_2 \rangle \langle s_3, q_2 \rangle \langle s_6, q_2 \rangle \langle s_4, q_1 \rangle \langle s_2, q_1 \rangle \langle s_7, q_1 \rangle \\ & \langle s_5, q_1 \rangle \langle s_3, q_0 \rangle \langle s_6, q_0 \rangle \langle s_4, q_0 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle \end{aligned}$$

– Now, the outermost DFS tracks back to the state $\langle s_4, q_1 \rangle$ which is also popped from the stack. But as $\langle s_4, q_1 \rangle \models \neg\Phi$ holds (since $q_1 \in F$), a nested DFS is started.

– **cycle_check(**$\langle s_4, q_1 \rangle$**)** fails to find a cycle.
But: The set of states T that cycle_check($\langle s_4, q_1 \rangle$) visited, remains valid!

– In the next step, $\langle s_2, q_1 \rangle$ is popped from U yielding

$$\langle s_7, q_1 \rangle \langle s_5, q_1 \rangle \langle s_3, q_0 \rangle \langle s_6, q_0 \rangle \langle s_4, q_0 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$$

Because $\langle s_2, q_1 \rangle \models \neg\Phi$, cycle_check($\langle s_2, q_1 \rangle$) is called.
It fails to find a cycle, since $Post(\langle s_2, q_1 \rangle) \setminus T = \varnothing$ (this is because $\langle s_4, q_1 \rangle$ as well as $\langle s_8, q_2 \rangle$ were already visited in the 1st nested DFS).

– The same applies to $\langle s_7, q_1 \rangle$ and $\langle s_5, q_1 \rangle$, since in the inner DFS, we already visited their successor states $\langle s_2, q_1 \rangle$ and $\langle s_7, q_1 \rangle$, respectively:

$$\langle s_3, q_0 \rangle \langle s_6, q_0 \rangle \langle s_4, q_0 \rangle \langle s_2, q_0 \rangle \langle s_0, q_0 \rangle$$

– From here, the outermost DFS continues without invoking cycle_check again.

Result:

$$\begin{aligned} TS_{Pet} \otimes \bar{\mathcal{A}} \models P_{pers} \quad & \iff \mathit{Traces}(TS_{Pet}) \cap \mathcal{L}_\omega(\bar{\mathcal{A}}) = \varnothing \\ & \iff \mathit{Traces}(TS_{Pet}) \cap \Big((2^{AP})^\omega \setminus P_{\mathit{live}} \Big) = \varnothing \\ & \iff \mathit{Traces}(TS_{Pet}) \subseteq P_{\mathit{live}} \\ & \iff TS_{Pet} \models P_{\mathit{live}} \end{aligned}$$

Exercises of Chapter 5

Answer to Exercise 5.2

$\varphi_1 = \Diamond \Box\, c$	no	$s_2 s_4 s_2 s_4 \ldots$
$\varphi_2 = \Box \Diamond\, c$	yes	
$\varphi_3 = \bigcirc \neg c \rightarrow \bigcirc \bigcirc\, c$	yes	
$\varphi_4 = \Box\, a$	no	$s_2 \ldots$
$\varphi_5 = a \,\mathsf{U}\, \Box\, (b \vee c)$	yes	
$\varphi_6 = (\bigcirc \bigcirc\, b) \,\mathsf{U}\, (b \vee c)$	no	$s_1 s_4 s_2 \ldots$

Answer to Exercise 5.4

(a) $\Box\, \neg\, (\mathit{Peter.use} \;\wedge\; \mathit{Betsy.use})$

(b) Finite time of usage means that in each state that a user uses the printer, for all possible following computations, at some point in time this user should have released the printer. This should hold for both *Peter* and *Betsy*, of course:

$$\Box\, ((\mathit{Peter.use} \;\Rightarrow\; \Diamond\, \mathit{Peter.release}) \;\wedge\; (\mathit{Betsy.use} \;\Rightarrow\; \Diamond\, \mathit{Betsy.release}))$$

(c) $\Box\, ((\mathit{Peter.request} \;\Rightarrow\; \Diamond\, \mathit{Peter.use}) \;\wedge\; (\mathit{Betsy.request} \;\Rightarrow\; \Diamond\, \mathit{Betsy.use}))$

(d) If a user requests access to the printer, it should be impossible to be in this *request* state forever:

$$\begin{aligned}&\Box\, ((\mathit{Peter.request} \;\Rightarrow\; \neg \Box\, \mathit{Peter.request})\\ &\quad\wedge\; (\mathit{Betsy.request} \;\Rightarrow\; \neg \Box\, \mathit{Betsy.request}))\end{aligned}$$

(e) Strict alternating access to the printer can be enforced by expressing that if a user is using the printer he/she will use it until he/she will not use it until the other user has used it.

$$\begin{aligned}&\Box\, ((\mathit{Peter.use} \;\Rightarrow\; \mathit{Peter.use} \,\mathsf{U}\, (\neg\, \mathit{Peter.use} \,\mathsf{U}\, \mathit{Betsy.use}))\\ &\quad\wedge\; \mathit{Betsy.use} \;\Rightarrow\; \mathit{Betsy.use} \,\mathsf{U}\, (\neg\, \mathit{Betsy.use} \,\mathsf{U}\, \mathit{Peter.use})))\end{aligned}$$

Answer to Exercise 5.5
We choose the following atomic propositions:

open_i	the door on floor i is unlocked
floor_i	the cabin is located on floor i (not moving)
req_i	the ith floor is requested

Then, we can formulate the given properties as LTL-formulas as follows for arbitrary $N \geqslant 0$:

(a) The doors are "safe", i.e., a floor door is never open if the cabin is not present at a given floor:

$$\varphi_1 = \Box \Big(\bigwedge_{i=0}^{N-1} (\mathit{open}_i \rightarrow \mathit{floor}_i) \Big)$$

(b) A requested floor will be served sometime:

$$\varphi_2 = \Box \left(\bigwedge_{i=0}^{N-1} (req_i \rightarrow \Diamond\, floor_i) \right)$$

(c) Again and again the lift returns to floor 0:

$$\varphi_3 = \Box\, (\neg floor_0 \rightarrow \Diamond\, floor_0) \wedge \Box \Diamond \neg floor_0$$

(d) When the top floor is requested, the lift serves it immediately and does not stop on the way there:

$$\varphi_4 = \Box \left(req_{N-1} \rightarrow \bigcirc \left(\left(\bigwedge_{i=0}^{N-2} \neg floor_i \right) \mathsf{U}\, floor_{N-1} \right) \right)$$

Answer to Exercise 5.6

(a) $\Box \varphi \rightarrow \Diamond \psi \equiv \varphi \,\mathsf{U}\, (\psi \vee \neg\varphi)$. We prove this by showing that:

$$\mathit{Words}(\Box \varphi \rightarrow \Diamond \psi) \;=\; \mathit{Words}(\varphi \,\mathsf{U}\, (\psi \vee \neg\varphi))$$

- $\subseteq$: Let $\sigma \in \mathit{Words}(\Box \varphi \rightarrow \Diamond \psi)$. Distinguish two cases:
  - $\sigma \models \Box \varphi \Rightarrow \sigma \models \Diamond \psi \Rightarrow \sigma \models \varphi \,\mathsf{U}\, \psi \Rightarrow \sigma \models \varphi \,\mathsf{U}\, (\psi \vee \neg\varphi)$
  - $\sigma \not\models \Box \varphi \Rightarrow \exists j.\ \sigma[j..] \not\models \varphi$. Let j be the smallest index such that $\sigma[j..] \not\models \varphi$. Then for all $i < j$ it holds $\sigma[i..] \models \varphi$. Thus, $\sigma \models \varphi \,\mathsf{U}\, \neg\varphi$, and it follows $\sigma \models \varphi \,\mathsf{U}\, (\psi \vee \neg\varphi)$.
- $\supseteq$: Let $\sigma \in \mathit{Words}(\varphi \,\mathsf{U}\, (\psi \vee \neg\varphi))$ such that $\sigma \models \Box \varphi$.
  To show: $\sigma \models \Diamond \psi$.
  Because of $\sigma \models \Box \varphi$, it holds: $\sigma \not\models \varphi \,\mathsf{U}\, (\neg\varphi)$. Since $\sigma \models \varphi \,\mathsf{U}\, (\psi \vee \neg\varphi)$, it holds: $\sigma \models \varphi \,\mathsf{U}\, \psi$. Therefore, also $\sigma \models \Diamond \psi$ holds.

(b) $\Diamond \Box \varphi \rightarrow \Box \Diamond \psi \equiv \Box\, (\varphi \,\mathsf{U}\, (\psi \vee \neg\varphi))$:

$$\begin{aligned}
\Diamond \Box \varphi \rightarrow \Box \Diamond \psi \quad &\equiv \neg \Diamond \Box \varphi \vee \Box \Diamond \psi \\
&\equiv \Box \Diamond \neg\varphi \vee \Box \Diamond \psi && \text{(* see remark *)} \\
&\equiv \Box \Diamond\, (\neg\varphi \vee \psi) && \text{(* distributive law *)} \\
&\equiv \Box\, (\Diamond \neg\varphi \vee \Diamond \psi) \\
&\equiv \Box\, (\neg \Box \varphi \vee \Diamond \psi) \\
&\equiv \Box\, (\Box \varphi \rightarrow \Diamond \psi) && \text{(* part (a) of this exercise *)} \\
&\equiv \Box\, (\varphi \,\mathsf{U}\, (\psi \vee \neg\varphi))\,.
\end{aligned}$$

Remark: We have two distributive laws regarding $\Diamond$ and $\Box$:

$$\begin{aligned}
\Diamond \Phi \vee \Diamond \Psi \quad &\equiv \Diamond\, (\Phi \vee \Psi) \\
\Box \Phi \wedge \Box \Psi \quad &\equiv \Box\, (\Phi \wedge \Psi)
\end{aligned}$$

But here, we have to show: $\Box \Diamond \neg\varphi \vee \Box \Diamond \psi \equiv \Box \Diamond\, (\neg\varphi \vee \psi)$. Therefore:

$$\begin{aligned}
\sigma \models \Box \Diamond \neg\varphi \vee \Box \Diamond \psi \quad &\iff \overset{\infty}{\exists}\, i.\ \sigma[i..] \models \neg\varphi \text{ or } \overset{\infty}{\exists}\, j.\ \sigma[j..] \models \psi \\
&\iff \overset{\infty}{\exists}\, k.\ \sigma[k..] \models (\neg\varphi \vee \psi) \\
&\iff \sigma \models \Box \Diamond\, (\neg\varphi \vee \psi).
\end{aligned}$$

(d) $\Diamond\,(\varphi \,\mathsf{U}\, \psi) \equiv \Diamond\,\psi$: We have to show $\mathit{Words}(\Diamond\,(\varphi \,\mathsf{U}\, \psi)) = \mathit{Words}(\Diamond\,\psi)$.

- $\mathit{Words}(\Diamond\,(\varphi \,\mathsf{U}\, \psi)) \subseteq \mathit{Words}(\Diamond\,\psi)$

$$\begin{aligned} \sigma \in \mathit{Words}(\Diamond\,(\varphi \,\mathsf{U}\, \psi)) &\implies \exists i \geqslant 0.\ \sigma[i..] \models \varphi \,\mathsf{U}\, \psi \\ &\implies \exists j \geqslant i.\ \sigma[j..] \models \psi \\ &\iff \sigma \models \Diamond\,\psi \\ &\iff \sigma \in \mathit{Words}(\Diamond\,\psi) \end{aligned}$$

- $\mathit{Words}(\Diamond\,(\varphi \,\mathsf{U}\, \psi)) \supseteq \mathit{Words}(\Diamond\,\psi)$

$$\begin{aligned} \sigma \in \mathit{Words}(\Diamond\,\psi) &\implies \exists j \geqslant 0.\ \sigma[j..] \models \psi \\ &\implies \sigma[j..] \models \varphi \,\mathsf{U}\, \psi \\ &\implies \sigma \models \Diamond\,(\varphi \,\mathsf{U}\, \psi) \end{aligned}$$

(g) $\Box\Diamond\varphi \to \Box\Diamond\psi \not\equiv \Box\,(\varphi \to \Diamond\,\psi)$. This can be shown as follows. Let $AP = \{a, b\}$, $\varphi = a$, $\psi = b$ and $\sigma = \varnothing\{a\}\varnothing^\omega$. The left hand side is fulfilled by σ, as its premise $\Box\Diamond\varphi$ is false. On the other hand, σ does not fulfill the right hand side, as b never holds along σ.

(i) $\bigcirc\Diamond\varphi \equiv \Diamond \bigcirc \varphi$: We have to show $\mathit{Words}(\bigcirc\Diamond\varphi) = \mathit{Words}(\Diamond \bigcirc \varphi)$.

$$\begin{aligned} \sigma \in \mathit{Words}(\bigcirc\Diamond\varphi) &\iff \exists i \geqslant 1.\ \sigma[i..] \models \varphi \\ &\iff \forall j \geqslant 0.\ \sigma[j{+}1..] \models \varphi \\ &\iff \sigma \models \Diamond \bigcirc \varphi \\ &\iff \sigma \in \mathit{Words}(\Diamond \bigcirc \varphi) \end{aligned}$$

(j) $(\Diamond\Box\varphi_1) \wedge (\Diamond\Box\varphi_2) \equiv \Diamond\,(\Box\varphi_1 \wedge \Box\varphi_2)$:
Note that due to the distribution law, $\Box\,(\varphi \wedge \psi) = \Box\varphi \wedge \Box\psi$, thus $\mathit{Words}(\Diamond\,(\Box\varphi_1 \wedge \Box\varphi_2)) = \mathit{Words}(\Diamond\Box\,(\varphi_1 \wedge \varphi_2))$. Thus, we have to show $\mathit{Words}((\Diamond\Box\varphi_1) \wedge (\Diamond\Box\varphi_2)) = \mathit{Words}(\Diamond\Box\,(\varphi_1 \wedge \varphi_2))$.

$$\begin{aligned} \sigma \in \mathit{Words}(\Diamond\Box\,(\varphi_1 \wedge \varphi_2)) &\iff \forall^\infty i.\ \sigma[i..] \models \varphi_1 \wedge \varphi_2 \\ &\iff \forall^\infty i.\ (\sigma[i..] \models \varphi_1 \wedge \sigma[i..] \models \varphi_2) \\ &\iff \forall^\infty i.\ (\sigma[i..] \models \varphi_1) \wedge \forall^\infty i.\ (\sigma[i..] \models \varphi_2) \\ &\iff \sigma \models \Diamond\Box\varphi_1 \wedge \sigma \models \Diamond\Box\varphi_2 \\ &\iff \sigma \models \Diamond\Box\varphi_1 \wedge \Diamond\Box\varphi_2 \\ &\iff \sigma \in \mathit{Words}(\Diamond\Box\varphi_1 \wedge \Diamond\Box\varphi_2) \end{aligned}$$

(k) $(\varphi_1 \,\mathsf{U}\, \varphi_2) \,\mathsf{U}\, \varphi_2 \equiv \varphi_1 \,\mathsf{U}\, \varphi_2$: We have to show $\mathit{Words}((\varphi_1 \,\mathsf{U}\, \varphi_2) \,\mathsf{U}\, \varphi_2) = \mathit{Words}(\varphi_1 \,\mathsf{U}\, \varphi_2)$.

$$\begin{aligned} & \sigma \in \mathit{Words}((\varphi_1 \,\mathsf{U}\, \varphi_2) \,\mathsf{U}\, \varphi_2) \\ \iff\ & \exists j \geqslant 0.\ \sigma[j..] \models \varphi_2 \text{ and } (\sigma[i..] \models \varphi_1 \,\mathsf{U}\, \varphi_2, 0 \leqslant i < j) \\ \iff\ & \exists j \geqslant 0.\ \sigma[j..] \models \varphi_2 \text{ and } (\exists k \geqslant i.\ \sigma[k..] \models \varphi_2 \text{ and } \sigma[l..] \models \varphi_1, 0 \leqslant l < k, 0 \leqslant i < j) \\ \iff\ & \exists j \geqslant 0.\ \sigma[j..] \models \varphi_2 \text{ and } (\exists k \geqslant 0.\ \sigma[k..] \models \varphi_2 \text{ and } \sigma[l..] \models \varphi_1, 0 \leqslant l < k) \\ \iff\ & \exists k \geqslant 0.\ \sigma[k..] \models \varphi_2 \text{ and } \sigma[l..] \models \varphi_1, 0 \leqslant l < k \\ \iff\ & \sigma \models \varphi_1 \,\mathsf{U}\, \varphi_2 \\ \iff\ & \sigma \in \mathit{Words}(\varphi_1 \,\mathsf{U}\, \varphi_2) \end{aligned}$$

Answer to Exercise 5.8
We prove two equivalences:

(a) The expansion law for the release operator R is given by:

$$\varphi \,\mathsf{R}\, \psi \equiv \psi \wedge (\varphi \vee \bigcirc\, (\varphi \,\mathsf{R}\, \psi))$$

According to the release semantics, either ψ holds at the current state and on the next state, $\varphi \,\mathsf{R}\, \psi$ holds again, or ψ is released because in the current state, both ψ and φ are satisfied. The correctness can be proven as follows:

$$\begin{aligned}
\varphi \,\mathsf{R}\, \psi &\equiv \neg(\neg\varphi \,\mathsf{U}\, \neg\psi) && \text{(* definition of } \mathsf{R} \text{*)}\\
&\equiv \neg\,(\neg\psi \vee (\neg\varphi \wedge \bigcirc\, (\neg\varphi \,\mathsf{U}\, \neg\psi))) && \text{(* expansion law of } \mathsf{U} \text{ *)}\\
&\equiv \psi \wedge \neg\,(\neg\varphi \wedge \bigcirc\, (\neg\varphi \,\mathsf{U}\, \neg\psi)) && \text{(* deMorgan *)}\\
&\equiv \psi \wedge (\varphi \vee \neg\bigcirc\, (\neg\varphi \,\mathsf{U}\, \neg\psi)) && \text{(* deMorgan *)}\\
&\equiv \psi \wedge (\varphi \vee \bigcirc \neg\,(\neg\varphi \,\mathsf{U}\, \neg\psi)) && \text{(* duality of } \bigcirc \text{ *)}\\
&\equiv \psi \wedge (\varphi \vee \bigcirc\, (\varphi \,\mathsf{R}\, \psi)) && \text{(* definition of } \mathsf{R} \text{ *)}
\end{aligned}$$

(b) Proof of the equivalence $\varphi \,\mathsf{R}\, \psi \equiv (\neg\varphi \wedge \psi)\ \mathsf{W}\ (\varphi \wedge \psi)$:

$$\begin{aligned}
\varphi \,\mathsf{R}\, \psi &\equiv \neg\,(\neg\varphi \,\mathsf{U}\, \neg\psi) && \text{(* definition of } \mathsf{R} \text{ *)}\\
&\equiv (\neg\varphi \wedge \neg\neg\psi)\ \mathsf{W}\ (\neg\neg\varphi \wedge \neg\neg\psi) && \text{(* duality of } \mathsf{U} \text{ and } \mathsf{W} \text{*)}\\
&\equiv (\neg\varphi \wedge \psi)\ \mathsf{W}\ (\varphi \wedge \psi)
\end{aligned}$$

(c) Proof of the equivalence $\varphi \,\mathsf{W}\, \psi \equiv (\neg\varphi \vee \psi)\ \mathsf{R}\ (\varphi \vee \psi)$:
The duality law for W and U states that:

$$\neg\,(\varphi \,\mathsf{W}\, \psi) \equiv (\varphi \wedge \neg\psi)\ \mathsf{U}\ (\neg\varphi \wedge \neg\psi)$$

$$\begin{aligned}
\varphi \,\mathsf{W}\, \psi &\equiv \neg\big[\underbrace{(\varphi \wedge \neg\psi)}_{\neg\Phi}\ \mathsf{U}\ \underbrace{(\neg\varphi \wedge \neg\psi)}_{\neg\Psi}\big]\\
&\equiv \underbrace{\neg\,(\varphi \wedge \neg\psi)}_{\Phi}\ \mathsf{R}\ \underbrace{\neg\,(\neg\varphi \wedge \neg\psi)}_{\Psi} && \text{(* } \Phi \,\mathsf{R}\, \Psi \equiv \neg(\neg\Phi \,\mathsf{U}\, \neg\Psi)\text{*)}\\
&\equiv (\neg\varphi \vee \psi)\ \mathsf{R}\ (\varphi \vee \psi)
\end{aligned}$$

(d) We have to prove, that $\varphi \,\mathsf{U}\, \psi \equiv \neg\,(\neg\varphi \,\mathsf{R}\, \neg\psi)$:

$$\varphi \,\mathsf{U}\, \psi \equiv \neg\,(\neg\varphi \,\mathsf{R}\, \neg\psi) \quad \Longleftrightarrow \quad \underbrace{\neg(\varphi \,\mathsf{U}\, \psi) \equiv (\neg\varphi \,\mathsf{R}\, \neg\psi)}_{\text{we prove this}}$$

$$\begin{aligned}
\neg(\varphi \,\mathsf{U}\, \psi) &\equiv \neg(\neg(\neg\varphi) \,\mathsf{U}\, \neg(\neg\psi))\\
&\equiv \neg\varphi \,\mathsf{R}\, \neg\psi && \text{(* definition of } \mathsf{R} \text{ *)}
\end{aligned}$$

Answer to Exercise 5.11

(a) The fair paths of *TS* are defined by:

$$\mathit{fair} = (\Box\Diamond\ (a \wedge b) \rightarrow \Box\Diamond\, \neg c) \wedge (\Diamond\Box\ (a \wedge b) \rightarrow \Box\Diamond\, \neg b)$$

The conclusion in the first conjunction $(\Box \Diamond (a \wedge b) \rightarrow \Box \Diamond \neg c)$ is fulfilled by every path, since no state in *TS* is labeled with *c*. Formally, this follows from $\Box \neg c \rightarrow \Box \Diamond \neg c$. Consider the second conjunct $(\Diamond \Box (a \wedge b) \rightarrow \Box \Diamond \neg b)$ of *fair*: Its premise is fulfilled only on the path $\pi = s_3^\omega$. But $\pi \not\models \Box \Diamond \neg b$. Therefore π is the only unfair path in *TS*. We conclude that:

$$\mathit{FairPaths}(TS) = \mathcal{L}_\omega \left((s_0 s_1)^\omega + (s_0 s_1)^+ s_2^\omega + s_3^+ s_4 s_5^\omega \right)$$

(b) $\varphi_2 = \bigcirc \neg a \rightarrow \Diamond \Box a$:
Consider the path $\pi_1 = s_3 s_4 (s_5)^\omega \in \mathit{FairPaths}(TS)$. For its corresponding trace

$$\mathit{trace}(\pi_1) = \sigma_1 = \{a, b\}\{b\}\varnothing^\omega$$

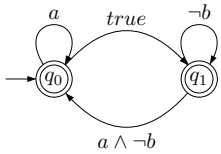
it holds $\sigma_1 \in \mathit{Words}(\bigcirc \neg a)$, but $\sigma_1 \notin \mathit{Words}(\Diamond \Box a)$.
$\Longrightarrow \sigma_1 \notin \mathit{Words}(\bigcirc \neg a \rightarrow \Diamond \Box a)$
$\Longrightarrow TS \not\models_{fair} \bigcirc \neg a \rightarrow \Diamond \Box a$.
$\varphi_4 = b \,\mathsf{U}\, \Box \neg b$:
Consider the path $\pi_2 = (s_0 s_1)^\omega \in \mathit{FairPaths}(TS)$. Here, we have

$$\mathit{trace}(\pi_2) = \sigma_2 = (\{a, b\}\{b\})^\omega$$

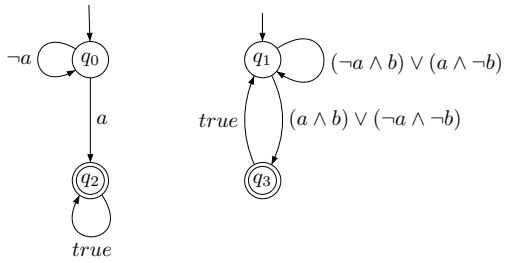
and $\sigma_2 \not\models b \,\mathsf{U}\, \Box \neg b$ since there exists no $i \geqslant 0$ s.t. $\sigma_2[i..] \models \Box \neg b$.
$\Longrightarrow TS \not\models_{fair} b \,\mathsf{U}\, \Box \neg b$
$\varphi_5 = b \,\mathsf{W}\, \Box \neg b$:
It holds $TS \models_{fair} \varphi_5$, since $(s_0 s_1)^\omega \models \Box b$, $(s_0 s)1)^+ s_2^\omega \models b \,\mathsf{U}\, \Box \neg b$, and $s_3^+ s_4 s_5^\omega \models b \,\mathsf{U}\, \Box \neg b$ and $s_3^\omega \models \Box b$.

