Logic, Learning, and Decision

Exercises

2023

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology

Contents

Chapter 1

Introduction and Overview

This first chapter gives a flavor of the whole course. When you have solved all the exercises of the chapter, it will be easy to follow the course. If you find some exercises of the chapter difficult at the moment, come back to them after covering the corresponding further chapters in detail. And there are always solutions at the end of the chapter (but not always the best or even correct ones – stay alert).

Exercise 1.1 (Four place buffer)**.** Model a four place buffer as an automaton. Only one element at a time can be added/removed.

Exercise 1.2 (Biomass cultivation)**.** A biological material is being cultivated in a heated tank and the rate of growth is to be maximized. The growth rate can be measured, and it is assumed to depend on the temperature in the tank. When the temperature drops too low or becomes to high, the material will grow too slowly. Therefore, it is desired for the temperature to stay within a certain interval $T_{min} < T < T_{max}$.

The heating in the tank is controlled using a simple relay, that can be turned on or off (all or nothing). It is decided that a logical controller with the following functionality should be installed.

If	$T < T_{min}$	the heater should be turned on,
if	$T > T_{max}$	the heater should be turned off,
if	$T_{min} < T < T_{max}$	the heater should be used in a way that maximizes the growth rate.

Choose a control strategy using the logical signal u as a control signal. For $u = 1$ the heater is on and for $u = 0$, the heater is off. The only available in-signal is:

$$y = \begin{cases} 1, & \textit{if the growth rate is high} \\ 0, & \textit{otherwise.} \end{cases}$$

The relationship between the temperature and y is shown in Figure 1.1.

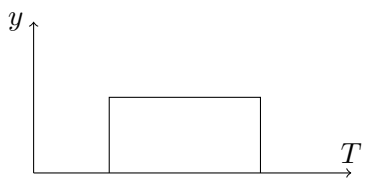


Figure 1.1: The relationship between temperature and y.

The automaton in Figure 1.2 shows the three possible states that the system can be in; too cold (left state), cozy (middle state), and too hot (right state). A peculiarity here is that using only the signal y, it is impossible to know whether it is too cold or

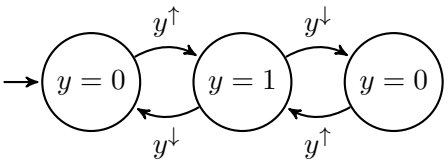


Figure 1.2: The three possible states of the system.

too hot at a particular occasion. When the signal goes low ($y \downarrow$), the system either ends up in the rightmost state or in the leftmost state. An automaton such as this one, where there exists states with several identically labeled outgoing transitions, is called *non-deterministic* (meaning it cannot be determined beforehand what will happen).

However, since we also have information about the state of the signal u, we can build a better model of the system (Figure 1.3).

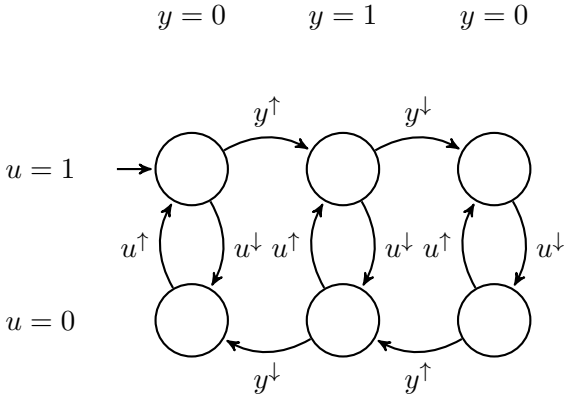


Figure 1.3: A better model of the system.

In the initial state, the tank is cold and the heater is turned on.

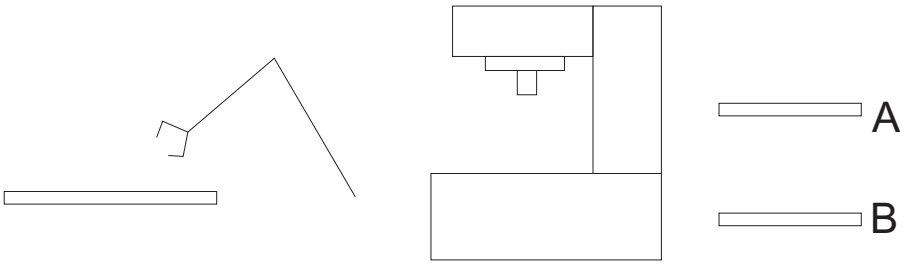
a) Formulate a desired sequence for the signals. Keep the number of on/off switchings of the heater as low as possible. Present both an ordinary automaton and an extended finite automaton[1] model of the desired sequence (y is input, u is output).

b) Give boolean state equations for ordinary and EFA models.

c) Construct a control function using SR function blocks (SR latches).