Answer to Exercise 5.13

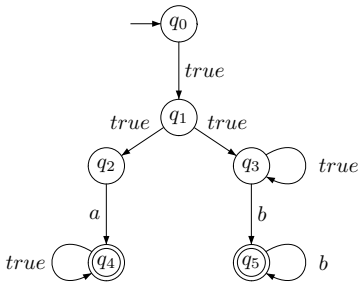
(a) $\varphi_1 = \Box \left(a \vee \neg \bigcirc b \right)$



(b) $\varphi_2 = \Diamond a \vee \Box \Diamond (a \leftrightarrow b)$



(c) $\varphi_3 = \bigcirc \bigcirc \big(a \vee \Diamond \Box b\big)$



Answer to Exercise 5.17

(a) Let $\psi = \Box\,(a \leftrightarrow \bigcirc \neg a)$ and $AP = \{a\}$.
First we transform ψ into the equivalent basic LTL-formula φ:

$$
\begin{aligned}
\psi &= \Box\,(a \leftrightarrow \bigcirc \neg a) \\
&= \neg\Diamond\,\neg(a \leftrightarrow \bigcirc \neg a) && \text{(* } \Box\varphi \equiv \neg\Diamond\,\neg\varphi \text{ *)} \\
&= \neg\Diamond\,\neg((a \wedge \bigcirc \neg a) \vee (\neg a \wedge \neg \bigcirc\ \neg a)) && \text{(* bijunktion *)} \\
&= \neg\Diamond\,\big(\neg(a \wedge \bigcirc \neg a) \wedge \neg(\neg a \wedge \neg \bigcirc\ \neg a)\big) && \text{(* deMorgan *)} \\
&= \neg\big[\,\mathsf{true}\ \mathsf{U}\ \big(\neg\underbrace{(a \wedge \bigcirc \neg a)}_{\varphi_1} \wedge \neg\underbrace{(\neg a \wedge \neg \bigcirc\ \neg a)}_{\varphi_2}\big)\big] = \varphi && \text{(* } \Diamond\varphi \equiv \mathsf{true}\ \mathsf{U}\,\varphi \text{ *)}
\end{aligned}
$$

(b) Now we compute $\mathit{closure}(\varphi)$:

$$
\begin{aligned}
\mathit{closure}(\varphi) = \{\quad & \mathsf{true}\,, \mathsf{false}\,, a, \neg a, \bigcirc \neg a, \neg \bigcirc\ \neg a, \\
& \varphi_1, \neg\varphi_1, \varphi_2, \neg\varphi_2, \\
& \neg\varphi_1 \wedge \neg\varphi_2, \neg(\neg\varphi_1 \wedge \neg\varphi_2), \\
& \mathsf{true}\ \mathsf{U}\,(\neg\varphi_1 \wedge \neg\varphi_2), \neg\,(\mathsf{true}\ \mathsf{U}\,(\neg\varphi_1 \wedge \neg\varphi_2))\ \}
\end{aligned}
$$

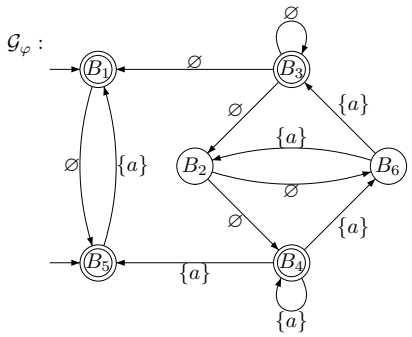
The elementary sets are:

	true	a	$\bigcirc \neg a$	$\overbrace{a \wedge \bigcirc \neg a}^{\varphi_1}$	$\overbrace{\neg a \wedge \neg \bigcirc\ \neg a}^{\varphi_2}$	$\neg\varphi_1 \wedge \neg\varphi_2$	true U $(\neg\varphi_1 \wedge \neg\varphi_2)$
B_1	1	0	0	0	1	0	0
B_2	1	0	0	0	1	0	1
B_3	1	0	1	0	0	1	1
B_4	1	1	0	0	0	1	1
B_5	1	1	1	1	0	0	0
B_6	1	1	1	1	0	0	1

(c) The GNBA $\mathcal{G}_\varphi = (Q, \Sigma, \delta, Q_0, \mathcal{F})$ is defined by:

$$
\begin{array}{ll}
Q & = \{B_1, B_2, B_3, B_4, B_5, B_6\} \\
\Sigma & = 2^{\{a\}} = \{\varnothing, \{a\}\} \\
Q_0 & = \{B_1, B_5\} \\
\mathcal{F} & = \left\{ F_{\mathsf{true}\,\mathsf{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)} \right\} \\
F_{\mathsf{true}\,\mathsf{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)} & = \{B_1, B_3, B_4, B_5\}
\end{array}
$$

The transition relation δ is given by the following graph representation (where also the unreachable parts are outlined):



Answer to Exercise 5.20

(a) Transform the negation $\neg\varphi = \neg\Box\,(a \rightarrow (\neg b\,\mathsf{U}\,(a \wedge b)))$ into an equivalent LTL-formula according to the basic LTL-syntax:

$$
\begin{array}{ll}
\neg\varphi & = \neg\Box\,(a \rightarrow (\neg b\,\mathsf{U}\,(a \wedge b))) \\
& \equiv \Diamond\,\neg(\neg a \vee (\neg b\,\mathsf{U}\,(a \wedge b))) \\
& \equiv \Diamond\,(a \wedge \neg(\neg b\,\mathsf{U}\,(a \wedge b))) \\
& \equiv \underbrace{true\,\mathsf{U}\,(a \wedge \neg(\neg b\,\mathsf{U}\,(a \wedge b)))}_{\psi}
\end{array}
$$

We then compute the closure:

$$
\begin{array}{ll}
closure(\psi) = \{ & true, false, a, \neg a, b, \neg b \\
& a \wedge b, \neg(a \wedge b), \neg b\,\mathsf{U}\,(a \wedge b), \neg(\neg b\,\mathsf{U}\,(a \wedge b)), \\
& a \wedge \neg(\neg b\,\mathsf{U}\,(a \wedge b)), \neg(a \wedge \neg(\neg b\,\mathsf{U}\,(a \wedge b))), \psi, \neg\psi\ \}
\end{array}
$$

(b) The elementary sets are as follows:

	a	b	$a \wedge b$	$\neg b\,\mathsf{U}\,(a \wedge b)$	$a \wedge \neg(\neg b\,\mathsf{U}\,(a \wedge b))$	ψ
B_1	0	0	0	0	0	0
B_2	0	0	0	0	0	1
B_3	0	0	0	1	0	0
B_4	0	0	0	1	0	1
B_5	0	1	0	0	0	0
B_6	0	1	0	0	0	1
B_7	1	0	0	0	1	1
B_8	1	0	0	1	0	0
B_9	1	0	0	1	0	1
B_{10}	1	1	1	1	0	0
B_{11}	1	1	1	1	0	1

(c) The elementary sets form the states of the GNBA $\mathcal{G}_{\neg\varphi} = \Big(Q, 2^{AP}, \delta, Q_0, \mathcal{F}\Big)$. We have

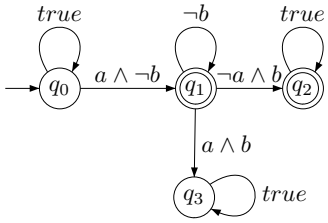
$$\begin{array}{ll} Q & = \{B_1, \ldots, B_{11}\} \\ Q_0 & = \{B_2, B_4, B_6, B_7, B_9, B_{11}\} \\ \mathcal{F} & = \Big\{F_{\neg b \,\mathsf{U}\, (a \wedge b)}, F_\psi\Big\}, \text{ where} \\ F_{\neg b \,\mathsf{U}\, (a \wedge b)} & = \{B_1, B_2, B_5, B_6, B_7, B_{10}, B_{11}\} \\ F_\psi & = \{B_1, B_3, B_5, B_7, B_8, B_{10}\} \end{array}$$

The transition relation is given as follows:

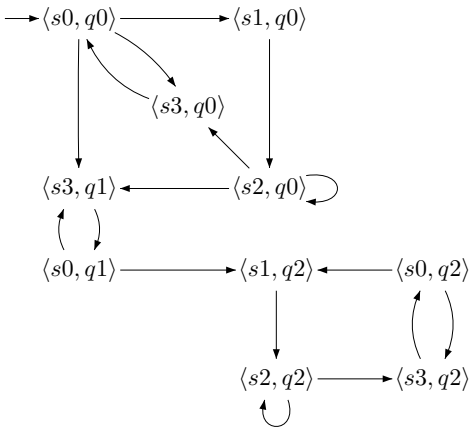
	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}
B_1	$\varnothing$				$\varnothing$						
B_2		$\varnothing$				$\varnothing$	$\varnothing$				
B_3			$\varnothing$					$\varnothing$		$\varnothing$	
B_4				$\varnothing$					$\varnothing$		$\varnothing$
B_5	$\{b\}$		$\{b\}$		$\{b\}$			$\{b\}$		$\{b\}$	
B_6		$\{b\}$		$\{b\}$		$\{b\}$	$\{b\}$		$\{b\}$		$\{b\}$
B_7	$\{a\}$	$\{a\}$			$\{a\}$	$\{a\}$	$\{a\}$				
B_8			$\{a\}$					$\{a\}$		$\{a\}$	
B_9				$\{a\}$					$\{a\}$		$\{a\}$
B_{10}	$\{a, b\}$		$\{a, b\}$		$\{a, b\}$			$\{a, b\}$		$\{a, b\}$	
B_{11}		$\{a, b\}$		$\{a, b\}$		$\{a, b\}$	$\{a, b\}$		$\{a, b\}$		$\{a, b\}$

(d) The following NBA $\mathcal{A}_{\neg\varphi}$ exactly recognizes $\mathit{Words}(\neg\varphi)$. To see this, consider the following equivalent LTL-formula:

$$\begin{aligned} \neg\varphi &\equiv \neg\Box\,(a \rightarrow (\neg b \,\mathsf{U}\, (a \wedge b))) \\ &\equiv \Diamond\, \neg(\neg a \vee (\neg b \,\mathsf{U}\, (a \wedge b))) \\ &\equiv \Diamond\, (a \wedge \neg(\neg b \,\mathsf{U}\, (a \wedge b))) \\ &\equiv \Diamond\, \big(a \wedge (b \,\mathsf{R}\, (\neg a \vee \neg b))\big) \end{aligned}$$



(e) The product transition system $TS \otimes \mathcal{A}_{\neg\varphi}$ is depicted below:



(f) We now check the validity of the persistence property "always $\bigwedge_{q\in F} \neg q$" where $F = \{q_1, q_2\}$ is the set of accepting states of $\mathcal{A}_{\neg\varphi}$. Therefore we use the nested DFS algorithm. Its outcome is:

$$\begin{aligned} U &= \langle s_0, q_0\rangle\langle s_3, q_1\rangle, \langle s_0, q_1\rangle\langle s_1, q_2\rangle\langle s_2, q_2\rangle\langle s_3, q_2\rangle\langle s_0, q_2\rangle \\ V &= \langle s_0, q_2\rangle\langle s_3, q_2\rangle\langle s_0, q_2\rangle \end{aligned}$$

Note that the top of the stack is indicated at the right. Therefore, in our traversal the NDFS algorithm yields the following counterexample:

$$\mathit{Reverse}(V.U) = \langle s_0, q_0\rangle\, \langle s_3, q_1\rangle\, \langle s_0, q_1\rangle\, \langle s_1, q_2\rangle\, \langle s_2, q_2\rangle\, \langle s_3, q_2\rangle\, \langle s_0, q_2\rangle\, \langle s_3, q_2\rangle\, \langle s_0, q_2\rangle$$

The corresponding trace $\sigma \in \mathit{Traces}(TS) = \varnothing\{a\}\varnothing\{b\}\{a, b\}(\{a\}\varnothing)^\omega$ indeed violates the formula $\varphi = \Box\,(a \rightarrow (\neg b\,\mathsf{U}\,(a \wedge b)))$. (Note that depending on the choice of the successor states during the NDFS, different counterexamples are obtained!)

Answer to Exercise 5.25

(a) The modelling of Szymanski's protocol in PROMELA is rather straightforward. A point of attention is the atomicity of the tests in the protocol on lines l3, l5, l7, l9 and l11. This is established by using the `atomic` construct. Recall that by using this construct we should be careful with non-initial statements within the atomic construct that can be blocking. Typical examples are output and input statements. Inspection of Szymanski's protocol reveals that such anomalous behaviour cannot occur, and thus does not need further attention.

```
/*
   Szymanski's mutual exclusion protocol
   with atomic tests on the global variable flag
*/

#define N 2     /* the number of processes */
```

```
byte flag[N];   /* this arrays ranges in 0..4  */

proctype P(byte i)    /* i is the process number */
{ byte j;

l1:   do                          /* non-critical section       */
      :: true ->                  /* infinite loop; true -> may */
l2:      flag[i] = 1;             /* be omitted                 */
l3:      atomic{
            j = 0;
            do
            :: j < N -> if
                        :: flag[j] >= 3 -> goto l3;
                        :: else -> skip;
                        fi; j++;
            :: j == N -> break;
            od;
            }        /* end of atomic test */
l4:      flag[i] = 3;
l5:      atomic{
            j = 0;
            do
            :: j < N -> if
                        :: flag[j] == 1 -> goto l6;
                        :: else -> skip;
                        fi; j++;
            :: j == N -> goto l8;
            od;
            }
l6:      flag[i] = 2;
l7:      atomic{
            j = 0;
            do
            :: j < N -> if
                        :: flag[j] == 4 -> goto l8;
                        :: else -> skip;
                        fi; j++;
            :: j == N -> goto l7;
            od;
            }
l8:      flag[i] = 4;
l9:      atomic{
            j = 0;
            do
            :: j < i -> if
                        :: flag[j] >= 2 -> goto l9;
                        :: else -> skip;
```

```
                        fi; j++;
            :: j == i -> break;
            od;
            }
l10:     /* critical section  */
l11:     atomic{
            j = i+1;
            do
            :: j < N ->  if
                         :: !(flag[j] < 2 || flag[j] > 3) -> goto l11;
                         :: else -> skip;
                         fi; j++;
            :: j == N -> break;
            od;
            }
l12:     flag[i] = 0;
     od;
}

init
{ byte k;
  atomic{
     do
     :: k<N -> flag[k] = 0; run P(k); k++;
     :: k==N -> break;
     od
  }
}
```

(b) In order to check the mutual exclusion property, the idea is to extend the specification with an auxiliary global variable `numbercritical`, say, that keeps track of the number of processes that are in the critical section. Initially, no process is in the critical section, so we initialize

```
byte numbercritical = 0;
```

We increment the variable once a process enters the critical section. Accordingly we change line `l10` into

```
l10:    numbercritical++;      /* critical section  */
```

In a similar way, we should administer each process that leaves the critical section. Thus, we modify line `l10` into

```
l10:    numbercritical++;      /* critical section  */
        numbercritical--;
```

Finally, we add an auxiliary process `Mutex` to the specification that checks for each reachable state whether `numbercritical` never exceeds one.

```
proctype Mutex()
{
  assert(numbercritical <= 1)
}

init
{ byte k;
  atomic{
     do
     :: k<N -> flag[k] = 0; run P(k); k++;
     :: k==N -> break;
     od;
     run Mutex();                 /* also start checking */
   }
}
```

Observe that we start the process `Mutex` within the `atomic`-clause of the `init` process to guarantee that checking the mutex property starts simultaneously with the execution of the rest of the specification.

Verifying this system with SPIN for various values of N (the number of processes) yields that Szymanski's protocol indeed satisfies the mutual exclusion property. (The interested reader might want to verify that by omitting the `atomic` construct at decreasing `numbercritical`, as discussed above, indeed leads to a violation of the mutual exclusion property.) The details of the verification results:

N	size of state vector (in bytes)	number of states	run time (in sec)	state space (in bytes)
2	36	307	0.0	11052
3	44	2387	0.1	105028
4	56	18365	1.1	477490
5	64	143138	10.8	9160812
6	72	714123	72.5	51416856

The second column indicates the number of bytes needed to store all information concerning a *single* state. This includes, amongst others, the values of all local variables that are relevant to that state, and the values of all global variables (including the content of channels). The memory usage (last column) is obtained by multiplying the state vector size with the number of states (third column). Note the significant increase of this memory usage on increase of N.

(c) To solve this exercise it is essential that we are able to check whether a process is currently at a particular label. The way in PROMELA to check whether a process `P` is currently at a statement labeled `label`, say, is to use the predicate

```
P[pid]@label
```

where `pid` is the process id that SPIN assigns to the process instantiation of `P`. Note that we these ids are used internally by SPIN. These process ids should not be confused with the process numbers that processes `P` have as parameters in our PROMELA specification! To be

able to make statements about certain process instantiations, this requires insight in how SPIN handles these process ids. The `init` process always obtains process id 0, and all other numbers are assigned in the order of instantiation. In case of doubt, the process ids can be obtained from the information generated by a simulation.

1. In order to check whether a process is in the inner sanctum (i.e., at one of the labels l8 through l12) or at the doorway (i.e., at label l8) we define for process id `n` two macros as follows:

   ```
   #define inner_sanctum(n) \
     (P[n]@l8 || P[n]@l9 || P[n]@l10 || P[n]@l11 || P[n]@l12)

   #define doorway(n)   P[n]@l4
   ```

   Expressing the constraint that whenever some process is in the inner sanctum, then there is no process in the doorway would be straightforward using implication. For two processes this amounts to:

   ```
   assert( (inner_sanctum(1) "implies" !doorway(2)) &
           (inner_sanctum(2) "implies" !doorway(1))
         )
   ```

   where we use the fact that for our specification the `n`-th instantiation of process `P` obtains process id `n`. Unfortunately, PROMELA does not support implication. By using the fact that for any two propositions Q and R we have that

   $$Q \Rightarrow R \text{ is equivalent to } \neg Q \vee R$$

   we add the following macro-definition to our specification:

   ```
   #define impl(p,q)   (!p || q)
   ```

   We thus obtain the following `Check` process:

   ```
   proctype Check()
   {
     assert( impl(inner_sanctum(1), !doorway(2)) &
             impl(inner_sanctum(2), !doorway(1))
           )
   }
   ```

   (Note that this piece of code must be adapted if we increase the number of processes N.) The process `Check` is instantiated in the `init`-part as final process; in this way we do not change the process ids that are assigned by SPIN to the `P`-processes. Verification with SPIN reveals that this property is indeed satisfied.

2. ```
   #define waitin(n)  (waiting_room(n) || inner_sanctum(n))

   #define last_sanctum(n)  (P[n]@l10 || P[n]@l11 || P[n]@l12)

   proctype Check()
   {
   ```
   ```

```
  assert( impl(last_sanctum(1), (!waitin(2) || 1 < 2)) &
          impl(last_sanctum(2), (!waitin(1) || 2 < 1))
        )
}
```

3. To verify this property we change the `Check` process as follows:

```
#define end_of_cs(n)    P[n]@l12

#define waiting_room(n)  (P[n]@l5 || P[n]@l6 || P[n]@l7)

#define waitin_flag4(n)  (!(waitin(n)) || flag[n] == 4)

proctype Check()
{
  assert( impl(end_of_cs(1), waitin_flag4(2)) &
          impl(end_of_cs(2), waitin_flag4(1))
        )
}
```

Answer to Exercise 5.26

1. The following PROMELA specification is obtained by a straightforward translation of the original Peterson's algorithm:

```
#define N 5 /* nr of processes (use 5 for demos) */
#define I 3 /* node given the smallest number    */
#define L 10 /* size of buffer  (>= 2*N) */

chan q[N] = [L] of { byte };

proctype process (chan in, out; byte ident)
{ byte e, f, d;

  printf("MSC: %d\n", ident);

activ:
  d = ident;
  do
  :: true -> out!d;
     in?e;
             if
     :: (e == ident) -> d = e;
                                goto announce
             :: else -> skip
             fi;
             if
```

```
              :: (e < d) -> out!d
      :: else -> out!e
              fi;
              in?f;
              if
              :: (f == ident) -> d = f;
                                 goto announce
              :: else -> skip
              fi;
              if
        :: (e >= d) && (e >= f) -> d = e
      :: else -> goto relay
      fi
  od;

relay:
end:
  do
  :: in?d -> if
              :: (d == ident) -> d = f;
                                 goto announce
              :: else -> skip
              fi;
              out!d
  od;

announce:
  printf("MSC: LEADER elected\n", d);   /* process d is leader */

stop:
  skip
}

init {
  byte i;
  atomic {
i = 1;
do
:: i <= N -> run process (q[i-1], q[i%N], (N+I-i)%N+1);
                      i = i+1
:: i > N -> break
od
  }
}
```

Unfortunately, we obtain an invalid end-state when verifying this specification using SPIN. The problem is that while being active, a process may wait for the arrival of a message (statement `in?e`) that will never come. This occurs in the situation when all other processes

have recognised the election of a leader and thus cannot originate a message. This scenario is obtained when simulating the above specification with SPIN guided by the counterexample that it provides when checking for invalid end-states.

In order to avoid the invalid end-states we modify the Promela specification slightly and let each process send an 'elect' message (`el` for short) on recognition of a leader process. The messages that are used for distributing process identities are named 'identify', `id` for short. Both types of messages carry a process identity as parameter. So, we define:

```
mtype = {id, el};    /* two symbolic message names */

chan q[N] = [L] of { mtype, byte };
```

in the pre-amble of our modified PROMELA specification. An `el`-message is sent as soon as a leader is detected. This appears in the `announce` state. To avoid the invalid end-state we also allow a process to receive an `el`-message when it waits for an `id`-message. The code of a process thus becomes:

```
proctype process (chan in, out; byte ident)
{ byte e, f, d;

  printf("MSC: %d\n", ident);

activ:
  d = ident;
  do
  :: true -> out!id(d);
     if :: in?id(e) -> skip
                :: in?el(f) -> out!el(f);
                               goto stop
             fi;
             if
     :: (e == ident) -> d = e;
                                goto announce
             :: else -> skip
             fi;
             if
             :: (e < d) -> out!id(d)
     :: else -> out!id(e)
             fi;
             in?id(f);
             if
             :: (f == ident) -> d = e;
                                goto announce
             :: else -> skip
             fi;
             if
       :: (e >= d) && (e >= f) -> d = e
     :: else -> goto relay      /* become inactive */
```

```
      fi
  od;

relay:
end:
  do
  :: in?id(d) -> if
                 :: (d == ident) -> d = e;
                                    goto announce
                 :: else -> skip
                 fi;
                 out!id(d)
  od;

announce:
  printf("MSC: %d is LEADER\n", d);
  nr_leaders = nr_leaders + 1;
  out!el(d);
  goto stop;  /* process d is leader */

stop:
  skip
}
```

Checking with SPIN reveals that this specification contains no invalid end-states.

2.
   - This property is formalised by `[] (nr_leaders <= 1)` where `nr_leaders` is an auxiliary variable that is increased on encountering a new leader (that is, in state `announce`). The property is satisfied.
   - The formalisation of this property is `<> (nr_leaders == 1)`, and needs no further extension of the PROMELA specification. The property is satisfied.
   - In order to prove this property we extend the Promela specification with an auxiliary variable `leader_id` that keeps track of the identity of the elected leader. It is assigned the value `d` in the `announce` state. The property `<> (leader_id == N)` is satisfied.
   - To this end, we add the auxiliary variable `mc` that counts the amount of messages that are sent. It is initialised to 0 and is increased each time an `id`-message is transmitted. Notice that we do not count `el`-messages, since these were added for our purposes and do not contribute to the leader election process. In order to check the property `[] (mc < = 18)`, where 18 corresponds to $2N\lfloor \log_2 N \rfloor + N$ for $N = 5$, we enable the super-trace algorithm (otherwise we get space problems) and obtain no errors

Answer to Exercise 5.27

1. As indicated in the exercise the main problem with encoding the algorithm of Berman and Garay in PROMELA is to keep the state space manageable. This is done by keeping the number of variables small. First, we define some macros that are convenient:

```
#define N 2   /* number of reliable processes */
#define K 1   /* number of unreliable processes */
#define M 3   /* total number of processes (i.e., N+K) */
              /* M must be odd for majority to be well-defined */
```

As unreliable processes send random bit-values around, and as the initial bit-values of all processes is left unspecified, it is convenient to have a procedure that assigns a bit to a variable in a completely non-deterministic, i.e., random way. This is done by:

```
#define RANDOM(x) \
   if \
   :: x = 1 \
   :: x = 0 \
   fi \
```

Here, one should make sure that there are no spaces in the text after the backslash that ends a line. For the communication structure between all the processes several choices do exist. For the sake of brevity, we do not discuss all possible implementations (such as buses and so on), but simply take the suggestion of the exercise and make a matrix of $M \times M$ channels, all of capacity one as follows:

```
typedef Achan {
  chan ch[M] = [1] of {bit};
}

Achan Mchan[M];   /* a matrix of M^2 communication channels */
```

Unreliable and reliable processes are assumed to follow exactly the same protocol, except that the values that are transmitted by the unreliable processes are non-deterministically determined. We, therefore, use a single `proctype` definition to define both reliable as well as unreliable processes. Each process has a parameter that is a process number, from 0 to $M-1$. In the following we adopt the convention that the first K processes, i.e., process 0 up to process $K-1$, are unreliable. The other processes, numbered from K through $M-1$ are thus reliable. (We realise that this is an abstraction of the reality where it is not known in advance which processes are reliable and which are not.) The `init` section now simply looks as follows:

```
init
{ byte k;   /* variable to iterate over the processes */

  atomic{
       do
       :: k < M -> run P(k); k++
       :: k == M ->  break
       od
  }
}
```

The PROMELA specification of a process is now as follows.

```
proctype P(byte me)   /* me is the process number */
{ bit  c;             /* current bit-value of a process */
  byte i,             /* the current round */
       j,             /* local variable for iterating over all processes */
       W;             /* the number of received one-values */

  atomic{ i = 0; RANDOM(c); }       /* determine initial value randomly */
  do
  :: i < K+1 -> /* broadcast own bit-value to all processes atomically */
      atomic{
        j = 0;
        do
        :: j < M  -> if
                     :: me < K  -> RANDOM(c) /* select randomly if unreliable */
                     :: me >= K -> skip
                     fi;
                     Mchan[me].ch[j]!c; j++
        :: j == M -> break
        od;
      }
      /* receive value from all processes */
      atomic{ j = 0; W = 0; }
      do
      :: j < M -> atomic{ Mchan[j].ch[me]?c; W = W+c; j++ }
      :: j == M -> break
      od;
      /* W equals the number of received one-values  */
      /* the number of received zero-values is (M-W) */
      /* the received majority is thus W > (M-W)     */
      if
      :: i == me ->  /* broadcast majority to all processes */
         atomic{
           j = 0;
           do
           :: j < M  -> c = (W > (M-W));         /* majority */
                        if
                        :: me < K  -> RANDOM(c)  /* select random value */
                        :: me >= K -> skip
                        fi;
                        Mchan[me].ch[j]!c; j++
           :: j == M -> break
           od;
         }
      :: else -> skip
      fi;
      atomic{
        Mchan[i].ch[me]?c;       /* receive the broadcasted majority */
```

```
        if
        :: W >= N     -> c = 1   /* if N or more identical values are */
        :: (M-W) >= N -> c = 0   /* received then set c to this value */
        :: else       -> skip    /* else, set c to last value received from i */
        fi;
        i++;                     /* goto next round */
      }
  :: else -> break
  od
}
```

The comments in the PROMELA specification should make clear which parts of the specification correspond to which parts of the informal description of the protocol. A few remarks are in order, though. First, we note that we frequently use atomic statements in order to keep the state space manageable. For the same reason, in each process only counts the number of one-values that have been received. Since the total number of processes is known (M), from this information the number of zero-values can be determined, and the majority value can be determined. Finally, the reader can easily verify that the only difference between an unreliable process (with `me < K`) and a reliable process (`me >= K`) is that an unreliable process sends arbitrary bit-values around.

Some possibilities to further diminish the size of the state space:

- include synchronisation points such that all processes start (some phase of) a round at the same time, or
- rather than broadcasting the majority value by process `me` in round `me`, just use a global variable that each process can read, or
- put some ordering on top of the transmission of values by processes (and re-use channels; then a single channel would be sufficient)
- for transmitting the value to a process for each one a new random value is chosen; this could be changed in selecting a random value that is sent to all of them
- avoid the explicit transmission of a process to itself; this would save M channels.

Note: depending on the version of XSPIN at hand, it may be necessary to set the verification options `-o1 -o2 -o3` when simulating the PROMELA specification. This can be set in the verification menu under advanced verification options.[1]

2. Two claims have to be specified in LTL:

   (a) "eventually every reliable process has the same value in its local variable". As local process variables (such as the variable `c`) cannot be used in SPIN in formulas, we first have to make the final bit-value of each reliable process visible. This is done by adding a bit-array of size N to the specification:

   ```
   bit V[N]; /* the final values of the reliable processes */
   ```

   At the end of the body of a (reliable) process, this value is set to `c`. As a first attempt of a formula we try `<> p` where `p` is defined by

[1] This is due to a new feature of SPIN that is present in its latest release, and was not present in earlier versions. We apologise for this unknown (also to us) feature.

```
#define p   ((V[0] == V[1]) && ..... && (V[N-1] == V[N]))
```

This specifies that all final values are equal. This formula, turns out to be trivially satisfied as, for instance, initially all values in the array `V` equal 0. Of course, this is not what we intended. We therefore need to make sure that when we check the validity of `p` as defined just above, all processes have indeed terminated.] We do so, for instance, by introducing another variable `T` that is a counter keeping track of the number of reliable processes that have finished execution of the protocol. Thus:

```
byte T;   /* the number of terminated reliable processes */
```

The desired property now becomes `<> (p && q)` with:

```
#define  q    (T == N)
```

After the loop in the main body of a process we now add the following PROMELA fragment:

```
atomic{
  if
  :: me >= K -> T++; V[me-K] = c  /* reliable */
  :: me < K  -> skip              /* unreliable */
  fi
}
```

As the first K processes are unreliable, we take the index `me-K` for storing the final bit-values of the reliable process `me`.

(b) "if all reliable processes have the same initial value then their final value is the same as their common initial value". It is clear that we can re-use the array `V` to refer to the final bit-values of a process. In addition, we need a mechanism to record the initial bit-values. Therefore we introduce:

```
bit I[N]; /* the initial values of the reliable processes */
```

and we change the initialisation

```
atomic{ i = 0; RANDOM(c); }
```

in the beginning of the body of a process into:

```
atomic{
  i = 0; RANDOM(c);
  if
  :: me >= K -> I[me-K] = c  /* reliable */
  :: me < K  -> skip         /* unreliable */
  fi
}
```

The property that we are looking for would naively be encoded as `<> (r -> s)` with

```
#define r   ((I[0] == I[1]) && ..... && (I[N-1] == I[N]))
#define s   ((I[0] == V[0]) && ..... && (I[N] == V[N]))
```

This has, however, a similar problem as the previous property: the formula `<> (r -> s)` is trivially satisfied in the initial state where all values are (by default) equal to zero. We therefore arrive at the following property

```
<> ((q && r) -> s)
```

where `q` is, as before, referring to the state in which all reliable processes have finished the execution of the protocol.

3. Verification with SPIN, using the supertrace algorithm indicates that both properties are satisfied by the protocol.

4. This is simply checked by verifying the first LTL-formula on a model with N equals to 2 and K equal to 1. The protocol thus consists of just two rounds. The shortest counter-example that SPIN generates provides the following scenario: the initial values of the two reliable processes (numbered 1 and 2) are 0 and 1. The unreliable process sends value 1 to process 1, and value 0 to process 2 in the first round, after which it sends the majority value 1 to both reliable processes. At the end of the first round, the local value of process 1 equals 1, whereas process 2 has value 0. In the second round, the unreliable process sends different values to process 1 (value 1) and process 2 (value 0). Despite the broadcasting of the majority value by process 1 at the end of this round, process 1 has value 1 whereas process 2 terminates with value 0.

Exercises of Chapter 6

Answer to Exercise 6.2
For any CTL state formula Φ_i, we have to compute $Sat(\Phi_i) = \{s \in S \mid s \models \Phi_i\}$. From this, we can decide $TS \models \Phi_i$ by checking $I \subseteq Sat(\Phi_i)$.

(a) $\Phi_1 = \forall(a \,\mathsf{U}\, b) \vee \exists \bigcirc (\forall\Box\, b)$:
Exemplary argumentation for s_0:

$$\begin{aligned} s_0 \models \Phi_1 &\iff s_0 \models \forall(a \,\mathsf{U}\, b) \vee \exists \bigcirc (\forall\Box\, b) \\ &\iff s_0 \models \forall(a \,\mathsf{U}\, b) \text{ or } s_0 \models \exists \bigcirc (\forall\Box\, b) \end{aligned}$$

We have $s_0 \not\models \forall(a \,\mathsf{U}\, b)$ because $s_0 \not\models a$ and $s_0 \not\models b$. But $s_0 \models \exists \bigcirc (\forall\Box\, b)$, since for $\pi = s_0 s_4^\omega$ we have $\pi \models \bigcirc(\forall\Box\, b)$. Thus, $s_0 \in Sat(\Phi_1)$.

Using a similar reasoning for the other states, we obtain:

$$Sat(\Phi_1) = \{s_0, s_1, s_2, s_3, s_4\} \text{ and } I \subseteq Sat(\Phi_1) \Longrightarrow TS \models \Phi_1$$

(b) $\Phi_2 = \forall\Box\, \forall(a \,\mathsf{U}\, b)$:

$$\begin{aligned} s \models \Phi_2 &\iff \forall \pi \in Paths(s).\ \pi \models \Box\, \forall(a \,\mathsf{U}\, b) \\ &\iff \forall \pi \in Paths(s).\ \forall i \geqslant 0.\ \pi[i] \models \forall(a \,\mathsf{U}\, b) \\ &\iff \forall \pi \in Paths(s).\ \forall i \geqslant 0.\ \forall \pi' \in Paths(\pi[i]).\ \pi' \models a \,\mathsf{U}\, b \end{aligned}$$

We consider the state s_0 and the path $\pi'' = s_0 s_4^\omega$. We have $\pi'' \not\models a \,\mathsf{U}\, b$. Choose $i = 0$ and $\pi' = \pi''$. Then the equivalences yield $s_0 \not\models \Phi_2$.

With the same arguments, we infer $s_1 \not\models \Phi_2, s_2 \not\models \Phi_2$ and $s_3 \not\models \Phi_2$.

For s_4, we have $s_4 \models \Phi_2$ since the only possible path is $\pi' = s_4^\omega$ with $\pi' \models a \,\mathsf{U}\, b$

The result is:

$$Sat(\Phi_2) = \{s_4\} \Longrightarrow TS \not\models \Phi_2$$

(c) $\Phi_3 = (a \wedge b) \rightarrow \exists\Box\, \exists \bigcirc\, \forall(b \,\mathsf{W}\, a)$:
The states s_0, s_1, s_3 and s_4 do not fulfill the state formula $(a \wedge b)$ in the premise. Therefore we have $s_0, s_1, s_3, s_4 \in Sat(\Phi_3)$.

We only have to consider the validity of the conclusion wrt. s_2:

$$\begin{aligned} s_2 \models \exists\Box\, \exists \bigcirc\, \forall(b \,\mathsf{W}\, a) &\iff \exists \pi \in Paths(s_2).\ \pi \models \Box\, \exists \bigcirc\, \forall(b \,\mathsf{W}\, a) \\ &\iff \exists \pi \in Paths(s_2).\ \forall i \geqslant 0.\ \pi[i] \models \exists \bigcirc\, \forall(b \,\mathsf{W}\, a) \\ &\Longrightarrow \pi[0] = s_2 \models \exists \bigcirc\, \forall(b \,\mathsf{W}\, a) \\ &\iff \exists \pi' \in Paths(s_2).\ \pi' \models \bigcirc \forall(b \,\mathsf{W}\, a) \\ &\Longrightarrow \pi'[1] = s_3 \models \forall(b \,\mathsf{W}\, a) \\ &\iff \forall \pi'' \in Paths(s_3).\ \pi'' \models b \,\mathsf{W}\, a \end{aligned}$$

Consider $\pi'' = s_3 s_0 (s_4)^\omega$. Then $\pi'' \not\models b \,\mathsf{W}\, a$ (because of s_0).
$\Longrightarrow s_2 \notin Sat(\Phi_3)$

This yields

$$Sat(\Phi_3) = \{s_0, s_1, s_3, s_4\} \text{ and } TS \models \Phi_3$$

(d) $\Phi_4 = (\forall\Box\,\exists\Diamond\,\Phi_3)$:
Because we can reach a $Sat(\Phi_3)$-state from every state in TS, we can directly infer

$$Sat(\Phi_4) = \{s_0, s_1, s_2, s_3, s_4\} \text{ and } TS \models \Phi_4$$

Answer to Exercise 6.3

(a) Let

$$\begin{aligned} \Phi_1 &= \forall\Diamond\, a \vee \forall\Diamond\, b \\ \Phi_2 &= \forall\Diamond\,(a \vee b) \end{aligned}$$

– We first consider the first implication, $\Phi_1 \Longrightarrow \Phi_2$. We prove for any transition system TS that $s \models \Phi_1 \Longrightarrow s \models \Phi_2$:

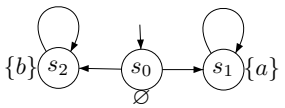
$$s \models \Phi_1 \iff s \models \forall\Diamond\, a \text{ or } s \models \forall\Diamond\, b$$

case 1:

$$\begin{aligned} s \models \forall\Diamond\, a &\iff \forall\pi \in Paths(s).\ \pi \models \Diamond\, a \\ &\iff \forall\pi \in Paths(s).\ \exists j \geqslant 0.\ \pi[j] \models a \\ &\Longrightarrow \forall\pi \in Paths(s).\ \exists j \geqslant 0.\ (\pi[j] \models a \text{ or } \pi[j] \models b) \\ &\iff \forall\pi \in Paths(s).\ \exists j \geqslant 0.\ (\pi[j] \models (a \vee b)) \\ &\iff \forall\pi \in Paths(s).\ \pi \models \Diamond\,(a \vee b) \\ &\iff s \models \forall\Diamond\,(a \vee b) \end{aligned}$$

case 2: analogous, by replacing a by b.

– The converse, i.e., $\Phi_2 \Longrightarrow \Phi_1$, does not hold. Consider the following transition system:

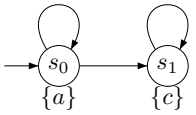


Then, $TS \models \Phi_2$ is fulfilled, but $TS \not\models \Phi_1$.

(b) Consider the CTL state formulae

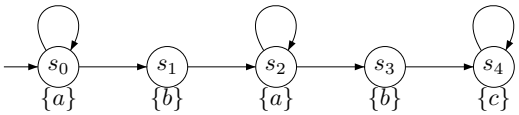
$$\Psi_1 = \exists(a\,\mathsf{U}\,\exists(b\,\mathsf{U}\,c)) \quad \text{and} \quad \Psi_2 = \exists(\exists(a\,\mathsf{U}\,b)\,\mathsf{U}\,c)$$

– We have $\Psi_1 \not\Rightarrow \Psi_2$. The following transition system provides a counterexample:



Obviously, $TS \models \Psi_1$. Because the proposition b does not hold in any state of TS, c has to hold in an initial state. As this is not the case, $TS \not\models \Psi_2$.

– $\Psi_2 \not\Rightarrow \Psi_1$. Again by counterexample:



We have $TS \models \Psi_2$ as the initial path $s_0 s_1 s_2 s_3 s_4 s_5^\omega$ satisfies $\exists(a \,\mathsf{U}\, b) \,\mathsf{U}\, c$. But $TS \not\models \Psi_1$ as the only state satisfying $\exists(b \,\mathsf{U}\, c)$, i.e., state s_3 cannot be reached via an a-path from the initial state.

Answer to Exercise 6.8
Consider the following two transition systems TS_1 and TS_2:

Note that TS_2 has two initial states: s_1 and s_2. We have that:

$$\begin{aligned} \mathit{Traces}(TS_1) &= \mathcal{L}\big(\{a\}^+.(\{b\}^\omega + \{c\}^\omega) + \{a\}^\omega\big) \text{ and} \\ \mathit{Traces}(TS_2) &= \mathcal{L}\big((\{a\}^+.\{b\}^\omega + \{a\}^+.\{c\}^\omega + \{a\}^\omega)\big) = \mathcal{L}\big(\{a\}^+.(\{b\}^* + \{c\}^*) + \{a\}^\omega\big). \end{aligned}$$

Although $\mathit{Traces}(TS_1) = \mathit{Traces}(TS_2)$, there is a CTL formula that distinguishes TS_1 and TS_2. Consider, e.g., the CTL formula:

$$\Phi \quad = \forall\Box\Big(a \rightarrow \big(\exists\Diamond\, b \wedge \exists\Diamond\, c\big)\Big)$$

It holds $TS_1 \models \Phi$ whereas $TS_2 \not\models \Phi$.

Answer to Exercise 6.9
First consider the fairness assumption *fair* in isolation. It consists of a strong and a weak fairness constraint, namely

- strong fairness constraint: $\Box\Diamond\forall\bigcirc(a \wedge \neg b) \rightarrow \Box\Diamond\forall\bigcirc(b \wedge \neg a)$.
  Consider first the premisse of this constraint. We have $\mathit{Sat}(\forall\bigcirc(a \wedge \neg b)) = \{s_0, s_4\}$. As each infinite path in TS passes through s_0 or s_4 infinitely often, each infinite path has to fulfill $\Box\Diamond\forall\bigcirc(b \wedge \neg a)$. As $\mathit{Sat}(\forall\bigcirc(b \wedge \neg a)) = \{s_1\}$, every fair path must visit state s_1 infinitely often.
- weak fairness constraint: $\Diamond\Box\exists\Diamond\, b \rightarrow \Box\Diamond\, b$.
  First consider its premise. As s_2 is reachable from every state in TS, $\mathit{Sat}(\exists\Diamond\, b) = S$. Thus every infinite path satisfies the premise of the weak fairness constraint. Each fair path must satisfy $\Box\Diamond\, b$, i.e., each fair path must visit either s_2 or s_4 infinitely often.

Note that by the strong fairness constraint, state s_1 is visited infinitely often. As s_2 is the only successor of s_2, s_2 is visited infinitely often. It thus suffices to consider paths that visit s_1 infinitely often, as they are the only fair paths.

We have to prove that $s_0 \models_{fair} \forall\Box\,(a \rightarrow \forall\Diamond\,(b \wedge \neg a))$. Therefore

$$\begin{aligned} & s_0 \models_{fair} \forall\Box\,(a \rightarrow \forall\Diamond\,(b \wedge \neg a)) \\ \iff & \forall \pi \in \mathit{FairPaths}(s_0).\ \pi \models_{fair} \Box\,(a \rightarrow \forall\Diamond\,(b \wedge \neg a)) \\ \iff & \forall \pi \in \mathit{FairPaths}(s_0).\ \forall i \geqslant 0.\ \pi[i] \models_{fair} (a \rightarrow \forall\Diamond\,(b \wedge \neg a))\,. \end{aligned}$$

As each fair path visits the $\neg a$-state s_2 (infinitely often), it suffices to show that

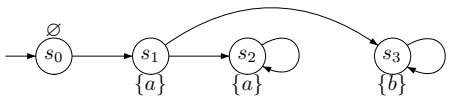
$$\begin{aligned} & \forall \pi \in \mathit{FairPaths}(s_0).\ \forall i \geqslant 0.\ \pi[i] \models_{fair} \forall\Diamond\,(b \wedge \neg a) \\ \iff & \forall \pi \in \mathit{FairPaths}(s_0).\ \forall i \geqslant 0.\ \forall \pi' \in \mathit{FairPaths}(\pi[i]).\ \pi' \models_{fair} \Diamond\,(b \wedge \neg a) \\ \iff & \forall \pi \in \mathit{FairPaths}(s_0).\ \forall i \geqslant 0.\ \underbrace{\forall \pi' \in \mathit{FairPaths}(\pi[i]).\ \exists j \geqslant 0.\ \pi'[j] \models_{fair} (b \wedge \neg a)}_{\text{true as } s_1 \text{ is visited infinitely often}} \end{aligned}$$

Therefore, $s_0 \models_{fair} \Phi$. As $I = \{\, s_0 \,\}$, this directly yields $TS \models_{fair} \Phi$.

Answer to Exercise 6.12
The following algorithm (see Algorithm 52) serves to check isomorphism of two ROBDDs, $\mathfrak{B}$ and $\mathfrak{C}$ say, with the same variable ordering. It relies on a simultaneous traversal of both ROBDDs, starting in their roots $v_0^{\mathfrak{B}}$ and $v_0^{\mathfrak{C}}$, respectively. The initial invocation is $\mathit{ISO}(v_0^{\mathfrak{B}}, v_0^{\mathfrak{C}})$. To avoid multiple calls of the algorithm with the same arguments, a hash table is used. This algorithm even runs in time $\mathcal{O}(\min\{\mathit{size}(\mathfrak{B}), \mathit{size}(\mathfrak{C})\})$, since it aborts as soon as a difference between the two ROBDDs has been detected.

Answer to Exercise 6.15

(a) We prove that there is no equivalent LTL-formula for the CTL-formula $\Phi_1 = \forall\Diamond\,(a \wedge \exists \bigcirc\, a)$ by exploiting Theorem 6.18. Removing the path-quantifiers in Φ yields the LTL-formula $\phi_1 = \Diamond\,(a \wedge \bigcirc\, a)$. We prove that $\Phi_1 \not\equiv \varphi_1$. Consider the transition system TS:



We have $TS \not\models_{LTL} \varphi_1$, because $s_0 s_1 s_3^\omega \not\models \varphi_1$. On the other hand, $TS \models_{CTL} \Phi_1$:

$$TS \models_{CTL} \forall\Diamond\,(a \wedge \exists \bigcirc\, a) \iff \forall \pi \in \mathit{Paths}(s_0).\ \exists j \geqslant 0.\ \pi[j] \models_{CTL} a \wedge \exists \bigcirc\, a$$

Choose $j = 1$. Then there exists a path $\pi' = s_1 s_2^\omega \in \mathit{Paths}(\pi[1])$ and $\pi'[1] \models a$.

$\implies TS \models_{CTL} \Phi_1$ and $TS \not\models_{LTL} \varphi_1$
$\implies \Phi_1 \not\equiv \varphi_1$
$\implies$ (by Theorem 6.18), there exists no equivalent LTL-formula for Φ_1.

Algorithm 52 Checking isomorphism of two ROBDDs, $ISO(v_1, v_2)$

Input: Two vertices v_1 and v_2 of equally ordered ROBDDs
Output: "yes", if ROBDDs rooted at v_1 and v_2 are isomorphic. Otherwise, "no".

---

if there is an entry (v_1, v_2, b) in the computed table **then**
  return b
else
  if v_1 and v_2 are terminal **then**
    $b := (val(v_1) = val(v_2))$
  else
    (* at least one of the nodes v_1 or v_2 is an inner node *)

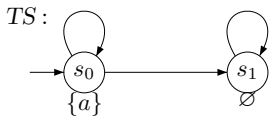
    if v_1 and v_2 are inner nodes with $var(v_1) = var(v_2)$ **then**
      $w_{0,1} := succ_0(v_1)$; $w_{1,1} := succ_1(v_1)$;
      $w_{0,2} := succ_0(v_2)$; $w_{1,2} := succ_1(v_2)$;

      (* apply the procedure recursively to $\langle w_{0,1}, w_{0,2}\rangle$ und $\langle w_{1,1}, w_{1,2}\rangle$ *)

      $b := ISO(w_{0,1}, w_{0,2}) \wedge ISO(w_{1,1}, w_{1,2})$
    else
      (* either exactly one of the nodes v_1 und v_2 is terminal *)
      (* or v_1 and v_2 have different variable labelings *)

      $b :=$ false
    fi
  fi
  insert (v_1, v_2, b) in the computed table and return b
fi

(b) Now, we prove that there exists no equivalent LTL-formula for the CTL-formula $\Phi_2 = \forall\Diamond\,\exists\bigcirc\,\forall\Diamond\,\neg a$ without using Theorem 6.18. This goes by contraposition. Assume φ_2 is an LTL-formula such that $\Phi_2 \equiv \varphi_2$. Consider the transition system TS:

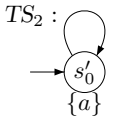


We have $Sat(\forall\Diamond\,\neg a) = \{s_1\}$. Therefore, $Sat(\exists\bigcirc\,\forall\Diamond\,\neg a) = \{s_0, s_1\}$ and $Sat(\Phi_2) = \{s_0, s_1\}$. It follows $TS_1 \models_{CTL} \Phi_2$.

Due to the assumption $\Phi_2 \equiv \varphi_2$, also $TS \models_{LTL} \varphi_2$ holds. Recall that:

$$TS \models_{LTL} \varphi_2 \iff \mathit{Traces}(TS) \subseteq \mathit{Words}(\varphi_2)$$

Now consider the trace $\sigma = \{a\}^\omega \in \mathit{Traces}(TS)$. Since $TS \models_{LTL} \varphi_2$, it follows $\sigma \in \mathit{Words}(\varphi_2)$. Consider now the transition system TS':



We have $\mathit{Traces}(TS') = \{\sigma\} \subseteq \mathit{Words}(\varphi_2)$. Thus, $TS' \models_{LTL} \varphi_2$. But, $TS' \not\models_{CTL} \Phi_2$ because $\neg a$ is not fulfilled by any state in TS'. Contradiction.

Answer to Exercise 6.16
We apply the CTL model checking algorithm.

(a) First consider the formula $\Phi_1 = \exists\Diamond\,\forall\Box\,c$:
It can be expressed equivalently in ENF:

$$\begin{aligned}\Phi_1 &= \exists\Diamond\,\forall\Box\,c\\ &\equiv \exists(\mathit{true}\,\mathsf{U}\,\forall\Box\,c)\\ &\equiv \exists(\mathit{true}\,\mathsf{U}\,\neg\exists\Diamond\,\neg c)\\ &\equiv \exists(\mathit{true}\,\mathsf{U}\,\neg\exists(\mathit{true}\,\mathsf{U}\,\neg c)).\end{aligned}$$

The bottom-up computation of the satisfaction sets yields:

- $Sat(\mathit{true}) = S$
- $Sat(c) = \{s_2, s_3, s_4\}$
- $Sat(\neg c) = \{s_0, s_1\}$
- $Sat(\exists\,(\mathit{true}\,\mathsf{U}\,\neg c))$ yields a backward search as follows:
  - $E = T = Sat(\neg c) = \{s_0, s_1\}$
  - Choose s_0: As $Pre(s_0) = \varnothing$, we get $E = \{s_1\}$
  - Choose s_1: $Pre(s_1) = \{s_0\}$. But $s_0 \notin Sat(\mathit{true}) \setminus T$ (i.e., it has already been visited), we get $E = \varnothing$

$\Longrightarrow T = \{s_0, s_1\}$.

- $Sat(\neg\exists\,(true\,\mathsf{U}\,\neg c)) = \{s_2, s_3, s_4\}$
- $Sat(\exists\,(true\,\mathsf{U}\,\neg\exists\,(true\,\mathsf{U}\,\neg c)))$ again yields a backward search:
  - $E = T = Sat(\neg\exists\,(true\,\mathsf{U}\,\neg c)) = \{s_2, s_3, s_4\}$
  - Choose s_2: $Pre(s_2) = \{s_2, s_3\}$, but all predecessors are already in T. $\Longrightarrow E = \{s_3, s_4\}$
  - Choose s_3: $Pre(s_3) = \{s_1\}$. Here we have $s_1 \notin T$ and $s_1 \in Sat(true)$. Therefore $T = T \cup \{s_1\} = \{s_1, s_2, s_3, s_4\}$ and $E = E \cup \{s_1\} = \{s_1, s_4\}$ (s_3 gets removed from E)
  - Choose s_1: $Pre(s_1) = \{s_0\}$. Again $s_0 \in Sat(true) \setminus T$ and therefore $T = T \cup \{s_0\} = S$ and $E = E \cup \{s_0\} = \{s_0, s_4\}$
  - Choose s_0: $Pre(s_0) = \varnothing$ and we directly continue with s_4:
  - Choose s_4: $Pre(s_4) = \{s_1, s_4\}$ but s_1, s_4 are already in T. Therefore we stop with $E = \varnothing$.

  $\Longrightarrow T = \{s_0, s_1, s_2, s_3, s_4\}$

Therefore, we have $Sat(\Phi_1) = \{s_0, s_1, s_2, s_3, s_4\}$ and $I \subseteq Sat(\Phi_1) \Longrightarrow TS \models \Phi_1$.

(b) $\Phi_2 = \forall(a\,\mathsf{U}\,\forall\Diamond\,c)$. First, we transform Φ_2 into ENF:

$$\begin{aligned} \forall(a\,\mathsf{U}\,\forall\Diamond\,c) &\equiv \neg\exists\,((\neg\forall\Diamond\,c)\,\mathsf{U}\,(\neg a \wedge \neg\forall\Diamond\,c)) \wedge \neg\exists\Box\,\neg c \qquad (*\exists\Box\,\exists\Box\,\Psi \equiv \exists\Box\,\Psi*) \\ &\equiv \neg\exists\,((\exists\Box\,\neg c)\,\mathsf{U}\,(\neg a \wedge \exists\Box\,\neg c)) \wedge \neg\exists\Box\,\neg c \end{aligned}$$

- $Sat(a) = \{s_0, s_1\}$ and $Sat(c) = \{s_2, s_3, s_4\}$.
- $Sat(\neg a) = \{s_2, s_3, s_4\}$ and $Sat(\neg c) = \{s_0, s_1\}$.
- $Sat(\exists\Box\,\neg c)$ is computed by a backward search starting in the $S \setminus Sat(\neg c)$ states:
  - $E = S \setminus Sat(\neg c) = \{s_2, s_3, s_4\}$ and $T = Sat(\neg c) = \{s_0, s_1\}$.
  - The counter array is initialized according to TS as follows:

    $$\begin{aligned} c[s_0] &= 1 \quad (*\ s_0 \text{ has only one successor } *) \\ c[s_1] &= 2 \quad (*\ s_3 \text{ and } s_4 \text{ are successors of } s_1\ *) \end{aligned}$$

  - Now, choose $s_2 \in E$ and check for each of its predecessors $Pre(s_2) = \{s_2, s_3\}$ whether it belongs to T. This is not the case, therefore we only remove s_2.
  - Choose $s_3 \in E$: We have $s_1 \in Pre(s_3) \cap T$. Therefore with s_3, we have found one successor of s_1 which violates $\neg c$. Therefore we decrement the number of possible successors of the state s_1

    $$c[s_1] = c[s_1] - 1 = 2 - 1 = 1$$

    and remove s_3 from the set E: $E = \{s_4\}$
  - Choose $s_4 \in E$: We have $Pre(s_4) = \{s_1, s_4\}$ and $s_1 \in T$. $\Longrightarrow c[s_1] = c[s_1] - 1 = 0$ and $E = \varnothing$. As $c[s_1] = 0$, s_1 is removed from T (all successor states violate $\neg c$) and included in E: $E = \{s_1\}$.
  - Choose $s_1 \in E$ with $Pre(s_1) = \{s_0\}$. As $s_0 \in T$, we decrement its counter $c[s_0] = 1 - 1 = 0$ and (because no successor state is left that could constitute a "good" path) remove it from $T = T \setminus \{s_0\} = \varnothing$. Then, s_0 is included in the set of "bad" states $E = \{s_0\}$. After another iteration, the algorithm stops with $T = \varnothing$.