Exercise 1.3 (Robot, Machine and Specification)**.** A system consists of a robot and a machine. The robot takes parts from the input buffer and spontaneously leaves them at the machine (it is possible to control when the robot picks the part, but is not possible to control when the robot puts it). The machine loads the part that was left by the robot, processes the part and after processing unloads the part to output buffer A or B.

[1]Extended Finite Automaton (EFA) is ordinary finite state automaton extended with variables. Each transition in addition to the event can have an extra condition called *guard*. The guard is any boolean expression involving variables. The guard have to be *true* in order for transition to be allowed. Each transition can assign new values to variables, this is called *action*. Action is performed when transition is taken.



Implement a safety device (in a form of a circuit) telling when it is fine to put, load and unload the part that will guarantee the following:

- the part is loaded for processing only after it was put into the machine;
- the part should not be put into the machine before the previous part was unloaded;
- and that all output goes to buffer A.

The safety device has no control over putting the part from the robot to the machine.

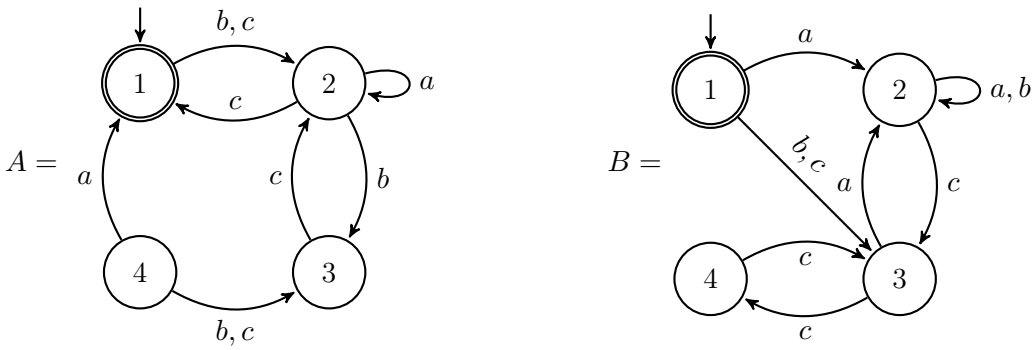
Exercise 1.4 (Synchronous composition)**.** Calculate the following synchronous compositions.

Hints: An event is enabled in a synchronized system if it is enabled in all automata. The set of events of an automaton is called *alphabet*. If an event belongs to only one automaton, it can be fired without synchronisation, and the other automata keep their states. (The other automata, that do not have the event in their alphabet, can be seen as having self-loops with this event on all states).

a) Calculate $M||R$:

b) Calculate $P||Q$:

c) Calculate $A||B$ (*note:* one transition labeled with two events, like from state 1 to state 2 of automaton A, is a shorthand for two transitions labeled by one event each):



Exercise 1.5 (Mars Pathfinder)**.** The Mars pathfinder system includes two tasks, one with high priority and one with low priority. Both tasks need exclusive access to a common resource, a shared information bus.

Automata representing the processes are shown in Fig. 1.4. Each process can be in one of three states: idle (I), requested the bus (R), and using the bus (U).

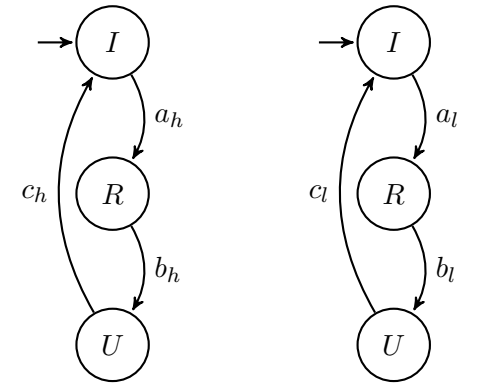
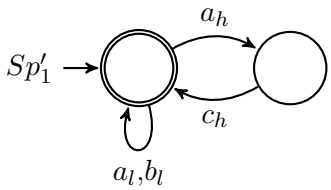


Figure 1.4: Automata for Mars Pathfinder.

The specification for the pathfinder consists of two parts shown in Fig. 1.5. One is that two processes never use the shared resource simultaneously, that is state UU is forbidden (Sp_0). The second part of the specification (Sp_1) determines the relative priority of the processes: when the high-priority process requests the information bus (event a_h), the specification states that none of the events of the low-priority task (a_l, b_l, c_l) can be executed before the high-priority task has completed its use of the information bus (event c_h).