The result is $Sat(\exists\Box\,\neg c) = T = \varnothing$

- $Sat(\neg\exists\Box\,\neg c) = S \setminus Sat(\exists\Box\,\neg c) = S$.
- $Sat(\neg a \wedge \exists\Box\,\neg c) = Sat(\neg a) \cap Sat(\exists\Box\,\neg c) = \{s_2, s_3, s_4\} \cap \varnothing = \varnothing$
- $Sat\big(\exists(\exists\Box\,\neg c)\,\mathsf{U}\,(\neg a \wedge \exists\Box\,\neg c)\big)$ invokes the $\exists\Psi_1\,\mathsf{U}\,\Psi_2$ backward search algorithm which directly terminates (as $E = Sat(\neg a \wedge \exists\Box\,\neg c) = \varnothing$).
  $\Longrightarrow T = \varnothing$
- $Sat\big(\neg\exists(\exists\Box\,\neg c)\,\mathsf{U}\,(\neg a \wedge \exists\Box\,\neg c)\big) = S$
- $Sat(\Phi_2) = Sat\big(\neg\exists(\exists\Box\,\neg c)\,\mathsf{U}\,(\neg a \wedge \exists\Box\,\neg c)\big) \cap Sat\big(\neg\exists\Box\,\neg c\big) = S \cap S = S$

$\Longrightarrow I \subseteq Sat(\Phi_2) \Longrightarrow TS \models \Phi_2$.

Answer to Exercise 6.18

(a) Prove that $Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$ is the largest set T such that

$$T \subseteq Sat(\Psi) \;\cup\; \{s \in Sat(\Phi) \mid Post(s) \cap T \neq \varnothing\}. \tag{B.1}$$

- We first prove that $T = Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$ satisfies (B.1). For $s \in T$ we obtain by the expansion law
$$\exists(\Phi\,\mathsf{W}\,\Psi) \quad \equiv \Psi \vee (\Phi \wedge \exists\bigcirc\,\exists(\Phi\,\mathsf{W}\,\Psi))$$
that either $s \in Sat(\Psi)$ or $s \in Sat(\Phi)$ and there exists $s' \in Post(s)$ with $s' \in T$. Hence, $s \in Sat(\Psi) \;\cup\; \{s \in Sat(\Phi) \mid Post(s) \cap T \neq \varnothing\}$. Therefore T satisfies (B.1).
- It remains to show that $Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$ is the *largest* set that satisfies (B.1). Let T be a set of states such that T satisfies (B.1). We prove that $T \subseteq Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$: Let $s \in T$. If $s \in Sat(\Psi)$ then $s \in Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$. Otherwise $s \in Sat(\Phi)$ and there exists $s_1 \in Post(s)$ with $s_1 \in T$. If $s_1 \in Sat(\Psi)$ then $\pi = ss_1$ satisfies $\Phi\,\mathsf{W}\,\Psi$ and $s \in Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$. Otherwise $s_1 \notin Sat(\Psi)$. Since $s_1 \in T$ there exists $s_2 \in Post(s_1)$ with $s_2 \in T$ and $s_1 \in Sat(\Phi)$. Continuing this inductive reasoning, we obtain either an infinite path π that satisfies $\Box\,\Phi$ or an initial path fragment $\pi = ss_1 \cdots s_n$ where $s_n \models \Psi$ and $s_i \models \Phi$ for $i < n$. In both cases, $s \in Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$.

Based on the above characterizations, the computation of the satisfaction sets can be done as follows:

Input: transition system TS without terminal states and state-formula $\exists(\Phi\,\mathsf{W}\,\Psi)$
Output: $Sat(\exists(\Phi\,\mathsf{W}\,\Psi))$

---

set of states $T = Sat(\Phi) \cup Sat(\Psi)$;

---

while $\exists s \in T$ with $s \notin Sat(\Psi)$ and $Post(s) \cap T = \varnothing$ **do**
  $T := T \setminus \{s\}$
od
return T;

(b) Prove that $Sat(\forall(\Phi \,\mathsf{W}\, \Psi))$ is the largest set T such that

$$T \subseteq Sat(\Psi) \;\cup\; \{s \in Sat(\Phi) \mid Post(s) \subseteq T\}. \tag{B.2}$$

- We first prove that $T = Sat(\forall(\Phi \,\mathsf{W}\, \Psi))$ satisfies (B.2). For $s \in T$ we obtain by the expansion law
$$\forall(\Phi \,\mathsf{W}\, \Psi) \quad \equiv \Psi \vee (\Phi \wedge \forall \bigcirc \forall(\Phi \,\mathsf{W}\, \Psi))$$
that either $s \in Sat(\Psi)$ or $s \in Sat(\Phi)$ and $s' \in T$ for all $s' \in Post(s)$. Hence $s \in Sat(\Psi) \;\cup\; \{s \in Sat(\Phi) \mid Post(s) \subseteq T\}$. Therefore T satisfies (B.2).
- It remains to show that $Sat(\forall(\Phi \,\mathsf{W}\, \Psi))$ is the largest set that satisfies (B.2). Let T be a set of states such that T satisfies (B.2). We prove that $T \subseteq Sat(\forall(\Phi \,\mathsf{W}\, \Psi))$: Let $s \in T$. If $s \in Sat(\Psi)$ then $s \in T$. Otherwise $s \in Sat(\Phi)$ and for all $s_1 \in Post(s_0)$ we have $s_1 \in T$. If $s_1 \in Sat(\Psi)$ then $\pi = ss_1$ satisfies $\Phi \,\mathsf{W}\, \Psi$. Otherwise $s_1 \notin Sat(\Psi)$. Since $s_1 \in T$, we have $Post(s_1) \subseteq T$ and $s_1 \in Sat(\Phi)$. Following this inductive reasoning all paths starting from s satisfy $\Phi \,\mathsf{W}\, \Psi$. Hence $s \in Sat(\forall(\Phi \,\mathsf{W}\, \Psi))$.

Based on the above characterizations, the computation of the satisfaction set can be done as follows:

Input: transition system TS without terminal states and state-formula $\forall(\Phi \,\mathsf{W}\, \Psi)$
Output: $Sat(\forall(\Phi \,\mathsf{W}\, \Psi))$

---

set of states $T = Sat(\Phi) \cup Sat(\Psi)$;

---

while $\exists s \in T$ with $s \notin Sat(\Psi)$ and $Post(s) \not\subseteq T$ **do**
  $T := T \setminus \{s\}$
od
return T;

Answer to Exercise 6.19

(a) We prove that for $TS_1 \subseteq TS_2$ and any arbitrary ECTL-formula Φ, we have

$$TS_1 \models \Phi \quad \Longrightarrow \quad TS_2 \models \Phi$$

We prove a slightly stronger claim, namely for all $s \in S_1$:

$$TS_1, s \models \Phi \quad \Longrightarrow \quad TS_2, s \models \Phi$$

In order to distinguish between TS_1 and TS_2, we annotate the considered transition system respectively. Induction on the syntactic structure of Φ:

- *Induction base:*
  - Let $\Phi = a \in AP$ and $TS_1, s \models \Phi$. Then $a \in L_1(s)$. Because of $TS_1 \subseteq TS_2$, we have $s \in S_2$ and $L_1(s) = L_2(s)$. Therefore $a \in L_2(s)$ and $TS_2, s \models a = \Phi$.

* Let $\Phi = \neg a$ for $a \in AP$ (Note that we only allow negations of atomic propositions!) and $TS_1, s \models \neg a$. Then $a \notin L_1(s) = L_2(s)$ (since $s \in S_2$). Therefore also $TS_2, s \models \neg a$.

– *Induction step:*

* Let $\Phi = \Phi_1 \wedge \Phi_2$:

$$\begin{aligned} TS_1, s \models \Phi &\iff TS_1, s \models \Phi_1 \text{ and } TS_1, s \models \Phi_2 \\ &\xrightarrow{\text{I.H.}} TS_2, s \models \Phi_1 \text{ and } TS_2, s \models \Phi_2 \\ &\iff TS_2, s \models \Phi \end{aligned}$$

* Let $\Phi = \exists \bigcirc \Psi$. We have

$$\begin{aligned} TS_1, s \models \Phi &\iff TS_1, s \models \exists \bigcirc \Psi \\ &\iff \exists \pi \in Paths_{TS_1}(s).\ \pi \models \bigcirc \Psi \\ &\iff \exists \pi \in Paths_{TS_1}(s).\ \pi[1] \models \Psi \\ &\iff \exists (s, \alpha, s') \in \rightarrow_1 .\ TS_1, s' \models \Psi \qquad (*\ \pi[1] = s'\ *) \\ &\xrightarrow{TS_1 \subseteq TS_2} s' \in S_2 \text{ and } (s, \alpha, s') \in \rightarrow_2 \text{ for some } \alpha \in Act \text{ and } TS_1, s' \models \Psi \\ &\xrightarrow{\text{I.H.}} \exists (s, \alpha, s') \in \rightarrow_2 .\ TS_2, s' \models \Psi \\ &\iff \exists \pi \in Paths_{TS_2}(s).\ \pi[1] \models \Psi \\ &\iff \exists \pi \in Paths_{TS_2}(s).\ \pi \models \bigcirc \Psi \\ &\iff TS_1, s \models \exists \bigcirc \Psi \end{aligned}$$

* Let $\Phi = \exists \Box \Psi$. We have

$$\begin{aligned} TS_1, s \models \Phi &\iff TS_1, s \models \exists \Box \Psi \\ &\iff \exists \pi \in Paths_{TS_1}(s).\ \pi \models \Box \Psi \\ &\iff \exists \pi \in Paths_{TS_1}(s).\ \forall i \geqslant 0.\ TS_1, \pi[i] \models \Psi \end{aligned}$$

Because of $S_1 \subseteq S_2$, $\pi[i] \in S_2$ for all $i \geqslant 0$. By induction hypothesis, we have for all $i \geqslant 0$:

$$TS_1, \pi[i] \models \Psi \quad \xrightarrow{\text{I.H.}} \quad TS_2, \pi[i] \models \Psi$$

As also $\rightarrow_1 \subseteq \rightarrow_2$, for all $i \geqslant 0$, we have $\pi[i] \xrightarrow{\alpha_i} \pi[i+1]$ for some $\alpha_i \in Act$. This shows that $\pi \in Paths_{TS_2}(s)$. Therefore we have $TS_2, \pi \models \Box \Psi$ and thus $TS_2, s \models \exists \Box \Psi$.

* Let $\Phi = \exists \Phi_1 \,\mathsf{U}\, \Phi_2$: Here, we have

$$\begin{aligned} TS_1, s \models \exists \Phi_1 \,\mathsf{U}\, \Phi_2 &\iff \exists \pi \in Paths_{TS_1}(s).\ \pi \models \Phi_1 \,\mathsf{U}\, \Phi_2 \\ &\iff \exists \pi \in Paths_{TS_1}(s).\ \exists j \geqslant 0.\ \pi[j] \models \Phi_2 \wedge \forall i < j.\ \pi[i] \models \Phi_1 \end{aligned}$$

From $S_1 \subseteq S_2$ it follows that $\pi[j] \in S_2$ and that $\pi[i] \in S_2$ for all $i \in \{0, \ldots, j-1\}$. Applying the induction hypothesis yields:

$$\begin{aligned} TS_1, \pi[j] \models \Phi_2 &\implies TS_2, \pi[j] \models \Phi_2 \\ TS_1, \pi[i] \models \Phi_1 &\implies TS_2, \pi[i] \models \Phi_1 \quad \text{for all } i \in \{0, \ldots, j-1\} \end{aligned}$$

As $\rightarrow_1 \subseteq \rightarrow_2$, $\pi \in Paths_{TS_2}(s)$. Therefore $TS_2, \pi \models \Phi_1 \,\mathsf{U}\, \Phi_2$ and with the existence of π in TS_2, also $TS_2, s \models \exists \Phi_1 \,\mathsf{U}\, \Phi_2$.

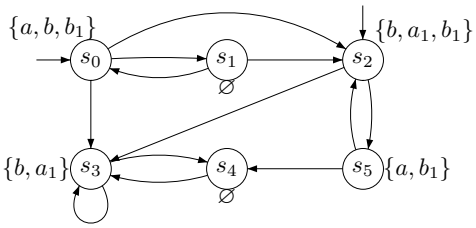
(b) Consider the CTL-formula $\Phi = \forall\Box\, a$. It cannot be expressed in ECTL as the quantification over all paths would need to be expressed using negated existential quantification. But as negations are allowed only wrt. atomic propositions, this is not possible in ECTL.

Answer to Exercise 6.21

(a) $Sat(b \wedge \neg a) = \{ s_2, s_3 \}$ and $Sat(\exists(b \,\mathsf{U}\, (a \wedge \neg b)) = \{ s_0, s_2, s_5 \}$.

(b) Let $fair = \Box\Diamond \underbrace{(b \wedge \neg a)}_{\Phi_1} \rightarrow \Box\Diamond \underbrace{\exists\,(b \,\mathsf{U}\, (a \wedge \neg b))}_{\Psi_1}$.

Introduce new atomic propositions a_1 (representing Φ_1) and b_1 (representing Ψ_1) and extend the labeling of TS accordingly:



The strongly connected components of TS are:

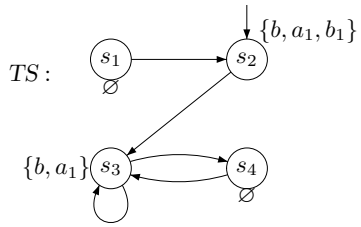
$$\begin{aligned} C_1 &= \{s_0, s_1\} \\ C_2 &= \{s_2, s_5\} \\ C_3 &= \{s_3, s_4\} \end{aligned}$$

Each execution fragment ultimately stays in one of these SCCs. According to the fairness assumption *fair*, there is no fair path visiting states in SCC C_3 infinitely often. SCC C_3 is thus of no relevance further. It now follows $Sat_{fair}\big(\exists\Box\,\mathsf{true}\big) = \{ s_0, s_1, s_2, s_5 \}$ as from these states either SCC C_1 or C_2 can be reached.

(c) First, we extend the labeling of states with the new atomic proposition a_{fair} such that $a_{fair} \in L(s)$ iff $s \in Sat_{fair}\big(\exists\Box\,\mathsf{true}\big)$. Now consider the CTL-formula $\Phi = \forall\Box\forall\Diamond\, a$. Rewriting Φ into existential normal form yields:

$$\begin{aligned} \Phi &= \forall\Box\,\forall\Diamond\, a \\ &= \neg\exists\Diamond\,\neg\forall\Diamond\, a \\ &= \neg\exists\Diamond\,\exists\Box\,\neg a \\ &= \neg\exists\,(\mathsf{true}\;\mathsf{U}\,\exists\Box\,\neg a)\,. \end{aligned}$$

– Compute the fair satisfaction set for subformula $\Phi = \exists\Box\,\neg a$: The state subgraph $G[\neg a]$ of TS is



The only SCC in $G[\neg a]$ is C_3. But we have

$$C_3 \cap Sat(a_1) \neq \varnothing \quad \text{and} \quad C_3 \cap Sat(b_1) = \varnothing.$$

Therefore $T = \varnothing$ and $Sat_{fair}(\exists\Box\,\neg a) = \big\{s \in S \mid Reach_{G[\neg a]}(s) \cap T \neq \varnothing\big\} = \varnothing$. Introduce new atomic proposition $a_{\exists\Box\,\neg a}$ and extend the labeling of TS according to $Sat_{fair}(\exists\Box\,\neg a)$ (In this case, no state labels are extended since $Sat_{fair}(\exists\Box\,\neg a) = \varnothing$).

– Now consider $\Phi = \exists\,(\mathsf{true}\ \mathsf{U}\ a_{\exists\Box\,\neg a})$:

$$Sat_{fair}(\exists\,(\mathsf{true}\ \mathsf{U}\ a_{\exists\Box\,\neg a})) = Sat(\exists\,(\mathsf{true}\ \mathsf{U}\ (a_{\exists\Box\,\neg a} \wedge a_{fair}))) = \varnothing$$

– Therefore $Sat_{fair}(\neg a_{\exists(\mathsf{true}\ \mathsf{U}\ \exists\Box\,\neg a)}) = \Big\{s \in S \mid a_{\exists(\mathsf{true}\ \mathsf{U}\ \exists\Box\,\neg a)} \notin L(s)\Big\}$.
This yields $Sat_{fair}(\neg a_{\exists(\mathsf{true}\ \mathsf{U}\ \exists\Box\,\neg a)}) = S$.

Answer to Exercise 6.24
We consider the maximal proper state subformulas $Sub(\Phi)$:

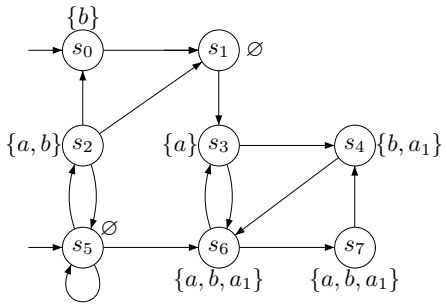
1. $\Psi = a$: $Sat(a) = \{s_2, s_3, s_6, s_7\}$
2. $\Psi = b$: $Sat(b) = \{s_0, s_2, s_4, s_6, s_7\}$
3. $\Psi = \exists\Box\,b$:
   The following equivalence is used to compute $Sat(\exists\Box\,b)$:

$$s \models_{CTL^*} \exists\varphi \quad \Longleftrightarrow \quad s \models_{CTL^*} \neg\forall\neg\varphi \quad \Longleftrightarrow \quad s \not\models_{CTL^*} \forall\neg\varphi \quad \Longleftrightarrow \quad s \not\models_{LTL} \neg\varphi$$

   According to the LTL semantics, we have $Sat_{LTL}(\neg\Box\,b) = Sat_{LTL}(\Diamond\,\neg b) = \{s_0, s_1, s_2, s_3, s_5\}$. Then, $S \setminus Sat_{LTL}(\neg\Box\,b) = \{s_4, s_6, s_7\}$ is the satisfaction set $Sat_{CTL^*}(\exists\Box\,b)$:

$$Sat_{CTL^*}(\exists\Box\,b) = \{s_4, s_6, s_7\}.$$

   The labeling is extended by a fresh atomic proposition a_1 that uniquely labels all states in to $Sat_{CTL^*}(\exists\Box\,b)$.
   The corresponding subformula $\exists\Box\,b$ of Φ is replaced by a_1. This yields:



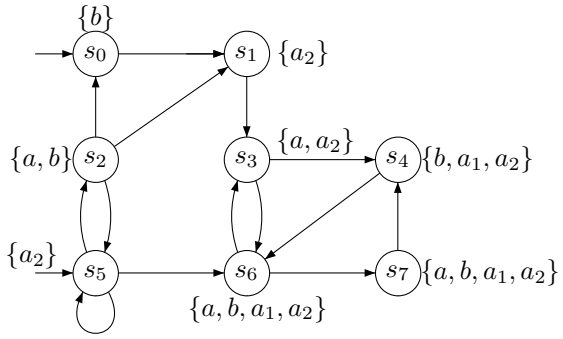
4. $\Psi = \exists \bigcirc (a \,\mathsf{U}\, a_1)$:
   The above equivalence for existentially quantified path formulas yields:

   $$s \models_{CTL^*} \exists \bigcirc (a \,\mathsf{U}\, a_1) \quad \Longleftrightarrow \quad s \not\models_{LTL} \neg \bigcirc (a \,\mathsf{U}\, a_1)\,.$$

   By the equivalence $\neg \bigcirc (a \,\mathsf{U}\, a_1) \equiv \bigcirc \neg (a \,\mathsf{U}\, a_1)$, the satisfaction set of $\neg (a \,\mathsf{U}\, a_1)$ can be inferred:

   $$\begin{aligned} Sat_{LTL}(\neg (a \,\mathsf{U}\, a_1)) &= \{s_0, s_1, s_2, s_5\} \\ Sat_{LTL}(\bigcirc \neg (a \,\mathsf{U}\, a_1)) &= \{s_0, s_2\} \\ Sat_{CTL^*}(\exists \bigcirc (a \,\mathsf{U}\, a_1)) &= S \setminus Sat_{LTL}(\bigcirc \neg (a \,\mathsf{U}\, a_1)) \\ &= S \setminus \{s_0, s_2\} \\ &= \{s_1, s_3, s_4, s_5, s_6, s_7\} \end{aligned}$$

   The labeling is extended by a new atomic proposition a_2 that labels all states in the set $Sat_{CTL^*}(\exists \bigcirc (a \,\mathsf{U}\, a_1))$. This yields:



   Again, the corresponding subformula Ψ of Φ is replaced by a_2.
5. $\Psi = \forall \Diamond \Box\, a_2$:
   In the case of universal quantification, we can directly apply the LTL-semantics:

   $$Sat_{LTL}(\Diamond \Box\, a_2) = \{s_0, s_1, s_3, s_4, s_6, s_7\}\,.$$

Because of $s_5 \in Q_0$, but $s_5 \notin Sat(\Phi)$, this yields $TS \not\models_{CTL^*} \Phi$.

Answer to Exercise 6.29
$\forall\Box\Diamond a$

Answer to Exercise 6.31

(a) The encoding of Pnueli's algorithm into NuSMV is rather straightforward. Besides the joint semaphore and the two variables y_0 and y_1 (that are stored in array `y`), each process is equipped with a local variable `pc` that acts as its program counter. The code reads as follows:

```
MODULE main

VAR y : array 0..1 of 0..1;
    s : 0..1;

    prc0 : process prc(0,y,s);
    prc1 : process prc(1,y,s);

ASSIGN

    init(y[0]) := 0..1;
    init(y[1]) := 0..1;
    init(s)    := 0..1;

MODULE prc(i,y,s)

VAR pc : {l1, l2, l3, l4, l5} ;

ASSIGN

    init(pc) := l1 ;

    next(pc) :=
      case
        (pc = l1)                              : l2 ;
        (pc = l2)                              : l3 ;
        (pc = l3) & ((y[1-i] = 0) | !(s = i)) : l4 ;
        (pc = l4)                              : l5 ;
        (pc = l5)                              : l1 ;
        !(pc = l2) & !(pc = 5)                 : pc ;
      esac;

    next(y[i]) :=
      case
        (pc = l2)              : 1 ;
        (pc = l5)              : 0 ;
        !(pc = l2) & !(pc = 5) : y[i] ;
```

```
    esac;

  next(s) :=
    case
      (pc = l2)  : i ;
      !(pc = l2) : s ;
    esac;
```

The mutual exclusion property can now be checked by extending the anbove NuSMV code with the CTL-formula:

```
SPEC !EF ( (prc0.pc = l4) & (prc1.pc = l4) )
```

Verification indeed shows that this property is satisfied.

(b) To check that Pnueli's protocol ensures absence of unbounded overtaking, i.e., when a process wants to enter its critical section, it eventually will be able to do so, we use the following CTL-formula:

```
AG ((y[0] = 1 -> AF (prc0.pc = l4)) & (y[1] = 1 -> AF (prc1.pc = l4)))
```

It is stating that it is always the case that in any state in which process 0 wants to enter its critical section (i.e., when `y[0] = 1`), then it always eventually will enter its critical section (i.e., `prc0.pc = l4`). Similarly, we require the same for process 1. Note that the formula:

```
AG ((y[0] = 1 -> EF (prc0.pc = l4)) & (y[1] = 1 -> EF (prc1.pc = l4)))
```

is stating that it is always the case that if process 0 wants to enter its critical section, it *potentially* is able to do so. (And similar for process 1). An execution in which process 0, however, never enters the critical section is then still possible, whereas such run would violate the first formulation (with `AF` rather than `EF`). We, therefore, prefer the first formulation.

Verifying the first property with NuSMV shows that the property is not satisfied; verifying the second property does indeed succeed. The following counter-example is generated:

```
-- specification AG ((y[0] = 1 -> AF prc0.pc = l4) &
--                   (y[1] = 1 -> AF prc1.pc = l4)) is false
-- as demonstrated by the following execution sequence
State 1.1:
_process_selector_ = main
y[0] = 0
y[1] = 0
s = 0
prc0.pc = l1
prc1.pc = l1
State 1.2:
[executing process prc1]
_process_selector_ = prc1
y[0] = 0
y[1] = 0
s = 0
```

```
prc0.pc = l1
prc1.pc = l1
State 1.3:
[executing process prc1]
_process_selector_ = prc1
y[0] = 0
y[1] = 0
s = 0
prc0.pc = l1
prc1.pc = l2
-- loop starts here --
State 1.4:
_process_selector_ = main
y[0] = 0
y[1] = 1
s = 1
prc0.pc = l1
prc1.pc = l3
State 1.5:
_process_selector_ = main
y[0] = 0
y[1] = 1
s = 1
prc0.pc = l1
prc1.pc = l3
```

(c) Note that in this detected loop (consisting of states `1.4` and `1.5`) only the `main` process is performing a step whereas all other processes are (and remain) idle. This unfair execution sequence is an undesired computation, as we expect that each individual process gets its turn every now and then. Therefore, we impose a fairness constraint on each process by incorporaying the following statement in each process `prc`:

```
FAIRNESS running
```

Verifying the new specification yields that the property

```
AG ((y[0] = 1 -> AF (prc0.pc = l4)) & (y[1] = 1 -> AF (prc1.pc = l4)))
```

is indeed satisfied.

Conclusion: Pnueli's algorithm suffers from unbounded overtaking in absence of any fairness constraint, and does not have this problem in case each process gets its turn infinitely often.

(d) The formulation in CTL that each process will occupy its critical section infinitely often is as follows:

```
(AG AF (prc0.pc = l4)) & (AG AF (prc1.pc = l4))
```

as for the unbounded overtaking case, this property turns out to be false in case the fairness constraints

```
FAIRNESS running
```

are absent, whereas the result is true in case the fairness constraint is present.

(e) There are four options to replace the atomic assignments to `s`, `y[0]` and `y[1]`. These four options are simply the 4 different orderings in which the assignments to `s`, `y[0]` and `y[1]` can be made: `sy0sy1`, `y0sy1s`, `y0ssy1` and `sy0y1s`. For instance, alternative `y0sy1s` refers to the ordering of assignments as given in the following NuSMV specification:

```
MODULE main

VAR y : array 0..1 of 0..1;
    s : 0..1;

    prc0 : process prc(0,y,s);
    prc1 : process prc(1,y,s);

ASSIGN

    init(y[0]) := 0..1;
    init(y[1]) := 0..1;
    init(s)    := 0..1;

MODULE prc(i,y,s)

VAR pc : {l1, l2a, l2b, l3, l4, l5} ;

ASSIGN

    init(pc) := l1 ;

    next(pc) :=
      case
        (pc = l1)                              : l2a ;
        (pc = l2a)                             : l2b ;
        (pc = l2b)                             : l3 ;
        (pc = l3) & ((y[1-i] = 0) | !(s = i)) : l4 ;
        (pc = l4)                              : l5 ;
        (pc = l5)                              : l1 ;
        !(pc = l2a) & !(pc = l2b) & !(pc = 5) : pc ;
      esac;

    next(y[i]) :=
      case
        (pc = l2a)              : 1 ;
        (pc = l5)               : 0 ;
        !(pc = l2a) & !(pc = 5) : y[i] ;
      esac;
```

```
next(s) :=
  case
    (pc = l2b)  : i ;
    !(pc = l2b) : s ;
  esac;
```

Note that line `2` of the NuSMV code of the original algorithm – in which both `y[i]` and `s` are assigned a value simultaneously – is now splitted into lines `2a` and `2b` such that first the assignment to `y[0]` is made and then the assignment to `s`. The other variants are obtained in a similar way (although the non-symmetric variants `y0ssy1` and `sy0y1s` require some small additional changes).

Verification with NuSMV now yields that only the implementation `y0sy1s` is correct, i.e., satisfies the mutual exclusion property.

Answer to Exercise 6.32

(a) The NuSMV code for tiles 0 (the blank tile) and tile 1, as a representative for the code of a non-blank tile, is as follows:

```
ASSIGN

-- determine the next positions of the blank tile

next(h[0]) :=  -- horizontal position of the blank tile
       case
         (move = l) & !(h[0] = 1) : h[0] - 1;  -- one position down
         (move = r) & !(h[0] = N) : h[0] + 1;  -- one position up
                                1 : h[0];      -- no change
       esac;

next(v[0]) :=  -- vertical position of the blank tile
       case
         (move = d) & !(v[0] = 1) : v[0] - 1;  -- one position down
         (move = u) & !(v[0] = K) : v[0] + 1;  -- one position up
                                1 : v[0];      -- no change
       esac;

-- determine the next positions of all non-blank tiles

next(h[1]) :=  -- horizontal position of tile 1
       case    -- swap horizontal position with blank tile if appropriate
         (move = l) & (v[1] = v[0]) & (h[1] = h[0] - 1) : h[0];
         (move = r) & (v[1] = v[0]) & (h[1] = h[0] + 1) : h[0];
                                                  1 : h[1];
       esac;
```

```
next(v[1]) :=  -- vertical position of tile 1
       case    -- swap vertical position with blank tile if appropriate
         (move = d) & (h[1] = h[0]) & (v[1] = v[0] - 1) : v[0];
         (move = u) & (h[1] = h[0]) & (v[1] = v[0] + 1) : v[0];
                                                    1 : v[1];
       esac;

-- and similar for all remaining tiles
```

The blank tile can only be moved one position to the left if it is not occupying some position in the leftmost column, that is, if `!(h[0] = 1)` is valid. Similarly, it can be moved one position to the right if it is not occupying some position in the rightmost column, i.e., if `!(h[0] = N)`. In all other cases, the horizontal position is unchanged. These three cases are defined by `next(h[0])`. In a completely similar way, the vertical moves of the blank tile are defined.

The basic idea for the moves of the non-blank tiles is that in each move we swap the blank tile with the tile that has actually been moved. Thus, in case the blank tile is moved one position to the left, then the new horizontal position of tile i becomes `h[0]` if and only if tile i is located in the same row (i.e., the vertical position of tile i and the blank tile are the same), and tile i is located one position to the left of the blank tile. This gives rise to the condition:

```
(move = l) & !(h[0] = 1) & (v[1] = v[0]) & (h[1] = h[0] - 1)
```

for the tile with $i = 1$. The first two conjuncts ensure that the blank tile can indeed be moved to the left, while the latter two conjuncts characterise that `tile[1]` is the right position to be moved one position to the right, i.e., to be swapped with the blank tile. Note that the second conjunct is superfluous due to the fourth conjunct, so we can simplify the above condition into:

```
(move = l) & (v[1] = v[0]) & (h[1] = h[0] - 1)
```

In a similar way, moves to the right, down and up (of the blank tile) are considered.

(b) The desired goal configuration of the puzzle is defined as follows:

```
goal :=
 ( (h[0] = 3 & v[0] = 1) & (h[1] = 2 & v[1] = 1) & (h[2] = 1 & v[2] = 1)
 & (h[3] = 3 & v[3] = 2) & (h[4] = 2 & v[4] = 2) & (h[5] = 1 & v[5] = 2)
 & (h[6] = 3 & v[6] = 3) & (h[7] = 2 & v[7] = 3) & (h[8] = 1 & v[8] = 3));
```

This corresponds to the goal configuration as depicted in the exercise.

(c) A possible solution to the puzzle is obtained by stating that the `goal` configuration is *not* reachable. In case the goal configuration is reachable, the model checker NuSMV will provide a counter-example, consisting of all required moves of the tiles, that leads us to the goal configuration. Verifying the property

```
SPEC  !EF goal
```

yields (after a run-time of about 3 minutes) the following sequence of 28 moves that changes the initial configuration into the desired goal configuration:

```
r r d d l l u u r r d d l l u u r r d d l l u u r r d d r
```

Exercises of Chapter 7

Answer to Exercise 7.1

- $TS_3 \not\sim TS_1, TS_2, TS_4$. Intuitively, this is because TS_3 does not have a terminal state while TS_1, TS_2, and TS_4 have terminal states. Note that a terminal state violates $\exists\bigcirc\,\mathrm{true}$. Consider the CTL-formula
$$\Phi \;=\; \forall\bigcirc\,\exists\bigcirc\,\mathrm{true}$$
It follows $TS_3 \models_{CTL} \Phi$ but $TS_1, TS_2, TS_4 \not\models_{CTL} \Phi$.
- $TS_1 \not\sim TS_2, TS_4$. A CTL formula that distinguishes TS_1 from TS_2 and TS_4 is:
$$\Phi \;=\; \exists\bigcirc\,\exists\bigcirc\,(a \wedge \neg b)$$
Then $TS_1 \not\models_{CTL} \Phi$ but $TS_2 \models_{CTL} \Phi$ and $TS_4 \models_{CTL} \Phi$.
- $TS_2 \not\sim TS_4$. A CTL formula that distinguishes TS_2 from TS_4 is:
$$\Phi \;=\; \exists\bigcirc\,\exists\bigcirc\,(a \wedge b)$$
Then $TS_2 \models_{CTL} \Phi$ but $TS_4 \not\models_{CTL} \Phi$.

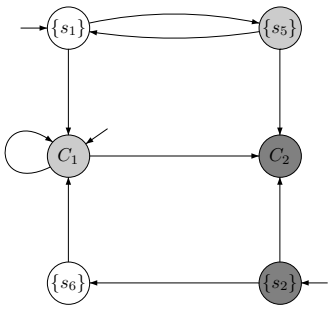
Answer to Exercise 7.5(b)
If $\mathcal{R}$ is an action-based bisimulation for TS then

$$\mathcal{R}_{state} = \mathcal{R} \cup \{(\langle s_1, \alpha\rangle, \langle s_2, \alpha\rangle) \mid (s_1, s_2) \in \mathcal{R}, \alpha \in Act\}$$

is a bisimulation for TS_{state}. Vice versa, given a bisimulation $\mathcal{R}_{state}$ for TS_{state} an action-based bisimulation for TS is obtained by all pairs $(s_1, s_2) \in S \times S$ such that $(s_1, s_2) \in \mathcal{R}_{state}$ or $(\langle s_1, \alpha\rangle, \langle s_2, \alpha\rangle) \in \mathcal{R}_{state}$ for some action $\alpha \in Act$.

Answer to Exercise 7.7

(a) The quotient transition system wrt. $\sim_{TS}$ is $TS' = (S', Act', \rightarrow', I', AP, L')$, where $S' = \{\{s_1\}, \{s_2\}, \{s_5\}, \{s_6\}, C_1, C_2\}$ with $C_1 = \{s_3, s_4, s_8, s_9, s_{11}\}$ and $C_2 = \{s_7, s_{10}, s_{12}, s_{13}\}$. Note that $s_5 \not\sim_{TS} s_{11}$ as s_5 has no direct successor labeled with $\{a, b\}$ whereas s_{11} does. Therefore, $s_1 \not\sim_{TS} s_6$ as $s_1 \rightarrow s_5$ and this cannot be matched by s_2. The other components of TS' are given in the pictorial representation:



where the white states are labeled with $\varnothing$, the light gray states with $\{a, b\}$ and the dark gray states with $\{a\}$.

(b) The master formula for each bisimulation equivalence class is defined as:

$$\Phi_{\{s_1\}} = \neg a \wedge \neg b \wedge \exists \bigcirc \exists \bigcirc (\neg a \wedge \neg b) \qquad \Phi_{\{s_2\}} = a \wedge \exists \bigcirc a$$
$$\Phi_{\{s_6\}} = \neg a \wedge \neg b \wedge \neg\exists \bigcirc \exists \bigcirc (\neg a \wedge \neg b) \qquad \Phi_{\{s_5\}} = a \wedge b \wedge \neg\exists \bigcirc (a \wedge b)$$
$$\Phi_{C_1} = a \wedge b \wedge \exists \bigcirc (a \wedge b) \qquad \Phi_{C_2} = a \wedge \neg b \wedge \neg\exists \bigcirc a$$

Answer to Exercise 7.8

(a) The execution of the inefficient quotienting algorithm is shown in Table B.

(b) The steps of applying the improved and efficient partition-refinement algorithm are shown in Table B.

Answer to Exercise 7.18
Note: there is an annoying typo in the exercise: $\sqsubseteq$ should read $\trianglelefteq$ and $\cong$ should read $\triangleq$, i.e., we consider stutter trace-inclusion and stutter trace-equivalence.

Let us discuss the solution of the exercise:

- $TS_2 \not\trianglelefteq TS_1$, since $ab^\omega \in \mathit{Traces}(TS_2)$, but there is path in TS_1 whose trace is stutter equivalent to ab^ω as there is no a-cycle in TS_1.

  $TS_1 \trianglelefteq TS_2$. This can be seen as follows. States s_1 and s_3, as well as s_2 and s_4 in TS_1 can be grouped, yielding TS_1', say, and it is easy to see that $TS_1 \triangleq TS_1'$. We have $\mathit{Traces}(TS_1') = \{(ab)^\omega, (ab)^* a^\omega\}$. It suffices to show that for each of these traces, there is a stutter-equivalent path in TS_2. For $(ab)^\omega$ this is $(t_1\, t_2)^\omega$, whereas for $(ab)^* a^\omega$ this is the path $(t_1\, t_2)^* t_1^\omega$.

- $TS_1 \not\trianglelefteq TS_3$, since $(ab)^* a^\omega \in \mathit{Traces}(TS_1)$, but there is no a-cycle reachable once a b-state is visited in TS_3.

  $TS_3 \trianglelefteq TS_1$. This can be seen as follows. Traces in TS_3 are of the form –ignoring stuttering– either $(ab)^\omega$ or a^ω. The trace of path $s_1 s_5^\omega$ in TS_1 is a^ω whereas the trace of path $(s_1 s_2)^\omega$ equals $(ab)^\omega$.

Outer Iteration			
1	$\Pi_{old} = \Pi_{AP} = \Big\{\{s_1, s_6\}, \{s_3, s_4, s_5, s_8, s_9, s_{11}\}, \{s_2, s_7, s_{10}, s_{12}, s_{13}\}\Big\}$ Inner iteration		
	1	$C = \{s_1, s_6\}$, $Pre(C) = \{s_2, s_5\}$ $\Pi = \Big\{\{s_1, s_6\}, \underline{\{s_5\}}, \underline{\{s_3, s_4, s_8, s_9, s_{11}\}}, \underline{\{s_2\}}, \underline{\{s_7, s_{10}, s_{12}, s_{13}\}}\Big\}$
	2	$C = \{s_3, s_4, s_5, s_8, s_9, s_{11}\}$, $Pre(C) = \{s_1, s_3, s_4, s_6, s_8, s_9, s_{11}\}$; Π unaffected
	3	$C = \{s_2, s_7, s_{10}, s_{12}, s_{13}\}$, $Pre(C) = \{s_2, s_3, s_4, s_5, s_8, s_9, s_{11}\}$; Π unaffected
2	$\Pi_{old} = \Pi_1 = \Big\{\{s_1, s_6\}, \{s_2\}, \{s_3, s_4, s_8, s_9, s_{11}\}, \{s_5\}, \{s_7, s_{10}, s_{12}, s_{13}\}\Big\}$ Inner iteration		
	1	$C = \{s_1, s_6\}$, $Pre(C) = \{s_2, s_5\}$; $\Pi = \Pi_{old}$, unaffected
	2	$C = \{s_2\}$, $Pre(C) = \varnothing$; $\Pi = \Pi_{old}$, unaffected
	3	$C = \{s_3, s_4, s_8, s_9, s_{11}\}$, $Pre(C) = \{s_1, s_3, s_4, s_6, s_8, s_9, s_{11}\}$; $\Pi = \Pi_{old}$, unaffected
	4	$C = \{s_5\}$, $Pre(C) = \{s_1\}$ $\Pi = \Big\{\underline{\{s_1\}}, \underline{\{s_6\}}, \{s_2\}, \{s_3, s_4, s_8, s_9, s_{11}\}, \{s_5\}, \{s_7, s_{10}, s_{12}, s_{13}\}\Big\}$
	5	$C = \{s_7, s_{10}, s_{12}, s_{13}\}$, $Pre(C) = \{s_2, s_3, s_4, s_5, s_8, s_9, s_{11}\}$; Π unaffected
3	$\Pi_{old} = \Pi_2 = \Big\{\{s_1\}, \{s_6\}, \{s_2\}, \{s_3, s_4, s_8, s_9, s_{11}\}, \{s_5\}, \{s_7, s_{10}, s_{12}, s_{13}\}\Big\}$ Inner iteration		
	1	$C = \{s_1\}$, $Pre(C) = \{s_5\}$; $\Pi = \Pi_{old}$, unaffected
	2	$C = \{s_6\}$, $Pre(C) = \{s_2\}$; $\Pi = \Pi_{old}$, unaffected
	3	$C = \{s_2\}$, $Pre(C) = \varnothing$; $\Pi = \Pi_{old}$, unaffected
	4	$C = \{s_3, s_4, s_8, s_9, s_{11}\}$, $Pre(C) = \{s_1, s_3, s_4, s_6, s_8, s_9, s_{11}\}$; $\Pi = \Pi_{old}$, unaffected
	5	$C = \{s_5\}$, $Pre(C) = \{s_1\}$; $\Pi = \Pi_{old}$, unaffected
	6	$C = \{s_7, s_{10}, s_{12}, s_{13}\}$, $Pre(C) = \{s_2, s_3, s_4, s_5, s_8, s_9, s_{11}\}$; $\Pi = \Pi_{old}$, unaffected
	$\Pi_3 = \Pi_{old}$, the algorithm terminates.		

Table B.1: Applying the first partition-refinement algorithm

Init.	$\Pi_{old} := \{S\}$	
	$\Pi := \mathit{Refine}(\Pi_{AP}, S) = \left\{\{s_1, s_6\}, \{s_3, s_4, s_5, s_8, s_9, s_{11}\}, \{s_2, s_7, s_{10}, s_{12}, s_{13}\}\right\}$	
It.1	$\Pi_{old} \neq \Pi$	
	Choose $C = \{s_1, s_6\}$ and $C' = S$	
	$\Pi_{old} := \Pi$	
	Compute $\mathit{Refine}(\Pi, C, C' \setminus C)$	
	$B = \{s_1, s_6\}$	$B_1 = \varnothing$ $B_2 = \varnothing$ $B_3 = \{s_1, s_6\}$
	$B = \{s_3, s_4, s_5, s_8, s_9, s_{11}\}$	$B_1 = \{s_5\}$ $B_2 = \varnothing$ $B_3 = \{s_3, s_4, s_8, s_9, s_{11}\}$
	$B = \{s_2, s_7, s_{10}, s_{12}, s_{12}\}$	$B_1 = \{s_2\}$ $B_2 = \varnothing$ $B_3 = \varnothing$
	$\Pi := \left\{\{s_1, s_6\}, \{s_5\}, \{s_3, s_4, s_8, s_9, s_{11}\}, \{s_2\}, \{s_7, s_{10}, s_{12}, s_{13}\}\right\}$	
It.2	$\Pi_{old} \neq \Pi$	
	Choose $C = \{s_5\}$ and $C' = \{s_3, s_4, s_5, s_8, s_9, s_{11}\}$	
	$\Pi_{old} := \Pi$	
	Compute $\mathit{Refine}(\Pi, C, C' \setminus C)$	
	$B = \{s_1, s_6\}$	$B_1 = \{s_1\}$ $B_2 = \varnothing$ $B_3 = \{s_6\}$
	$B = \{s_3, s_4, s_8, s_9, s_{11}\}$	$B_1 = \varnothing$ $B_2 = \varnothing$ $B_3 = \{s_3, s_4, s_8, s_9, s_{11}\}$
	$B = \{s_7, s_{10}, s_{12}, s_{13}\}$	$B \cap \mathit{Pre}(C') = \varnothing$, and B is stable wrt. C and $C' \setminus C$
	$\Pi := \left\{\{s_1\}, \{s_6\}, \{s_5\}, \{s_3, s_4, s_8, s_9, s_{11}\}, \{s_2\}, \{s_7, s_{10}, s_{12}, s_{13}\}\right\}$	
It.3	$\Pi_{old} \neq \Pi$	
	Choose $C = \{s_6\}$ and $C' = \{s_1, s_6\}$	
	$\Pi_{old} := \Pi$	
	Compute $\mathit{Refine}(\Pi, C, C' \setminus C)$	
	$B = \{s_3, s_4, s_8, s_9, s_{11}\}$	$B \cap \mathit{Pre}(C') = \varnothing$, and B is stable wrt. C and $C' \setminus C$
	$B = \{s_7, s_{10}, s_{12}, s_{13}\}$	$B \cap \mathit{Pre}(C') = \varnothing$, and B is stable wrt. C and $C' \setminus C$
	$\Pi := \left\{\{s_1\}, \{s_6\}, \{s_5\}, \{s_3, s_4, s_8, s_9, s_{11}\}, \{s_2\}, \{s_7, s_{10}, s_{12}, s_{13}\}\right\}$	
	$\Pi_{old} = \Pi$	

Table B.2: Applying the efficient bisimulation quotienting algorithm

- $TS_2 \not\trianglelefteq TS_3$, since $(ab)^*a^\omega \in \mathit{Traces}(TS_2)$, but there is no a-cycle reachable once a b-state is visited in TS_3.

  $TS_3 \trianglelefteq TS_2$. By a similar reasoning as for $TS_3 \trianglelefteq TS_1$.

Answer to Exercise 7.19

- $TS_1 \not\approx TS_2$. This can be seen as follows. Consider state s_5 in TS_1. From this state, only a state labeled with $\{a, b\}$ can be reached via a stutter step. However, no c-state can be reached from s_5 via a stutter path. The only candidate states in TS_2 that could be stutter-bisimilar to s_5 are the states v_5 and v_9 as these are equally labeled to s_5. But, since from both these states a c-state can be reached via a stutter-step, they cannot be stutter-bisimilar to state s_5. As there is no state inn TS_2 that is stutter-bisimilar to s_5, it follows $TS_1 \not\approx TS_2$.
- $TS_1 \approx TS_3$. In order to show this, consider the relation:

$$\begin{aligned} R \;=\; & \{(s_1,t_1),(s_1,t_2),(s_2,t_1),(s_2,t_2),(s_4,t_1),(s_4,t_2),(s_3,t_6),\\ & (s_5,t_4),(s_6,t_3),(s_7,t_3),(s_8,t_8),(s_{10},t_8),(s_{11},t_8),(s_9,t_7)\} \end{aligned}$$

  We claim that (the reflexive, symmetric, and transitive closure of) R is a stutter-bisimulation for $TS_1 \oplus TS_2$. Obviously, all paired states in R are equally labeled. We need to check that this relation satisfies the conditions of a stutter bisimulation. Consider the pair (s_4, t_1).

  - The transition $s_4 \to s_5$ with $(s_5, t_1) \notin R$ can be matched by state t_1 by $t_1 \to t_2 \to t_4$ since $(s_5, t_4) \in R$ and $(s_4, t_2) \in R$. Intuitively, t_1 can mimic s_4 by stuttering once, namely in state t_2.
  - $s_4 \to s_{10}$ with $(s_{10}, t_1) \notin R$ can be mimicked by $t_1 \to t_2 \to t_5$ since $(s_{10}, t_5) \in R$ and $(s_4, t_2) \in R$.
  - $t_1 \to t_3$ with $(s_4, t_3) \notin R$ is matched by $s_4 \to s_2 \to s_7$ as $(s_7, t_3) \in R$ and $(s_4, t_1) \in R$.

  For the other pairs of states, a similar reasoning applies.

  It remains to check the conditions for the initial states. For initial states $I_1 = \{s_1, s_7\}$ of TS_1 and initial states $I_3 = \{t_1, t_2, t_3\}$ of TS_3 we have: $(s_1, t_1) \in R$, $(s_1, t_2) \in R$, and $(s_7, t_3) \in R$.
- As $TS_1 \not\approx TS_2$ and $TS_1 \approx TS_3$, it follows $TS_2 \not\approx TS_3$. (as $TS_2 \approx TS_3$ and $TS_1 \approx TS_3$ would yield $TS_2 \approx TS_3$.)

Answer to Exercise 7.20
Let $\tau : LTL \to LTL\backslash_{\bigcirc}$ and TS be an arbitrary transition system. We prove that for any LTL formula ϕ such that $\mathit{Words}(\phi)$ is stutter insensitive:

$$\forall \sigma \in \mathit{Traces}(TS).\ (\sigma \models \phi \quad \text{iff} \quad \sigma \models \tau(\phi))\,.$$

The proof is by structural induction on ϕ.

Induction base. For $a \in AP$, we can simply set $\tau(a) = a$.

Induction step.

- Case $\phi = \neg\phi'$. Set $\tau(\phi) = \neg\tau(\phi')$.
- Case $\phi = \phi' \wedge \phi''$. Set $\tau(\phi) = \tau(\phi') \wedge \tau(\phi'')$.
- Case $\phi = \phi' \,\mathsf{U}\, \phi''$. Set $\tau(\phi) = \tau(\phi') \,\mathsf{U}\, \tau(\phi'')$.
- Case $\phi = \bigcirc \phi'$. The situation is more difficult, as we need to delete the occurrence of $\bigcirc$. Trace σ is called *stutter-free* if it either does not contain any stutter steps, or only stutters in the last state. Formally, σ is stutter-free if either
  - $\sigma[i] \neq \sigma[i{+}1]$ for any $i \geqslant 0$ or
  - $\exists k \geqslant 0.\ (\forall i < k.\, \sigma[i] \neq \sigma[i{+}1]) \wedge (\forall i \geqslant k.\, \sigma[i] = \sigma[i{+}1])$

  Note that each equivalence class of stutter-equivalent traces contains a unique stutter-free trace. In addition, every suffix of a stutter-free trace is stutter-free.

  For any $\mathrm{LTL}\backslash_{\bigcirc}$ formula ψ, we have:

  1. *Words*(ψ) is stutter-insensitive.
  2. For any stutter-free trace σ, $\sigma \models \psi$ iff $\sigma \in$ *Words*(ψ).

  These facts can easily be proven by structural induction on ψ. Due to these results, it suffices to show that for every LTL formula ϕ there exists a $\mathrm{LTL}\backslash_{\bigcirc}$ formula $\tau(\phi)$ that agrees with ϕ for all stutter-free traces, i.e.,

$$\forall \sigma \in \mathit{Traces}(TS) \wedge \sigma \text{ is stutter-free }.\ (\sigma \models \phi \quad \text{iff} \quad \sigma \models \tau(\phi))\,.$$

  Assume without loss of generality that $AP = \{\, a_0, ..., a_{n-1} \,\}$. Let $\mathit{val} : AP^n \to \{\,\mathrm{true}, \mathrm{false}\,\}^n$ be the set of all valuations over AP. For each $\nu \in \mathit{val}$, let β_ν be the formula

$$\alpha_0 \wedge \ldots \wedge \alpha_{n-1} \quad \text{where} \quad \alpha_j = \begin{cases} a_j & \text{if } \nu(a_j) = \mathrm{true} \\ \neg a_j & \text{if } \nu(a_j) = \mathrm{false} \end{cases}$$

  The following two observations are useful:

  1. For $\nu, \nu' \in \mathit{val}$ with $\nu \neq \nu'$:

$$\sigma \models \beta_\nu \wedge \bigcirc\, \beta_{\nu'} \quad \text{iff} \quad \sigma \models \beta_\nu \,\mathsf{U}\, \beta_{\nu'} \qquad \text{for all stutter-free traces } \sigma \text{ over } AP.$$

     This follows directly from the fact that if $\nu \neq \nu'$, then $\beta_\nu \neq \beta_{\nu'}$, and thus $\beta_\nu \wedge \bigcirc\, \beta_{\nu'}$ and $\beta_\nu \,\mathsf{U}\, \beta_{\nu'}$ coincide for any stutter-free trace.
  2. For stutter-free trace σ, we have $\sigma \models \bigcirc \phi'$ iff either
     - $\sigma[0] = \sigma[1] = ...$ and $\sigma[0] \models \phi'$, or
     - $\sigma[0] \neq \sigma[1]$ and $\underline{\sigma[1]\sigma[2]\ldots}_{\text{trace } \sigma \text{ with } \sigma[0] \text{ stripped off}} \models \phi'$.

  These properties suggest to define:

$$\tau(\bigcirc \phi') = \bigvee_{\nu \in \mathit{val}} \left(\Big(\Box\, \beta_\nu \wedge \tau(\phi') \Big) \vee \bigvee_{\nu \neq \nu'} \Big(\beta_\nu \,\mathsf{U}\, \big(\beta_{\nu'} \wedge \tau(\phi') \big) \Big) \right)$$

It follows directly that $\tau(\bigcirc\,\phi')$ is an LTL$\backslash_{\bigcirc}$-formula and is equivalent to $\bigcirc\,\phi'$. This completes the proof.