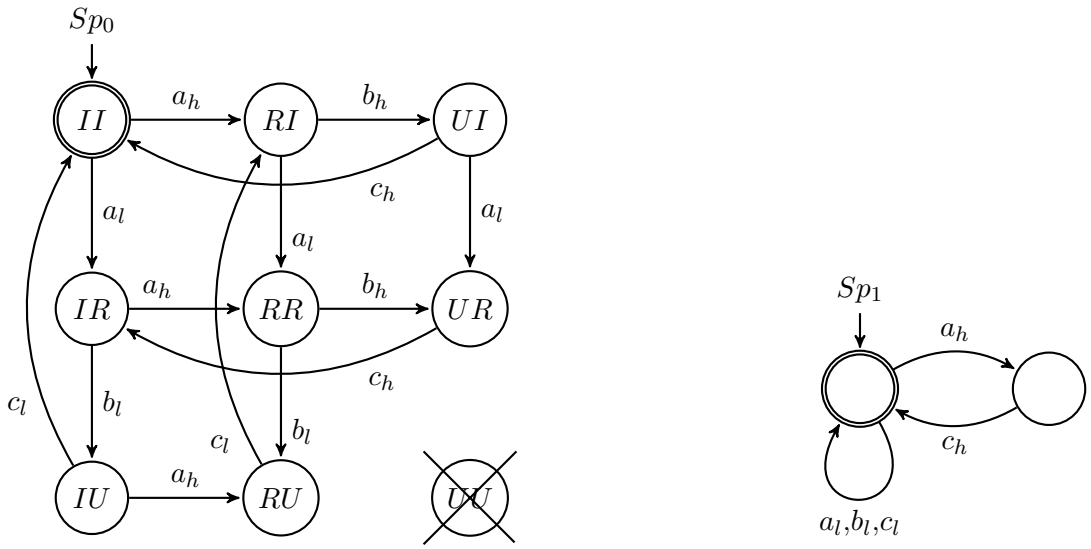
a) Compute the total specification $Sp = Sp_0||Sp_1$. What is the problem with it? Construct a controller to remove the problem.

b) Consider using specification Sp_1' instead of Sp_1:



Calculate the new total specification $Sp' = Sp_0||Sp_1'$. Compare Sp' to Sp.

c) Is there an even "better" specification that still takes care of the main issue – that



(a) Mutual exclusion specification

(b) Priority specification

Figure 1.5: Specifications for Mars Pathfinder. Events with indices l belong to low-priority task, and with h to high-priority task.

the high priority task will get access to the information bus as soon as possible after a request?

Exercise 1.6 (Mars Pathfinder – controller implementation)**.** Consider controller C in Fig. 1.6 obtained as a solution to the previous exercise. Assume that all the events a_l, b_l, c_l, a_h, b_h and c_h correspond to rising edges on the corresponding signals. That is, the event a_l occurs at the rising edge "$a_l^{\uparrow}$". The signals can be assumed to stay high for just one clock cycle.

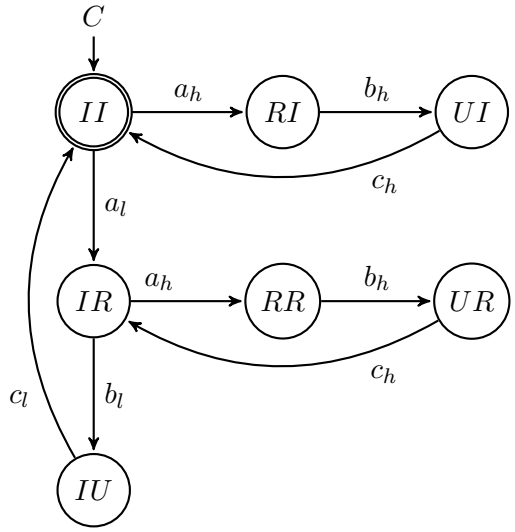


Figure 1.6: Mars Pathfinder Controller.

a) For the controller implementation, find boolean state equations for the seven states of the controller C. Only consider the controller states (call them x_{II}, x_{IR} etc. for example) and the input from the plant. That is, give $x^+ = f(x, y)$ (we have no output signals u at the moment).

b) What will happen with your boolean state representation if the low priority task makes its request when C is in state RI?

c) Suggest (informally) how controller should control the plant; what should be the output of a controller that will be at the same time an input of the Pathfinder.

Exercise 1.7 (Automatic drilling unit)**.** An automatic drilling unit uses bidirectional pneumatic pistons controlled by electromagnetically controlled bistable valves.

1) Cylinder A pushes a workpiece into the drilling unit.

2) Cylinder B locks the workpiece in position.