Answer to Exercise 7.22

(b) Assume $TS_1 \approx TS_2$, and let $\mathcal{R}$ be a stutter bisimulation for (TS_1, TS_2). We prove that $\mathcal{R}$ is also an observational bismulation for (TS_1, TS_2). Evidently, the conditions (A) and (B.1) are satisfied by $\mathcal{R}$. As (B.2) is the symmetric counterpart of (B.3), it suffices to consider one of them. We therefore prove (B.2). Let $(s_1, s_2) \in \mathcal{R})$ and $s_1' \in \mathit{Post}(s_1)$. Distinguish two cases:

1. $(s_1, s_1') \in \mathcal{R}$. Let $u_0 u_1 \ldots u_n = s_2$, i.e., the path just consisting of state s_2. Then $u_n = s_2$, and as $(s_1, s_2) \in \mathcal{R}$ and $(s_1, s_1') \in \mathcal{R}$, it follows by the fact that $\mathcal{R}$ is an equivalence that $s_1', s_2) \in \mathcal{R}$. (B.2) is then trivially satisfied (take m=0).
2. $(s_1, s_1') \notin \mathcal{R}$. Since $\mathcal{R}$ is a stutter bisimulation, there exists a finite path fragment of the form $s_2 u_1 \ldots u_n s_2'$ with $n \geqslant 0$, such that: $(s_2, u_i) \in \mathcal{R}$ for all $i \in \{\,1, \ldots, n\,\}$ and $(s_1', s_2') \in \mathcal{R}$. Let m=n. As $(s_2, u_i) \in \mathcal{R}$, it follows $L_2(s_2) = L_2(u_1) = ... = L_2(u_n)$. Thus, (B.2) follows.

(c) Let $\mathcal{R}$ be the equivalence on (TS_1, TS_2) inducing the following equivalence classes:

$$C_1 = \{s_1, s_2, t_1, t_2\}\,, C_2 = \{s_3, s_4, t_3\}\,, C_3 = \{s_6, s_7, t_5\}\,, C_4 = \{s_5, t_4\}$$

It is not difficult to check that $\mathcal{R}$ is a stutter bisimulation. Thus $TS_1 \approx TS_2$. By part (b) of this exercise, it follows $TS_1 \approx_{obs} TS_2$.

Answer to Exercise 7.24
If $s_1 \approx^n_{TS} s_2$ then we put

$$\nu_i^*(s_1, s_2) \;=\; \min\big\{n \in \mathbb{N} \mid \text{there exists a normed bisimulation } (\mathcal{R}, \nu_1, \nu_2) \text{ for } TS \text{ with } (s_1, s_2) \in \mathcal{R} \text{ and } \nu_i(s_1, s_2) = n \big\}$$

for $i = 1, 2$. It is now easy to check that $(\approx^n_{TS}, \nu_1^*, \nu_2^*)$ is a normed bisimulation for TS.

Answer to Exercise 7.25
Consider the transition systems TS_1 (left) and TS_2 (right) shown in Figure B.1. We first show that $TS_1 \not\approx^n TS_2$. This goes by contraposition. Assume there is a normed simulation $(\mathcal{R}, \nu_1, \nu_2)$ for (TS_1, TS_2). Then, $(s_1, t_1) \in \mathcal{R}$. The only possibility for t_1 to simulate the transition $s_1 \to s_3$ (in TS_1) is to perform the stutter step $t_1 \to t_2$ —possibly after finitely many times traversing the cycle $t_1\, t_2\, t_1$— followed by the transition $t_2 \to t_4$. Thus, $(s_1, t_2) \in \mathcal{R}$ and

$$\nu_2(s_1, t_1) > \nu_2(s_1, t_2).$$

The only possibility for t_2 to simulate the transition $s_1 \to s_2$ (in TS_1) is to move to t_1 followed by $t_1 \to t_3$. Thus,

$$\nu_2(s_1, t_2) > \nu_2(s_1, t_1).$$

Contradiction. Hence, $TS_1 \not\approx^n TS_2$.

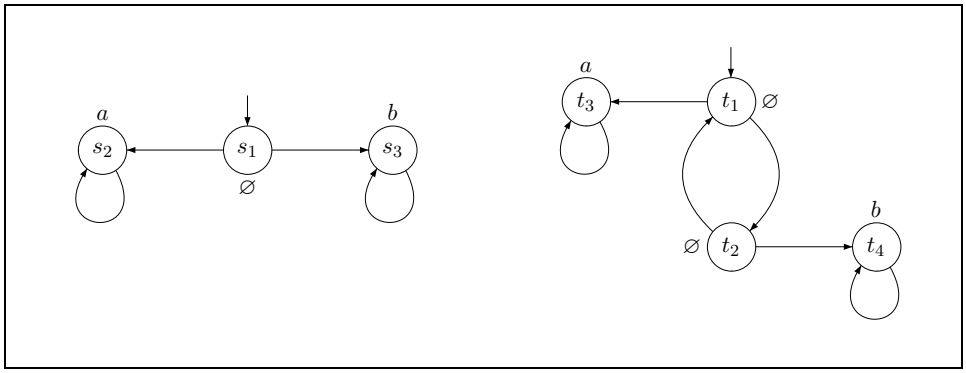


Figure B.1: Transition systems TS_1 (left) and TS_2 (right)

Let us now show that $TS_1 \approx^{div} TS_2$. The relation

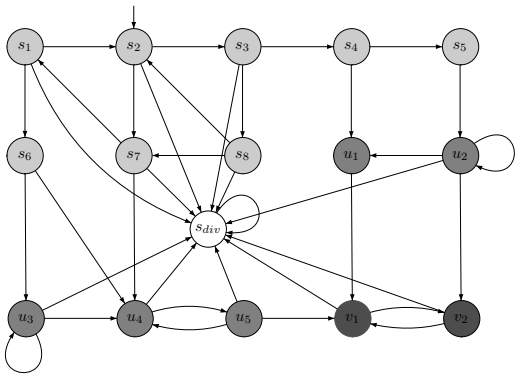
$$\mathcal{R} = \{\, (s_1, t_1), (s_1, t_2), (s_2, t_3), (s_3, t_4) \,\}$$

is a divergence-sensitive stutter bisimulation. The divergence sensitivity of $\mathcal{R}$ follows from the fact that s_1 is divergent (because of its self-loop), as well as its related states t_1 and t_2 (because of the cycle $t_1\, t_2\, t_1$), and s_2 and s_3, as well as their related states t_3 and t_4 are divergent (because of their self-loops).

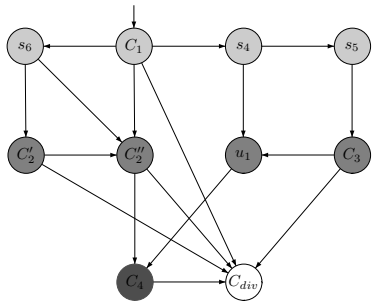
Relation $\mathcal{R}$ is a stutter bisimulation. The only interesting cases are (s_1, t_1) and (s_1, t_2). The transition $s_1 \to s_2$ can be mimicked by t_1 and t_2 by the stutter paths $t_1 \to t_3$ and $t_2 \to t_1 \to t_3$, respectively. Transition $s_1 \to s_3$ can be mimicked by t_1 and t_2 by the stutter paths $t_1 \to t_2 \to t_4$ and $t_2 \to t_4$, respectively. Vice versa, the transitions $t_1 \to t_3$ and $t_2 \to t_4$ can be simulated by the transitions $s_1 \to s_2$ and $s_1 \to s_3$, respectively.

Answer to Exercise 7.30

(a) The divergence-sensitive expansion $\overline{TS}$ is as follows:



(b) The first step is to remove stutter cycles (in fact, SCCs) from $\overline{TS}$. This yields the transition system, say $\overline{TS}'$, depicted below where $C_1 = \{ s_1, s_2, s_3, s_7, s_8 \}$, $C_2' = \{ u_3 \}$, $C_2'' = \{ u_4, u_5 \}$, $C_3 = \{ u_2 \}$, $C_4 = \{ v_1, v_2 \}$ and $C_{div} = \{ s_{div} \}$.



We now apply Algorithm 37 to determine $\overline{TS}' / \approx$.

The initial partition is:

$$\Pi_{AP} = \Big\{ \underbrace{\{s_6, C_1, s_4, s_5\}}_{B_1}, \underbrace{\{C_2', C_2'', u_1, C_3\}}_{B_2}, \underbrace{\{C_4\}}_{B_3}, \underbrace{\{C_{div}\}}_{B_4} \Big\}$$

Iteration 1: Choose $C = B_4$. Check whether C is a splitter for B_1 and B_2, respectively. Note that B_3 and B_4 are singleton sets and cannot be split any further. We have $Pre(C) = \{C_1, C_2', C_2'', C_3, C_4\}$.

- Consider B_1. $Bottom(B_1) = \{s_5, s_6\}$. Since $B_1 \neq C$, $B_1 \cap Pre(C) = \{C_1\} \neq \varnothing$ and $Bottom(B_1) \setminus Pre(C) = \{s_5, s_6\} \neq \varnothing$, C is a splitter for B_1. B_1 is splitted by C into two subblocks:
  $B_{11} = B_1 \cap Pre^*_\Pi(C) = \{C_1\}$ and $B_{12} = B_1 \setminus Pre^*_\Pi(C) = \{s_4, s_5, s_6\}$.
- Consider B_2. $Bottom(B_2) = \{C_2'', u_1\}$. Since $B_2 \neq B_4$, $B_2 \cap Pre(C) = \{C_2', C_2'', C_3\} \neq \varnothing$ and $Bottom(B_2) \setminus Pre(C) = \{u_1\} \neq \varnothing$, C is a splitter for B_2. B_2 is splitted by C into two subblocks:
  $B_{21} = B_2 \cap Pre^*_\Pi(C) = \{C_2', C_2'', C_3\}$ and $B_{22} = B_1 \setminus Pre^*_\Pi(C) = \{u_1\}$.

At the end of this iteration, we have:

$$\Pi = \Big\{ \underbrace{\{C_1\}}_{B_{11}}, \underbrace{\{s_6, s_4, s_5\}}_{B_{12}}, \underbrace{\{C_2', C_2'', C_3\}}_{B_{21}}, \underbrace{\{u_1\}}_{B_{22}}, \underbrace{\{C_4\}}_{B_3}, \underbrace{\{C_{div}\}}_{B_4} \Big\}$$

Iteration 2: Choose $C = B_{22}$. Check whether C is a splitter for B_{12} and B_{21}. We have $Pre(C) = \{s_4, C_3\}$.

- Consider B_{12}. We have $Bottom(B_{12}) = \{s_5, s_6\}$. C is a splitter for B_{12}, since $B_{12} \neq C$, $B_{12} \cap Pre(C) = \{s_4\} \neq \varnothing$, and $Bottom(B_{12}) \setminus Pre(C) = \{s_5, s_6\} \neq \varnothing$. B_{12} is split by C into two subblocks:

$$B_{121} = B_{12} \cap Pre^*_\Pi(C) = \{s_4\} \quad \text{and} \quad B_{122} = B_{12} \setminus Pre^*_\Pi(C) = \{s_5, s_6\}.$$

∗ Consider B_{21}. We have $Bottom(B_{21}) = \{C_2'', C_3\}$. C is a splitter for B_{21}, since $B_{21} \neq C$, $B_{21} \cap Pre(C) = \{C_3\} \neq \varnothing$ and $Bottom(B_{21}) \setminus Pre(C) = \{C_2''\} \neq \varnothing$. B_{21} is split by C into two subblocks:

$$B_{211} = B_{21} \cap Pre^*_\Pi(C) = \{C_3\} \quad \text{and} \quad B_{212} = B_{21} \setminus Pre^*_\Pi(C) = \{C_2', C_2''\}.$$

At the end of this iteration we have:

$$\Pi = \Big\{ \underbrace{\{C_1\}}_{B_{11}}, \underbrace{\{s_4\}}_{B_{121}}, \underbrace{\{s_6, s_5\}}_{B_{122}}, \underbrace{\{C_3\}}_{B_{211}}, \underbrace{\{C_2', C_2''\}}_{B_{212}}, \underbrace{\{u_1\}}_{B_{22}}, \underbrace{\{C_4\}}_{B_3}, \underbrace{\{C_{div}\}}_{B_4} \Big\}$$

Iteration 3: Choose $C = B_{211}$, and check whether C is a splitter for B_{122}. We have $Pre(C) = \{s_5\}$.

∗ Consider B_{122}. We have $Bottom(B_{122}) = \{s_5, s_6\}$. C is a splitter for B_{122}, since $B_{122} \neq C$, $B_{122} \cap Pre(C) = \{s_5\} \neq \varnothing$, and $Bottom(B_{122}) \setminus Pre(C) = \{s_6\} \neq \varnothing$. B_{122} is split by C into two subblocks:

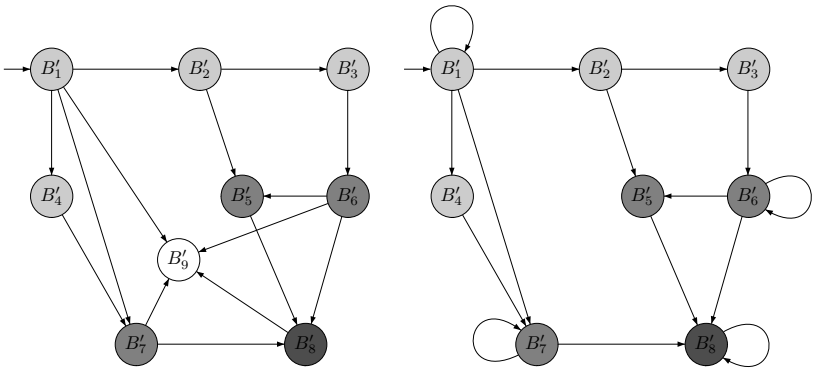
$$B_{1221} = B_{122} \cap Pre^*_\Pi(C) = \{s_5\} \quad \text{and} \quad B_{1222} = B_{122} \setminus Pre^*_\Pi(C) = \{s_6\}.$$

At the end of this iteration, we have:

$$\begin{aligned} \Pi &= \Big\{ \underbrace{\{C_1\}}_{B_{11}}, \underbrace{\{s_4\}}_{B_{121}}, \underbrace{\{s_5\}}_{B_{1221}}, \underbrace{\{s_6\}}_{B_{1222}}, \underbrace{\{C_3\}}_{B_{211}}, \underbrace{\{C_2', C_2''\}}_{B_{212}}, \underbrace{\{u_1\}}_{B_{22}}, \underbrace{\{C_4\}}_{B_3}, \underbrace{\{C_{div}\}}_{B_4} \Big\} \\ &= \Big\{ \underbrace{\{s_1, s_2, s_3, s_7, s_8\}}_{B_1'}, \underbrace{\{s_4\}}_{B_2'}, \underbrace{\{s_5\}}_{B_3'}, \underbrace{\{s_6\}}_{B_4'}, \underbrace{\{u_2\}}_{B_6'}, \underbrace{\{u_3, u_4, u_5\}}_{B_7'}, \underbrace{\{u_1\}}_{B_5'}, \underbrace{\{v_1, v_2\}}_{B_8'}, \underbrace{\{s_{div}\}}_{B_9'} \Big\} \end{aligned}$$

There are no more splitters for any blocks, thus the algorithm terminates.

(c) $\overline{TS}' / \approx$ and $TS/\approx^{div}$ are shown in the following figures (left and right), respectively.



(d) CTL$\setminus_\bigcirc$ master formulae for the equivalence classes are listed below. Here Φ_i is the master formula of equivalence class B_i'.

$$\begin{aligned} \Phi_1 &= \exists\Box\, a & \Phi_2 &= a \wedge \exists\bigcirc \forall\bigcirc c \\ \Phi_3 &= \forall\bigcirc (\exists\bigcirc (b \wedge \forall\bigcirc c)) & \Phi_4 &= \forall\bigcirc (\neg\exists\bigcirc (b \wedge \forall\bigcirc c)) \\ \Phi_5 &= b \wedge \forall\bigcirc \neg b & \Phi_6 &= \exists\bigcirc (b \wedge \forall\bigcirc c) \\ \Phi_7 &= \neg\exists\bigcirc (b \wedge \forall\bigcirc c) & \Phi_8 &= \forall\Box\, c \end{aligned}$$

Exercises of Chapter 8

Answer to Exercise 8.1
Actions α_1 and α_2 are independent if for any $s \in S$ with $\alpha_1, \alpha_2 \in Act(s)$:

$$\alpha_1 \in Act(\alpha_2(s)) \qquad \text{and} \qquad \alpha_2 \in Act(\alpha_1(s)) \qquad \text{and} \qquad \alpha_1(\alpha_2(s)) = \alpha_2(\alpha_1(s)).$$

Thus, if α_1 and α_2 are never enabled together in the same state, they are independent. We consider all action-pairs and denote by $+$ their independency and by $-$ their dependency.

action pair	$\alpha\beta$	$\alpha\gamma$	$\alpha\delta$	$\alpha\tau$	$\beta\gamma$	$\beta\delta$	$\beta\tau$	$\gamma\delta$	$\gamma\tau$	$\delta\tau$
independent	$+$	$+$	$+$	$+$	$+$	$-$	$-$	$+$	$+$	$+$

Let us justify some of these cases. Consider (α, β). The only state in which these actions are both enabled is s. As

$$\alpha \in Act(\underbrace{\beta(s)}_{t}) \qquad \text{and} \qquad \beta \in Act(\underbrace{\alpha(s)}_{u}) \qquad \text{and} \qquad \underbrace{\alpha(\beta(s))}_{v} = \underbrace{\beta(\alpha(s))}_{v},$$

it follows that α and β are independent.

Consider (α, γ). The only state in which these actions are both enabled is t. As

$$\alpha \in Act(\underbrace{\gamma(t)}_{s}) \qquad \text{and} \qquad \gamma \in Act(\underbrace{\alpha(t)}_{v}) \qquad \text{and} \qquad \underbrace{\alpha(\gamma(t))}_{u} = \underbrace{\gamma(\alpha(t))}_{u},$$

it follows that α and β are independent.

Consider (α, δ). As there is no state in which these actions are both enabled, these actions are independent. By a similar reasoning, the pairs (α, τ), (β, γ), (γ, δ), and (δ, τ) are independent.

Consider (β, δ). The only state in which both these actions are enabled is u. But as, e.g., $\beta \notin Act(\delta(u))$, these actions are dependent. By a similar reasoning it follows that β and τ are also dependent.

Answer to Exercise 8.6
Before starting with providing the answers to this exercise, we first determine the pairs of independent and dependent actions, as well as the stutter actions in *TS*.

- Action η is independent of $\{\, \alpha, \beta, \gamma, \delta \,\}$, α and γ are dependent, as well as δ and β.
- α and γ (as it only occurs as label of self-loops) are the only stutter actions. β is not a stutter action due to, e.g., $s_7 \to s_8$, neither is δ (due to $s_9 \to s_{10}$), nor is η (due to, e.g., $s_6 \to s_1$).

Let us now consider the indicated ample sets by first considering the cycle condition (A4). First, we observe that for the cycle s_6, s_7, s_8, s_9, s_6, action η is enabled in any state of the cycle but is

not included in any of the ample sets of these states. Thus, the current ample sets violate (A4). This should be fixed by adding η to the ample set of one of the states on the cycle.

Consider the constraints (A1) through (A4) for each of the ample sets (+ stands for satisfied, − for violated):

- $\mathit{ample}(s_6) = \{\, \alpha, \gamma \,\}$.
  - (A1) +: $\varnothing \neq \mathit{ample}(s_6) \subseteq \mathit{Act}(s_6) = \{\, \alpha, \gamma, \eta \,\}$.
  - (A2) +: since the set of actions depending on $\mathit{ample}(s_6)$ is $\varnothing$ (recall that an action is dependent on a set A of actions, if it is not a member of A, and depends on one of the actions in A.
  - (A3) +: as α and γ are stutter actions
  - (A4) −: on the cycle $s_6 \rightarrow s_6$, action η is in some state enabled, but does not belong to $\mathit{ample}(s_6)$. To fix this, we fully expand s_6. Note that this yields that the cycle s_6, s_7, s_8, s_9, s_6 now fulfills (A4).
- $\mathit{ample}(s_7) = \{\, \beta \,\}$.
  - (A1) +: $\varnothing \neq \mathit{ample}(s_7) \subseteq \mathit{Act}(s_7)$.
  - (A2) +: the only actions depending on $\mathit{ample}(s_7)$ is δ. It is, however, easy to check that δ can only be executed (starting from state s_7 once β has occurred.
  - (A3) −: $\mathit{ample}(s_7) \neq \mathit{Act}(s_7)$ and β is not a stutter action. In order to repair this deficiency, the only possible solution is to add η to the ample set, i.e., $\mathit{ample}(s_7) = \{\, \beta, \eta \,\}$. It follows directly that this set fulfills (A1) through (A3), as s_7 is now fully expanded.
  - (A4) +: the only cycle to which s_7 belongs is s_6, s_7, s_8, s_9, s_6 which satisfies (A4), as observed above.
- $\mathit{ample}(s_8) = \{\, \alpha \,\}$.
  - (A1) +: $\varnothing \neq \mathit{ample}(s_8) \subseteq \mathit{Act}(s_8)$.
  - (A2) −: action γ depends on $\mathit{ample}(s_8)$, and therefore should only be able to occur once α has occurred before. The execution fragment $s_8 \xrightarrow{\gamma} s_8 \xrightarrow{\gamma} s_8$, however, violates this rule. The only solution to this is to make γ not dependent on $\mathit{ample}(s_8)$. This is established by setting $\mathit{ample}(s_8) = \{\, \alpha, \gamma \,\}$. (Note that this clearly satisfies (A1), and (A2)).
  - (A3) +: as α and γ are both stutter actions.
  - (A4) +: because of the self-loop, we should add η to $\mathit{ample}(s_8)$.
- $\mathit{ample}(s_9) = \{\, \alpha, \beta, \delta \,\}$.
  - (A1) −: since $\mathit{ample}(s_9) \setminus \mathit{Act}(s_9) = \{\, \alpha \,\}$. To repair this we set $\mathit{ample}(s_9) = \{\, \beta, \delta \,\}$.
  - (A2) +: as there is no action dependent on $\{\, \beta, \delta \,\}$.
  - (A3) −: as β, δ are both not stutter actions. In order to fix this, we set $\mathit{ample}(s_9) = \mathit{Act}(s_9) = \{\, \beta, \delta, \eta \,\}$. (A1) through (A3) then trivially hold.

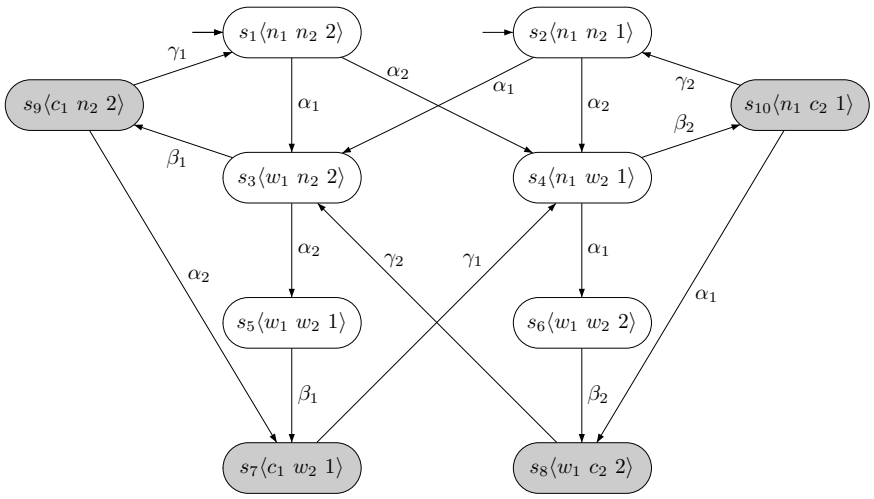
(A4) +: as state s_9 is fully expanded all cycles that contain this s_9 fulfill (A4).

- $\mathit{ample}(s_{10}) = \{\gamma, \eta\}$. As this state is fully expanded, (A1) through (A3) are trivially satisfied. (Note that $\mathit{ample}(s_{10}) = \{\gamma\}$ is not a a valid choice, as it violates (A4): on the cycle $s_{10} \to s_{10}$, action η is enabled in some state on the cycle, but is not in the ample set.)

This yields:

- $\mathit{ample}(s_6) = \{\alpha, \gamma, \eta\}$
- $\mathit{ample}(s_7) = \{\beta, \eta\}$
- $\mathit{ample}(s_8) = \{\alpha, \gamma, \eta\}$
- $\mathit{ample}(s_9) = \{\beta, \delta, \eta\}$
- $\mathit{ample}(s_{10}) = \{\gamma, \eta\}$

Answer to Exercise 8.7
Consider the transition system TS_{Pet} for the Peterson mutual exclusion algorithm as depicted below.



(a) The following pairs of actions are independent:

$$(\gamma_1, \alpha_2), \qquad (\gamma_2, \alpha_1), \qquad (\beta_2, \alpha_1), \qquad (\beta_1, \alpha_2).$$

Let us justify some of these independencies. The fact that (γ_1, α_2) are independent can be seen as follows. The only state in which both actions are enabled is s_9. The order $\gamma_1\alpha_2$ and

$\alpha_2\gamma_1$ both end up in state s_4. The independency of γ_2 and α_1 follows by symmety. The only state in which both β_2 and α_1 are enabled is s_4, and both the orders $\beta_2\alpha_1$ and $\alpha_1\beta_2$ end up in state s_8. The independency of β_1 and α_2 follows by symmetry. Note that α_1 and α_2 are not independent, as executing them in state s_1, for instance, in the order $\alpha_1\alpha_2$ and $\alpha_2\alpha_1$ yields distinct states.

(b) Compute the ample sets for each state satisfying (A1)-(A3) as follows: (Note that these ample sets might be changed while executing the algorithm, as (A4') is checked on the fly.)

$$\begin{array}{lll} \mathit{ample}(s_1) = \{\alpha_1, \alpha_2\} & \mathit{ample}(s_2) = \{\alpha_1, \alpha_2\} & \mathit{ample}(s_3) = \{\alpha_2\} \\ \mathit{ample}(s_4) = \{\alpha_1\} & \mathit{ample}(s_5) = \{\beta_1\} & \mathit{ample}(s_6) = \{\beta_2\} \\ \mathit{ample}(s_7) = \{\gamma_1\} & \mathit{ample}(s_8) = \{\gamma_2\} & \mathit{ample}(s_9) = \{\alpha_2, \gamma_1\} \\ \mathit{ample}(s_{10}) = \{\alpha_1, \gamma_2\} & & \end{array}$$

Note that states s_1, s_2, s_5, s_6, s_7 and s_8 and s_9 and s_{10} are fully expanded. Let $\Phi = \neg(\mathit{crit}_1 \wedge \mathit{crit}_2)$. The program variables are $R, U, \mathit{mark}(s_i), \mathit{ample}(s_i)$, and b. In each step of the algorithm, variables that change value are indicated by means of $\boxed{\text{box}}$.

In main procedure:

Initially, $\boxed{R := \varnothing,\ U := \varepsilon,\ b := \mathit{true}}$. Since $I \setminus R = \{s_1, s_2\}$, choose s_1 say, and invoke visit(s_1).

– In visit(s_1):
Initially, $\boxed{R := \{s_1\},\ U := \langle s_1\rangle,\ \mathit{ample}(s_1) := \{\alpha_1, \alpha_2\},\ \mathit{mark}(s_1) := \varnothing}$.

Iteration (repeat...until):

1. $s' := s_1$. $\mathit{ample}(s_1) \neq \mathit{mark}(s_1)$, so choose $\alpha_1 \in \mathit{ample}(s_1)$, and $\boxed{\mathit{mark}(s_1) := \{\alpha_1\}}$. Since $s_3 = \alpha_1(s_1)$, we have
$\boxed{R := \{s_1, s_3\},\ U := \langle s_1 s_3\rangle,\ \mathit{ample}(s_3) := \{\alpha_2\},\ \mathit{mark}(s_3) := \varnothing}$
2. $s' := s_3$. $\mathit{ample}(s_3) \neq \mathit{mark}(s_3)$, so choose $\alpha_2 \in \mathit{ample}(s_3)$, i.e., $\boxed{\mathit{mark}(s_3) := \{\alpha_2\}}$. Since $s_5 = \alpha_2(s_3)$:
$\boxed{R := \{s_1, s_3, s_5\},\ U := \langle s_1 s_3 s_5\rangle,\ \mathit{ample}(s_5) := \{\beta_1\},\ \mathit{mark}(s_5) := \varnothing}$.
3. $s' := s_5$. $\mathit{ample}(s_5) \neq \mathit{mark}(s_5)$, so choose $\beta_1 \in \mathit{ample}(s_5)$, i.e., $\boxed{\mathit{mark}(s_5) := \{\beta_1\}}$. Since $s_7 = \beta_1(s_5)$:
$\boxed{R := \{s_1, s_3, s_5, s_7\},\ U := \langle s_1 s_3 s_5 s_7\rangle,\ \mathit{ample}(s_7) := \{\gamma_1\},\ \mathit{mark}(s_7) := \varnothing}$.
4. $s' = s_7$. $\mathit{ample}(s_7) \neq \mathit{mark}(s_7)$, so choose $\gamma_1 \in \mathit{ample}(s_7)$, i.e., $\boxed{\mathit{mark}(s_7) := \{\gamma_1\}}$. Since $s_4 = \gamma_1(s_7)$:
$\boxed{R := \{s_1, s_3, s_5, s_7, s_4\},\ U := \langle s_1 s_3 s_5 s_7 s_4\rangle,\ \mathit{ample}(s_4) := \{\alpha_1\},\ \mathit{mark}(s_4) := \varnothing}$.
5. $s' = s_4$. $\mathit{ample}(s_4) \neq \mathit{mark}(s_4)$, so choose $\alpha_1 \in \mathit{ample}(s_4)$, i.e., $\boxed{\mathit{mark}(s_4) := \{\alpha_1\}}$. Since $s_6 = \alpha_1(s_4)$:
$\boxed{R := \{s_1, s_3, s_5, s_7, s_4, s_6\},\ U := \langle s_1 s_3 s_5 s_7 s_4 s_6\rangle,\ \mathit{ample}(s_6) := \{\beta_2\},\ \mathit{mark}(s_6) := \varnothing}$.

6. $s' = s_6$. $ample(s_6) \neq mark(s_6)$, so choose $\beta_2 \in ample(s_6)$, i.e., $\boxed{mark(s_6) := \{\beta_2\}}$. Since $s_8 = \beta_2(s_6)$:

   $\boxed{R := \{s_1, s_3, s_5, s_7, s_4, s_6, s_8\},\ U := \langle s_1 s_3 s_5 s_7 s_4 s_6 s_8 \rangle,\ ample(s_8) := \{\gamma_2\},\ mark(s_8) := \varnothing}$.
7. $s' := s_8$. $ample(s_8) \neq mark(s_8)$, so choose $\gamma_2 \in ample(s_8)$, i.e., $\boxed{mark(s_8) := \{\gamma_2\}}$. Since $s_3 = \gamma_2(s_8)$ and $s_3 \in R$, we have: $\boxed{ample(s_8) := Act(s_8) = \{\gamma_2\}}$.
8. $s' := s_8$. Since $mark(s_8) = ample(s_8)$, s_8 is fully explored, and $s_8 \models \Phi$. We therefore pop s_8 from the stack, yielding $\boxed{U := \langle s_1 s_3 s_5 s_7 s_4 s_6 \rangle,\ b := true}$.
9. $s' := s_6$. Since $mark(s_6) = ample(s_6)$, s_6 is fully explored, and $s_6 \models \Phi$. Thus, $\boxed{U := \langle s_1 s_3 s_5 s_7 s_4 \rangle,\ b := true}$.
10. $s' := s_4$. Since $mark(s_4) = ample(s_4)$, s_4 is fully explored, and $s_4 \models \Phi$. Thus, $\boxed{U := \langle s_1 s_3 s_5 s_7 \rangle,\ b := true}$.
11. $s' := s_7$. Since $mark(s_7) = ample(s_7)$, s_7 is fully explored, and $s_7 \models \Phi$. Thus, $\boxed{U := \langle s_1 s_3 s_5 \rangle,\ b := true}$.
12. $s' := s_5$. Since $mark(s_5) = ample(s_5)$, s_5 is fully explored, and $s_5 \models \Phi$. Thus, $\boxed{U := \langle s_1 s_3 \rangle,\ b := true}$.
13. $s' := s_3$. Since $mark(s_3) = ample(s_3)$, s_3 is fully explored, and $s_3 \models \Phi$. Thus, $\boxed{U := \langle s_1 \rangle,\ b := true}$.
14. $s' := s_1$. C $ample(s_1) \neq mark(s_1)$, so choose $\alpha_2 \in ample(s_1)$, i.e., $\boxed{mark(s_1) := \{\alpha_1, \alpha_2\}}$. Since $s_4 = \alpha_2(s_1) \in R$ and $s_4 \notin U$, do nothing.
15. $s' := s_1$. Since $mark(s_1) = ample(s_1)$, s_1 is fully explored, and $s_1 \models \Phi$. Thus, $\boxed{U := \varepsilon,\ b := true}$.

Terminate the iteration.

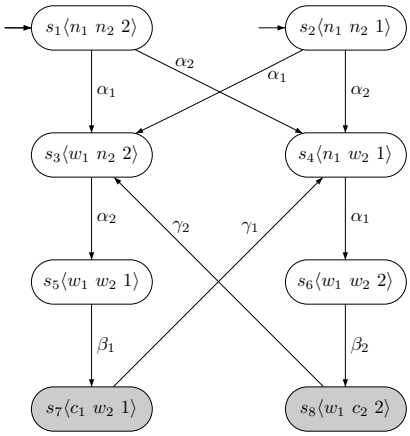
(c) In main procedure:
Since $b = true$, $I \setminus R = \{s_2\}$, choose s_2 and visit(s_2) is called.

– In visit(s_2):
Initially, $\boxed{R := \{s_1, s_3, s_5, s_7, s_4, s_6, s_8\},\ U := \langle s_2 \rangle,\ ample(s_2) := \{\alpha_1, \alpha_2\},\ mark(s_2) := \varnothing}$.

Iteration (repeat...until):

1. $s' := s_2$. Choose $\alpha_2 \in ample(s_2)$, so $\boxed{mark(s_2) := \{\alpha_2\}}$. Since $s_4 = \alpha_2(s_2) \in R$ and $s_4 \notin U$, do nothing.
2. $s' := s_2$. Choose $\alpha_1 \in ample(s_2)$, so $\boxed{mark(s_2) := \{\alpha_2, \alpha_1\}}$. Since $s_3 = \alpha_1(s_2) \in R$ and $s_3 \notin U$, do nothing.
3. $s' := s_2$. Since $mark(s_2) = ample(s_2)$, s_2 is fully explored, and $s_2 \models \Phi$. Thus, $\boxed{U := \varepsilon,\ b := true}$.

In main procedure:
Since $b = true$, the algorithm returns "yes".
The reduced transition system of Peterson's mutual exclusion algorithm becomes:



Answer to Exercise 8.9
Assume that the white states are labeled with $\varnothing$, the light gray state with $\{\, a \,\}$, and the dark gray state with $\{\, b \,\}$.

(a) Consider the transition systems in Figure 8.25. TS and $\hat{TS}$ are not stutter-trace equivalent since TS has a trace $\varnothing^\omega$, which however is not a trace in $\hat{TS}$. Condition (A3) is violated as $\mathit{ample}(s) \neq \mathit{Act}(s)$ and $\alpha \in \mathit{ample}(s)$ is not a stutter action.

(b) Consider now the transition systems in Figure 8.26. TS and $\hat{TS}$ are not stutter-trace equivalent, as $\varnothing\varnothing\{\, a \,\}^\omega \in \mathit{Traces}(TS)$, but there is no stutter-equivalent trace in $\hat{TS}$ (for the simple reason that there is no a-state). Condition (A4) is violated as in the cycle $u \to v \to u$ in $\hat{TS}$, there is a state in which α is enabled, but is not in any of the ample sets of the states on the cycle.

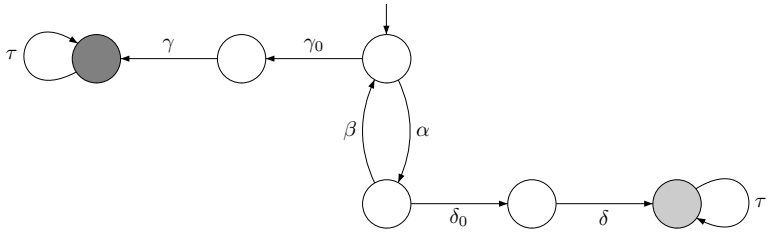
(c) Consider now the transition systems in Figure 8.27. Clearly, TS and $\hat{TS}$ are not stutter trace-equivalent, as TS has e.g., trace $(\varnothing\varnothing\{\, a \,\})^\omega$ which is not in $\hat{TS}$ (for the simple fact that there is no a-state). One of the violations is that state t must be fully expanded, as both α and γ are not stutter actions.

Answer to Exercise 8.10
(A1)-(A4) do not allow for any state reduction since the following pairs of actions are dependent:

$$(\alpha, \gamma_0),\ \ (\beta, \delta_0),\ \ (\gamma_0, \delta_0),\ \ (\gamma, \beta),\ \ (\alpha, \delta).$$

The initial state s_0 has to be fully expanded to fulfill the dependency condition (A2). The same argument applies to $\alpha(s_0)$ which has to be fully expanded too. Since β and γ are dependent, the γ_0-successor of $\alpha(s_0)$ has to be fully expanded. The same holds for the δ_0-successor of s_0, since α and δ are dependent. The nonemptiness condition (A1) then yields that all states are fully expanded.

Although, no reduction is possible according to (A1) through (A4), the following reduced transition is stutter-trace equivalent to *TS*:



Answer to Exercise 8.14

(a) First notice that all actions in *TS* are independent and that all actions, except β_2 and γ are stutter actions. Let's check the conditions (A1)-(A5) one by one:

(A1) Is clearly satisfied by all ample sets.

(A2) Holds for all ample sets, because all actions are independent of each other.

(A3) Has to be checked for:

- *ample*(s_1): Holds as β_1 is a stutter action.
- *ample*(s_2): Holds as α_1 is a stutter action.
- *ample*(s_5): Holds as α_2 is a stutter action.

(A4) There is just one cycle in $\widehat{TS}$, namely $s_1s_2s_5s_8s_9s_1$ and all actions that exist in *TS* belong to the union of ample sets, thus the condition (A4) holds.

(A5) Is clearly satisfied by all ample sets.

(b)
- Consider the following relation

$$\mathcal{R} = \{\{s_2, s_5, s_8\} \times \{s'_2, s'_5, s'_8\}, \{s_1, s_3, s_6\} \times \{s'_1\}, \{s_4, s_7, s_9\} \times \{s'_9\}\}\,.$$

- Let the function ν_1 be defined as follows:

$\nu_1(s_i, s'_j)$	s'_1	s'_2	s'_3	s'_4	s'_5	s'_6	s'_7	s'_8	s'_9
s_1	2								
s_2								2	
s_3	1								
s_4									2
s_5								1	
s_6	0								
s_7									1
s_8		2			1			0	
s_9									0

– Define the ν_2 function as follows:

$\nu_2(s_i, s'_j)$	s'_1	s'_2	s'_3	s'_4	s'_5	s'_6	s'_7	s'_8	s'_9
s_1									
s_2		2			1			0	
s_3									
s_4									2
s_5		2			1			0	
s_6									
s_7									1
s_8		2			1			0	
s_9									0

– We now check that $(\mathcal{R}, \nu_1, \nu_2)$ is a normed simulation:
  * (s_1, s'_1):
    · $s_1 \to s_2$: $s'_1 \to s'_2$, $(s_2, s'_2) \in \mathcal{R}$
    · $s_1 \to s_3$: $(s_3, s'_1) \in \mathcal{R}$, $\nu_1(s_3, s'_1) < \nu_1(s_1, s'_1)$
  * (s_2, s'_2)
    · $s_2 \to s_4$: $s'_2 \to s'_5$, $(s_2, s'_5) \in \mathcal{R}$ and $\nu_2(s_2, s'_5) < \nu_2(s_2, s'_2)$
    · $s_2 \to s_5$: $s'_2 \to s'_5$, $(s_5, s'_5) \in \mathcal{R}$
  * (s_3, s'_1)
    · $s_3 \to s_5$: $s'_1 \to s'_2$, $(s_5, s'_2) \in \mathcal{R}$
    · $s_3 \to s_6$: $(s_6, s'_1) \in \mathcal{R}$, $\nu_1(s_6, s'_1) < \nu_1(s_3, s'_1)$
  * (s_2, s'_5)
    · $s_2 \to s_4$: $s'_5 \to s'_8$, $(s_2, s'_8) \in \mathcal{R}$ and $\nu_2(s_2, s'_8) < \nu_2(s_2, s'_5)$
    · $s_2 \to s_5$: $s'_5 \to s'_8$, $(s_5, s'_8) \in \mathcal{R}$
  * (s_5, s'_5)
    · $s_5 \to s_7$: $s'_5 \to s'_8$, $(s_5, s'_8) \in \mathcal{R}$ and $\nu_2(s_5, s'_8) < \nu_2(s_5, s'_5)$
    · $s_5 \to s_8$: $s'_5 \to s'_8$, $(s_8, s'_8) \in \mathcal{R}$
  * (s_5, s'_2)
    · $s_5 \to s_7$: $s'_2 \to s'_5$, $(s_5, s'_5) \in \mathcal{R}$ and $\nu_2(s_5, s'_5) < \nu_2(s_5, s'_2)$
    · $s_5 \to s_8$: $s'_2 \to s'_5$, $(s_8, s'_5) \in \mathcal{R}$
  * (s_6, s'_1)
    · $s_6 \to s_8$: $s'_1 \to s'_2$, $(s_8, s'_2) \in \mathcal{R}$
  * (s_2, s'_8)
    · $s_2 \to s_4$: $s'_8 \to s'_9$, $(s_4, s'_9) \in \mathcal{R}$
    · $s_2 \to s_5$: $(s_5, s'_8) \in \mathcal{R}$, $\nu_1(s_5, s'_8) < \nu_1(s_2, s'_8)$
  * (s_5, s'_8)
    · $s_5 \to s_7$: $s'_8 \to s'_9$, $(s_7, s'_9) \in \mathcal{R}$
    · $s_5 \to s_8$: $(s_8, s'_8) \in \mathcal{R}$, $\nu_1(s_8, s'_8) < \nu_1(s_5, s'_8)$
  * (s_8, s'_8)
    · $s_8 \to s_9$: $s'_8 \to s'_9$, $(s_9, s'_9) \in \mathcal{R}$
  * (s_8, s'_5)

  - $s_8 \to s_9$: $s'_5 \to s'_8$, $(s_8, s'_8) \in \mathcal{R}$ and $\nu_2(s_8, s'_8) < \nu_2(s_8, s'_5)$
* (s_8, s'_2)
  - $s_8 \to s_9$: $s'_2 \to s'_5$, $(s_8, s'_5) \in \mathcal{R}$ and $\nu_2(s_8, s'_5) < \nu_2(s_8, s'_2)$
* (s_4, s'_9)
  - $s_4 \to s_7$: $(s_7, s'_9) \in \mathcal{R}$, $\nu_1(s_7, s'_9) < \nu_1(s_4, s'_9)$
* (s_7, s'_9)
  - $s_7 \to s_9$: $(s_9, s'_9) \in \mathcal{R}$, $\nu_1(s_9, s'_9) < \nu_1(s_7, s'_9)$
* (s_9, s'_9)
  - $s_9 \to s_1$: $s'_9 \to s'_1$, $(s_1, s'_1) \in \mathcal{R}$

– We then check that $(\mathcal{R}^{-1}, \nu_1^-, \nu_2^-)$ is a normed simulation:

* (s'_1, s_1):
  - $s'_1 \to s'_2$: $s_1 \to s_2$, $(s'_2, s_2) \in \mathcal{R}^{-1}$
* (s'_2, s_2)
  - $s'_2 \to s'_5$: $s_2 \to s_5$, $(s'_5, s_5) \in \mathcal{R}^{-1}$
* (s'_5, s_5)
  - $s'_5 \to s'_8$: $s_5 \to s_8$, $(s'_8, s_8) \in \mathcal{R}^{-1}$
* (s'_8, s_8)
  - $s'_8 \to s'_9$: $s_8 \to s_9$, $(s'_9, s_9) \in \mathcal{R}^{-1}$
* (s'_9, s_9)
  - $s'_9 \to s'_1$: $s_9 \to s_1$, $(s'_1, s_1) \in \mathcal{R}^{-1}$
* (s'_1, s_3)
  - $s'_1 \to s'_2$: $s_3 \to s_5$, $(s'_2, s_5) \in \mathcal{R}^{-1}$
* (s'_2, s_5)
  - $s'_2 \to s'_5$: $s_5 \to s_8$, $(s'_5, s_8) \in \mathcal{R}^{-1}$
* (s'_5, s_8)
  - $s'_5 \to s'_8$: $(s'_8, s_8) \in \mathcal{R}^{-1}$, $\nu_1^-(s'_8, s_8) < \nu_1^-(s'_5, s_8)$
* (s'_1, s_6)
  - $s'_1 \to s'_2$: $s_6 \to s_8$, $(s'_2, s_8) \in \mathcal{R}^{-1}$
* (s'_2, s_8)
  - $s'_2 \to s'_5$: $(s'_5, s_8) \in \mathcal{R}^{-1}$, $\nu_1^-(s'_5, s_8) < \nu_1^-(s'_2, s_8)$
* (s'_9, s_4)
  - $s'_9 \to s'_1$: $s_4 \to s_7$, $(s'_9, s_7) \in \mathcal{R}^{-1}$ and $\nu_2^-(s'_9, s_7) < \nu_2^-(s'_9, s_4)$
* (s'_9, s_7)
  - $s'_9 \to s'_1$: $s_7 \to s_9$, $(s'_9, s_9) \in \mathcal{R}^{-1}$ and $\nu_2^-(s'_9, s_9) < \nu_2^-(s'_9, s_7)$
* (s'_5, s_2)
  - $s'_5 \to s'_8$: $s_2 \to s_5$, $(s'_8, s_5) \in \mathcal{R}^{-1}$
* (s'_8, s_5)
  - $s'_8 \to s'_9$: $s_5 \to s_7$, $(s'_9, s_7) \in \mathcal{R}^{-1}$
* (s'_8, s_2)
  - $s'_8 \to s'_9$: $s_2 \to s_4$, $(s'_9, s_4) \in \mathcal{R}^{-1}$

Exercises of Chapter 9

Answer to Exercise 9.1

(a) The transition system $TS(\mathit{LightSwitch}) = (S, Act, \rightarrow, I, AP, L)$ where:

- $S = \{ \langle \text{off}, t_x, t_y \rangle \mid t_x, t_y \in \mathbb{R}_{\geqslant 0} \} \cup \{ \langle \text{on}, t_x, t_y \rangle \mid t_x, t_y \in \mathbb{R}_{\geqslant 0} \}$, where t_x, t_y are shorthands for the clock evaluation η with $\eta(x) = t_x$ and $\eta(y) = t_y$ respectively.
- $I = \{ \langle \text{off}, 0, 0 \rangle \}$.
- $Act = \{ \mathit{sw\_off}, \mathit{sw\_on}, d \}$ for $d \in \mathbb{R}_{\geqslant 0}$.
- $\rightarrow$ is defined by the following rules:

$$
\begin{array}{lcll}
\langle \text{off}, t_x, t_y \rangle & \xrightarrow{d} & \langle \text{off}, t_x + d, t_y + d \rangle & \text{for all } t_x, t_y \geqslant 0 \text{ and } d \geqslant 0 \\
\langle \text{off}, t_x, t_y \rangle & \xrightarrow{sw\_on} & \langle \text{on}, 0, 0 \rangle & \text{for all } t_x \geqslant 1 \\
\langle \text{on}, t_x, t_y \rangle & \xrightarrow{d} & \langle \text{on}, t_x + d, t_y + d \rangle & \text{for all } t_x, t_y \geqslant 0 \text{ and } d \geqslant 0 \text{ with } t_y + d \leqslant 3 \\
\langle \text{on}, t_x, t_y \rangle & \xrightarrow{sw\_on} & \langle \text{on}, 0, t_y \rangle & \text{for all } t_x \geqslant 2 \text{ and } t_y \leqslant 3 \\
\langle \text{on}, t_x, t_y \rangle & \xrightarrow{sw\_on} & \langle \text{off}, 0, t_y \rangle & \text{for all } t_x \geqslant 0 \text{ and } t_y = 3.
\end{array}
$$

  The set of reachable states in $TS(\mathit{LightSwitch})$ is:

$$\{\langle \text{off}, t_x, t_y \rangle \mid t_x, t_y \in \mathbb{R}_{\geqslant 0}\} \cup \{\langle \text{on}, t_x, t_y \rangle \mid t_x \in \mathbb{R}_{\geqslant 0}, 0 \leqslant t_y \leqslant 3\}.$$

- AP and L are implicitly given by the clock constraints as the conditions of the transitions.

(b) The region transition system $RTS(\mathit{LightSwitch}, \Phi)$ is as follows:

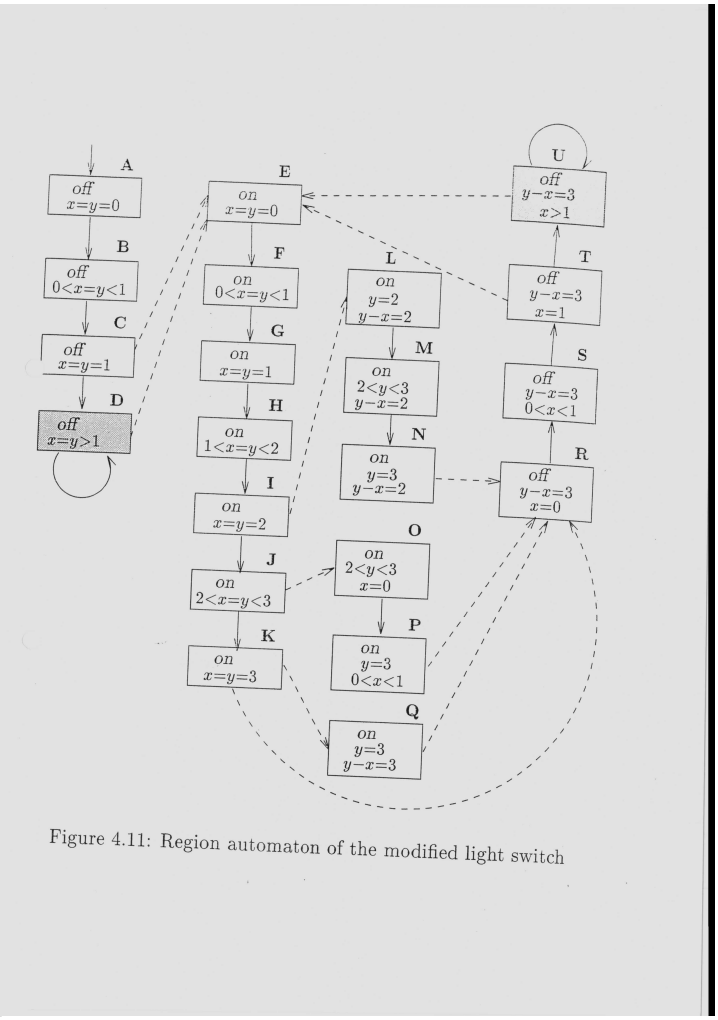


Figure 4.11: Region automaton of the modified light switch

(c)
- TA *LightSwitch* is timelock-free as for each reachable state in $TS(LightSwitch)$, there is at least one time-divergent path starting from it. It is easy to check that $RTS(LightSwitch, \Phi)$ has no such reachable states, and therefore *LightSwitch* has no timelock.
- TA *LightSwitch* is non-zeno. This follows from Lemma 9.24. Consider all control cycles in TA *LightSwitch*. On the control cycle on $\hookrightarrow$ on, clock x is reset and for constant c=2, the guard $x \geqslant 2$ on the edge on $\hookrightarrow$ on is violated. This also implies that any control cycle that contains the control cycle on $\hookrightarrow$ on, satisfies this condition. The second (and last) control cycle is off $\hookrightarrow$ on $\hookrightarrow$ off. On this cycle, clock x is reset and for constant c=1, the edge off $\hookrightarrow$ on is disabled for $x < c$. As all control cycles sastify the constraints that time elapses with at least one time unit on each control cycle, it follows from Lemma 9.24, that the timed automaton is non-zeno.

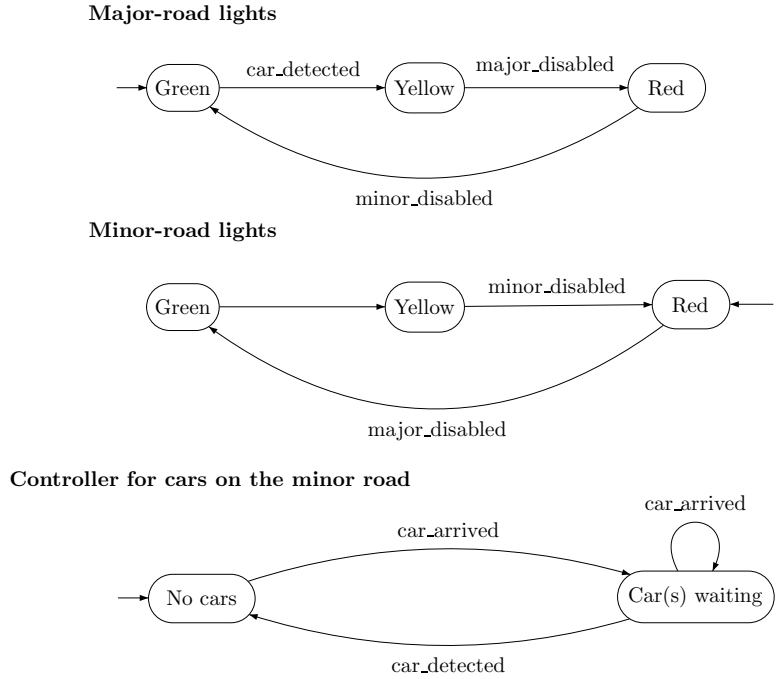
Answer to Exercise 9.2

(a) The state space is $\{\,(x,y) \mid x \geqslant 0 \wedge y \geqslant 0\,\}$, i.e., $\mathbb{R}^2_{\geqslant 0}$. The important point is that e.g., any point $y > x{+}4$ can be reached by resetting clock x. E.g., at point (4,4), resetting x yields (0,4), from there time may advance to (1,5), say, and we can reset x again, yielding (0,5). Thus, the surface $\{\,(x,y) \mid x \leqslant 4 \wedge y \geqslant 0\,\}$ is reachable. By just advancing x (i.e., no reset), also values (x,y) with $x, y \geqslant 4$ can be reached. By symmetry, the same holds for resetting of y.

(b) The state space is $\{\,(x,y) \mid 0 \leqslant y \leqslant x{+}4 \wedge 0 \leqslant x \leqslant y{+}4\,\}$. The crucial point is that clock x cannot be reset once $y > 4$, which is possible in part (a) of this exercise.

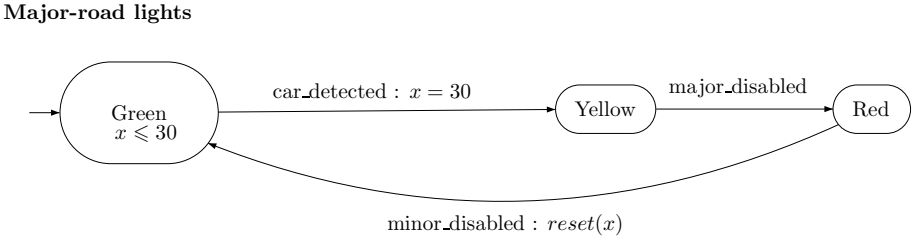
Answer to Exercise 9.3

1. The timed automata are as follows:

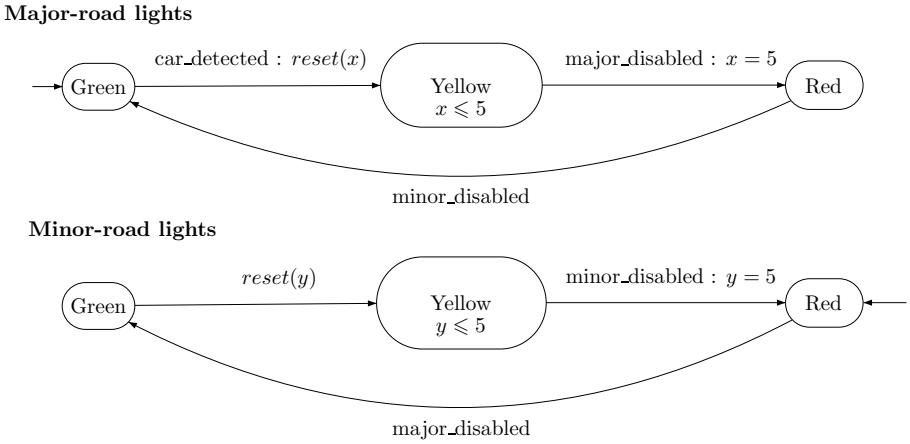


2. Required adaptations:

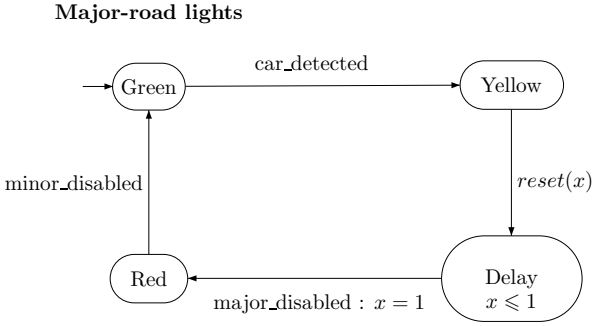
   (a) A Major light stays on green for 30 seconds (this is similar to what it would be for the traffic light on the minor road):



(b) All interim lights stay on for 5 seconds:



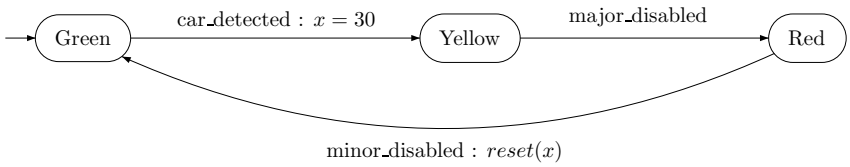
(c) The one second delay between switching the lights:



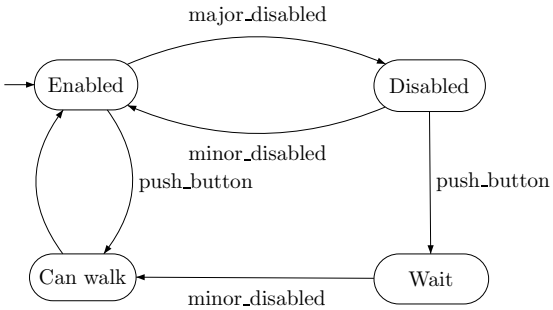
Here for the Major-road light, for switching from red to green, the solution is similar.

(d) The Major-road lights must be on green for at least 30 seconds:

Major-road lights



3. Pedestrian:



4. Does the crossing indeed only allows pedestrians to cross when the 'minor lights' are set to red? **Yes**, since we allow walking only after the red light for the minor road is ON. Which is ensured by synchronization of automata by labels 'major_disabled' and 'minor_disabled'.

Answer to Exercise 9.4

(a) $\forall\Diamond^{\leqslant 4} c$.
We have $TA_a \models \forall\Diamond^{\leqslant 4} c$ (due to the location invariants and the fact that y is reset on reaching the second location) whereas this formula is violated by all other timed automata, as these automata all allow at least one behaviour that reaches the c-location after 4 time units.

(b) $\forall\Diamond^{\leqslant 2} b \wedge \exists\Box\, \neg c$.
The first conjunct is only satisfied by the timed automata (a), (b), and (e), and among these automata only (b) satisfies the second conjunct; (a) and (e) refute this conjunct due to the location invariant on their second location.

(c) $\exists\Box\, a \wedge \forall\Box\, (b \;\Rightarrow\; \Diamond^{\leqslant 2} c)$.
The first conjunct holds in timed automata (c) and (d), (d) however violates the second conjunct (absence of location invariant in the second location), whereas (c) satisfies it.

(d) $\exists\Box\, a \wedge \exists\Diamond\, \exists\Box\, b$.
The first conjunct is satisfied by automata (c) and (d); the second conjunct is refuted by

(c), as this automata does not allow to stay forever in the second location, whereas (d) does allow this.

(e) $\forall\Diamond^{\leqslant 5} c \wedge \exists\Diamond^{>4} c$.
The first conjunct is satisfied by automata (a) and (e); the second conjunct is refuted by (a) and satisfied by (e).

(f) $\forall\Diamond^{\leqslant 6} c \wedge \exists\Diamond^{>5} c$.
The first conjunct is satisfied by timed automata (a), (e), and (f); the second conjunct is refuted by (a) and (e), but satisfied by (f).

Answer to Exercise 9.5

(a) The semantics of *TA* is given by the transition system $TS(TA) = (S, Act, \rightarrow, I, AP, L)$ where:

- $S = \{\langle l_0, t_x\rangle \mid t_x \in \mathbb{R}_{\geqslant 0}\} \cup \{\langle l_1, t_x\rangle \mid t_x \in \mathbb{R}_{\geqslant 0}\} \cup \{\langle l_2, t_x\rangle \mid t_x \in \mathbb{R}_{\geqslant 0}\}$, where t_x is the shorthand for the clock evaluation η with $\eta(x) = t_x$.
- $I = \{\langle l_0, 0\rangle\}$.
- $Act = \{\alpha, \beta, \gamma, \delta, d\}$ for $d \in \mathbb{R}_{\geqslant 0}$. We assume that the edges in *TA* are labeled with α, β, and so on.
- The transition relation $\rightarrow$ is defined as:

$$
\begin{array}{lcll}
\langle l_0, t_x\rangle & \xrightarrow{d} & \langle l_0, t_x{+}d\rangle & \text{for all } t_x \geqslant 0 \text{ and } d \geqslant 0 \text{ with } t_x + d \leqslant 2 \\
\langle l_0, t_x\rangle & \xrightarrow{\alpha} & \langle l_0, 0\rangle & \text{for all } 1 < t_x \leqslant 2 \\
\langle l_0, t_x\rangle & \xrightarrow{\beta} & \langle l_1, 0\rangle & \text{for } t_x = 2 \\
\langle l_1, t_x\rangle & \xrightarrow{d} & \langle l_1, t_x + d\rangle & \text{for all } t_x \geqslant 0 \text{ and } d \geqslant 0 \text{ with } t_x + d \leqslant 1 \\
\langle l_1, t_x\rangle & \xrightarrow{\gamma} & \langle l_2, 0\rangle & \text{for } t_x = 1 \\
\langle l_2, t_x\rangle & \xrightarrow{d} & \langle l_2, t_x + d\rangle & \text{for all } t_x \geqslant 0 \text{ and } d \geqslant 0 \text{ with } t_x + d \leqslant 2 \\
\langle l_2, t_x\rangle & \xrightarrow{\delta} & \langle l_0, t_x\rangle & \text{for } 1 \leqslant t_x < 2
\end{array}
$$

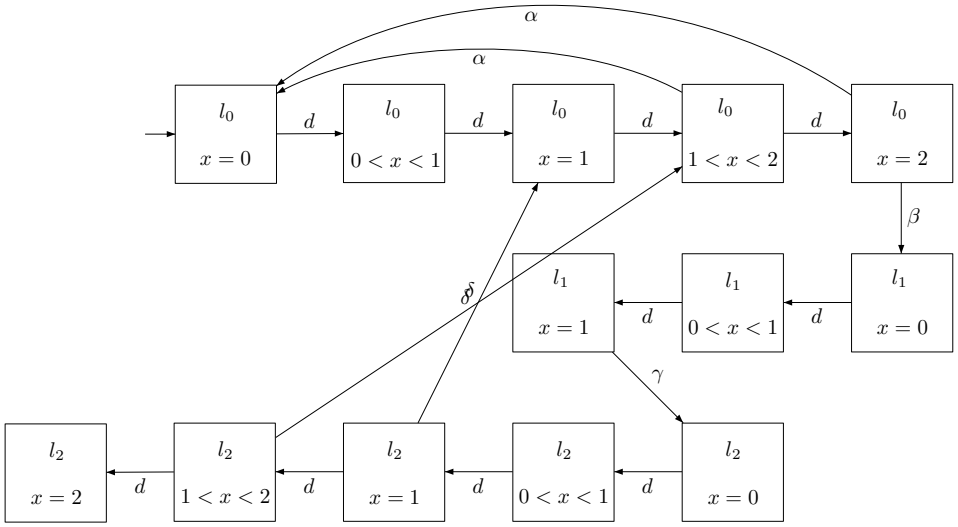
  The set of reachable states in $TS(TA)$ is:

$$\{\langle l_0, t_x\rangle \mid 0 \leqslant t_x \leqslant 2\} \cup \{\langle l_1, t_x\rangle \mid 0 \leqslant t_x \leqslant 1\} \cup \{\langle l_2, t_x\rangle \mid 0 \leqslant t_x \leqslant 2\}.$$

- AP and L are implicit in the clock constraints as the conditions of the transitions.

(b) $Sat(\exists\Diamond^{\leqslant 4} a) = \{\langle l_0, t_x\rangle \mid 0 \leqslant t_x \leqslant 2\} \cup \{\langle l_1, t_x\rangle \mid 0 \leqslant t_x \leqslant 1\} \cup \{\langle l_2, t_x\rangle \mid 0 \leqslant t_x \leqslant 2\}$.

(c) The region automaton $RTS(TA, \text{true})$ is as follows:



Note that the region transition system contains a state, $\langle l_2, x{=}2\rangle$, which has no outgoing transition. There is no time-divergent path starting from this state, and thus the timed automaton *TA* contains a timelock.

Exercises of Chapter 10

Answer to Exercise 10.1

(a) First observe that $Cyl(s_0 s_1 s_6)$ is included in $Cyl(s_0 s_1)$, and that $Cyl(s_0 s_1)$, $Cyl(s_0 s_5 s_6)$,and $Cyl(s_0 s_5 s_4 s_3)$ do not intersect. By definition

$$Pr\,(Cyl(s_0 \ldots s_n)) = i_{init}(s_0) \cdot \prod_{i=0}^{n-1} \mathbf{P}\,(s_i, s_{i+1})\,.$$

Thus

$$\begin{aligned} Pr\,(Cyl(s_0 s_1)) &= 1 \cdot \tfrac{1}{3} = \tfrac{1}{3} \\ Pr\,(Cyl(s_0 s_5 s_6)) &= 1 \cdot \tfrac{2}{3} \cdot \tfrac{1}{4} = \tfrac{1}{6} \\ Pr\,(Cyl(s_0 s_5 s_4 s_3)) &= 1 \cdot \tfrac{2}{3} \cdot \tfrac{1}{4} \cdot 1 = \tfrac{1}{6} \end{aligned}$$

Then $Pr\,(Cyl(s_0 s_1) \cup Cyl(s_0 s_5 s_6) \cup Cyl(s_0 s_5 s_4 s_3) \cup Cyl(s_0 s_1 s_6)) = \frac{1}{3} + \frac{1}{6} + \frac{1}{6} = \frac{2}{3}$.

(b) First note that states in the set B are reachable from any state of the given MC. Thus,

$$\tilde{S} = Pre^*(B) \setminus B = \{s_0, s_1, s_4, s_5, s_6\}.$$

Let vector $\mathbf{x} = (\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6})^{\mathbf{T}}$, and

$$\mathbf{A} = \begin{pmatrix} 0 & \frac{1}{3} & 0 & \frac{2}{3} & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 0 & 0 \\ \frac{2}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{2} \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \frac{1}{2} \end{pmatrix}$$

The probability $Pr(s_0 \models \Diamond B)$ is given by x_0 which is the component of the solution of:

$$\mathbf{x} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$$

Solving this system of linear equations yields $\mathbf{x} = (1, 1, 1, 1, 1)^T$. Thus $Pr(s_0 \models \Diamond B) = x_0 = 1$. Intuitively, this follows from the fact that eventually the MC will end up in one of its BSCCs, and both contain a B-state.

(c) (I) Let $S_{=0} = \{\, s_5 \,\}$ as $s_5 \notin B \cup C$, $S_{=1} = \{\, s_2, s_3, s_4 \,\}$, and $S_? = \{\, s_0, s_1, s_6 \,\}$. Note that s_4 is incorporated into $S_{=1}$, as $s_4 \in C$ and reaches the B-state s_3 with probability one. Let vector $\mathbf{x} = (x_0, x_1, x_6)^T$, and:

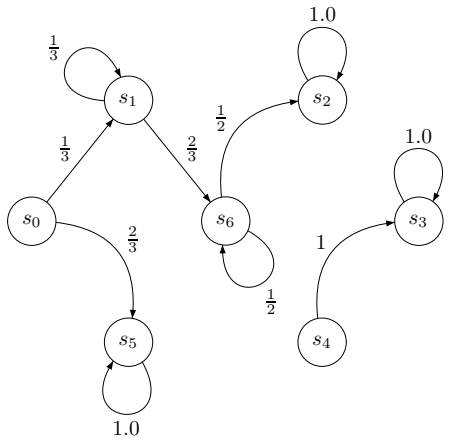
$$\mathbf{A} = \begin{pmatrix} 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & \frac{1}{2} \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ \frac{1}{2} \end{pmatrix}.$$

We are interested in $\mathbf{x}^{(5)}$ where $\Upsilon(\mathbf{x}) = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$, with $\mathbf{x}^{(i+1)} = \Upsilon(\mathbf{x}^{(i)})$, and $\mathbf{x}^{(0)} = \mathbf{0}$. By successive matrix-vector multiplication, we get

$$\mathbf{x}^1 = \begin{pmatrix} 0 \\ 0 \\ \frac{1}{2} \end{pmatrix},\ \mathbf{x}^2 = \begin{pmatrix} 0 \\ \frac{1}{3} \\ \frac{3}{4} \end{pmatrix},\ \mathbf{x}^3 = \begin{pmatrix} \frac{1}{9} \\ \frac{11}{18} \\ \frac{7}{8} \end{pmatrix},\ \mathbf{x}^4 = \begin{pmatrix} \frac{11}{54} \\ \frac{85}{108} \\ \frac{15}{16} \end{pmatrix},\ \mathbf{x}^5 = \begin{pmatrix} \frac{85}{324} \\ \frac{575}{648} \\ \frac{31}{32} \end{pmatrix}$$

We thus obtain $Pr(s_0 \models C\, U^{\leqslant 5}\, B) = \frac{85}{324}$.

(II) Making all states in B and in $S \setminus (C \cup B)$ absorbing yields the MC:



Its probability matrix $\mathbf{P}$ is:

$$\mathbf{P} = \begin{pmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{2}{3} & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{2}{3} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

As we are interested in $Pr(s_0 \models C \,\mathsf{U}^{\leqslant 5}\, B)$ let $\mathbf{i}_{init} = (1,0,0,0,0,0,0)$ and compute $\mathbf{x} = \mathbf{i}_{init} \cdot \mathbf{P}^5$. The resulting probability is then computed as $Pr(s_0 \models C \,\mathsf{U}^{\leqslant 5}\, B) = x_2 + x_3$, where $\mathbf{x} = (x_0, x_1, x_2, x_3, x_4, x_5, x_6)$. After some matrix-verctor multiplications we obtain $\frac{85}{324}$.

(d) We have $Pr(s_0 \models \Diamond \Box\, D) = Pr(s_0 \models V)$ where V is the union of all BSCCs T with $T \subseteq D$. As D is a BSCC, it follows that $D = T$, i.e., it suffices to compute $Pr(s_0 \models D)$. This goes as in part (b) of this exercise. Let $S_{=0} = \{ s_1, s_2, s_6 \}$, $S_{=1} = \{ s_3, s_4 \}$, and $S_?$ the remaining states, i.e., $\{ s_0, s_5 \}$.

Let $\mathbf{x} = (x_0, x_5)^T$, and

$$\mathbf{A} = \begin{pmatrix} 0 & \frac{2}{3} \\ \frac{1}{2} & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 0 \\ \frac{1}{4} \end{pmatrix}.$$

The required probability is then obtained by computing the fixed point of the series $\mathbf{x}^{(0)} = \mathbf{0}$, $\mathbf{x}(i{+}1) = \mathbf{A}{\cdot}\mathbf{x}^{(i)} + \mathbf{b}$. This yields:

$$\mathbf{x}^{(0)} = (0,0)^T \quad \mathbf{x}^{(1)} = (0, \frac{1}{4})^T \quad \mathbf{x}^{(2)} = (\frac{1}{6}, \frac{1}{4})^T \quad \mathbf{x}^{(3)} = (\frac{1}{6}, \frac{1}{3})^T \ldots$$

Using infinite sums we can also directly compute:

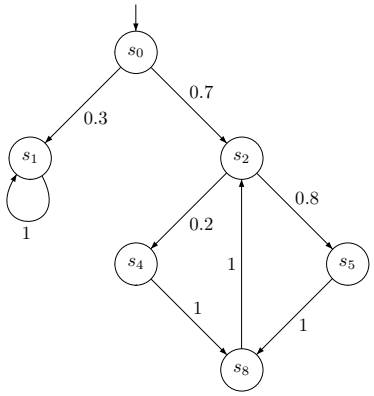
$$Pr(s_0 \models D) = \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i \cdot \left(\frac{2}{3}\right)^{i+1} \cdot \frac{1}{4}$$

which reduces to

$$\frac{1}{6}\cdot\sum_{i=0}^{\infty}\left(\frac{1}{3}\right)^{i} \;=\; \frac{1}{6}\cdot\left(\frac{1}{1-\frac{1}{3}}\right) \;=\; \frac{1}{4}.$$

Answer to Exercise 10.2
Notice that (a) and (b) are almost sure reachability properties; (c) is a repeated reachability property; and (d) and (e) are both persistence properties.

(a) Making B_1 absorbing, followed by removing all states that are unreachable from s_0 yields the MC $\mathcal{D}' = (S', \mathbf{P}', L')$:



It immediately follows from the MC that $Pr(s_0 \models \Diamond\, B_1) = 0.3 \neq 1$.
Alternatively, we do a backward search in MC $\mathcal{D}'$: $Pre^*(B_1) = \{s_0, s_1\}$, $A = S' \setminus \{s_0, s_1\} = \{s_2, s_4, s_5, s_6\}$
$Pre^*(A) = A \cup \{s_0\}$, $S' \setminus Pre^*(A) = \{s_1\}$
Since $s_0 \notin S' \setminus Pre^*(A)$, thus $Pr(s_0 \models \Diamond\, B_1) \neq 1$.

(b) Making B_2 absorbing and removing the state that are unreachable from state s_7 yields an MC with only state s_7 equipped with a self loop with probability one. Evidently, $Pr(s_7 \models \Diamond\, B_2) = 1$ since $s_7 \in B_2$.

(c) The reachable BSCCs from s_0 are:

$$T_1 = \{s_3, s_6, s_9\} \qquad \text{and} \qquad T_2 = \{s_2, s_4, s_5, s_8\}$$

We have for B_1, B_2, and B_3 the following results:

$$\begin{array}{lll} T_1 \cap B_1 = \varnothing & T_2 \cap B_1 = \varnothing & Pr(s_0 \models \Box\Diamond\, B_1) \neq 1 \\ T_1 \cap B_2 = \{s_6\} & T_2 \cap B_2 = \{s_8\} & Pr(s_0 \models \Box\Diamond\, B_2) = 1 \\ T_1 \cap B_3 = \{s_3, s_9\} & T_2 \cap B_3 = \varnothing & Pr(s_0 \models \Box\Diamond\, B_3) \neq 1 \end{array}$$

(d) Since $T_1 \not\subseteq B_2$, $T_2 \not\subseteq B_2$, it follows $Pr(s_0 \models \Box\Diamond\, B_2) \neq 1$.

(e) Since $T_1 \subseteq B_4$ and $T_2 \subseteq B_4$, it follows $Pr(s_0 \models \Box\Diamond\, B_4) = 1$.

Answer to Exercise 10.6
Note that there is a flaw in the figure in the book: r should be c. (Otherwise, the requested probability is 0.)

Under the assumption that $r = c$, we approach the problem in the following way. Clearly, for all c-states s we have $Pr(s \models b \,\mathsf{U}\, c) = 1$. This applies to $S_{=1} = \{ s_0, s_3, s_7 \}$. It is also clear that for states s_4 and s_6, the probability is zero, as they are neither b nor c-states. That is, $S_{=0} = \{ s_4, s_6 \}$. It remains to consider the remaining states $S_? = \{ s_1, s_2, s_5 \}$. Let $\mathbf{x} = (x_1, x_2, x_5)^T$ and

$$\mathbf{A} = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} \frac{1}{3} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}.$$

Solving this iteratively yields that $\mathbf{x} = (\frac{17}{19}, \frac{18}{19}, \frac{14}{19})^T$.

Answer to Exercise 10.16
Consider $\mathcal{M} \otimes \mathcal{A}$ where $\mathcal{A}$ is a DRA for the ω-regular property P. Let T be an accepting BSCC T of $\mathcal{M} \otimes \mathcal{A}$ that is reachable from some initial state $\langle s, \delta(q_0, L(s)) \rangle$. The existence of such BSCC is given since $Pr(P) > 0$. Let $\widehat{\pi}^+$ be a finite path fragment from $\langle s, \delta(q_0, L(s)) \rangle$ to T in $\mathcal{M} \otimes \mathcal{A}$ and let $\widehat{\pi}$ be the projection of $\widehat{\pi}^+$ to the states in $\mathcal{M}$. Then, almost all paths in $\mathit{Cyl}(\widehat{\pi})$ fulfill P. This follows from the fact that almost surely the runs for the paths in $\mathit{Cyl}(\widehat{\pi})$ visit exactly the (automata-)states of T infinitely often (see Theorem 10.27). Thus, almost surely the runs of the paths in $\mathit{Cyl}(\widehat{\pi})$ are accepting.

Answer to Exercise 10.29

(a) As in Algorithm 47 on page 878 use a partitioning-refinement technique where in the beginning of each iteration a set $\{T_1, \ldots, T_k\}$ of pairwise disjoint nonempty sets $T_j \subseteq S$ and a function $A : S \to 2^{Act}$ is given such that $(T_j, A|_{T_j})$ is a sub-MDP and any end component (T, A) where $T \models \mathit{sfair}$ is contained in some $(T_j, A|_{T_j})$. Then, for each $1 \leqslant j \leqslant k$:

- compute the nontrivial strongly connected components $U_1, \ldots, U_m$ of $G_{(T_j, A|_{T_j})}$,
- for each $1 \leqslant h \leqslant m$ and state $u \in U_h$, remove all actions $\alpha \in A(u)$ from $A(u)$ where $\mathit{Post}(u, \alpha) \setminus U_h \neq \varnothing$. If no action has been removed then check the fairness condition, i.e., if $\neg(U_h \models \mathit{sfair})$ then pick some strong fairness constraint $\Box\Diamond a_i \to \Box\Diamond b_i$ such that $U_h \cap \mathit{Sat}(b_i) = \varnothing$ and $U_h \cap \mathit{Sat}(a_i) \neq \varnothing$ and do the following:
  - for all states $u \in U_h \setminus \mathit{Sat}(a_i)$ remove all actions $\alpha \in A(u)$ from $A(u)$ where $\mathit{Post}(u, \alpha) \cap \mathit{Sat}(a_i) \neq \varnothing$,
  - replace U_h with $U_h \setminus \mathit{Sat}(a_i)$.

Repeat the whole procedure until there have been no changes in the last iteration.

(b) Consider the union V of the sets T of all end components (T, A) such that $T \models \mathit{fair}$ using the algorithm designed for Exercise 10.29(a). Then check whether V is reachable from all states.