3) Cylinder A returns to its initial position.

4) The drilling is performed, using cylinder C.

5) When the drilling has finished, the workpiece is released and ejected.

The cylinders are fitted with position sensors y_A^0 and y_A^1, y_B^0 and y_B^1, and y_C^0 and y_C^1, respectively. Sensor y_A^0, for example, is high when the piston-rod in cylinder A is fully retracted, and sensor y_A^1 is high when the piston is fully extracted.

a) Formulate a time-distance diagram for the desired piston-rod movements.

b) Formulate the desired sequence using extended finite automata. Let u_A, u_B, and u_C be the control signals and let the sequence be started by the event *start*.

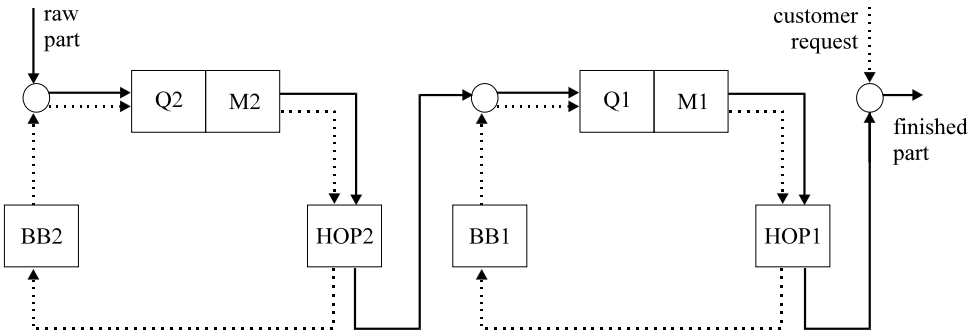
c) Give boolean state equations for the EFA model.

d) Construct a control function using SR function blocks.

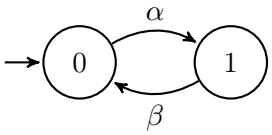
Exercise 1.8 (PIN code reader)**.** A PIN (Personal Identification Number) code is a 4-digit (usually) number that is meant to identify a person or a group of persons. For instance, entrances to building may be locked and can only be unlocked by punching in a given PIN code on a numeric terminal. In that case, the door unlocks if the specific sequence of number is entered, irrespective of the numbers punched before the sequence. This may be modelled by an automaton that reaches a marked state if the specific sequence of numbers occurs.

Draw an automaton that represents a PIN code reader that accepts the PIN code 5521.

Exercise 1.9 (Kanban)**.** This example illustrates how the synchronous composition can be used to build a complex model from simpler sub-models. Kanban denotes a special type of just-in-time production where the initiation of a product is governed by a customer request. When the order arrives a "kanban", a sort of tag, is sent backwards through the production process. In our (highly simplified) example we only have two work-cells but could have an arbitrary number. Each cell consists of an output hopper HOP_i, an input bulletin board BB_i, and a queue of work-pieces Q_i in front of each machine M_i, see the figure. Information circulates via kanbans in each cell, as indicated by the dotted lines below, while the products move towards the right according to the solid lines.



For the problem to be manageable, all units are modelled by a two-state automaton as shown below. The two events for the respective unit types are described in the table.



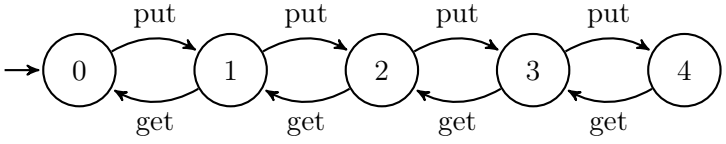
Unit	α	β
HOP_1	$e10$	$e100$
BB_1	$e12$	$e100$
Q_1	$e12$	$e11$
M_1	$e11$	$e10$
HOP_2	$e20$	$e12$
BB_2	$e22$	$e12$
Q_2	$e22$	$e21$
M_2	$e21$	$e20$

A customer request is represented by the event $e100$ being executed in the rightmost cell. The order either can be blocked if HOP_1 is empty, or be carried out directly. A kanban-card is then transferred to BB_1 and this card represents a request to the previous cell (to the left) for a part-product. If HOP_2 is empty, nothing happens. Otherwise, Q_1 picks a raw work-piece (event $e12$) and hands it to M_1 (event $e11$) and the finished product is placed in HOP_1. In the left cell, that produces part-products from raw work-pieces, works similarly. If BB_2 is not empty, a raw work-piece can be transferred into the cell to Q_2. M_2 produces and places the part-product in HOP_2

Generate a model of the kanban-system by synchronising all the units. The result will have 16 states.

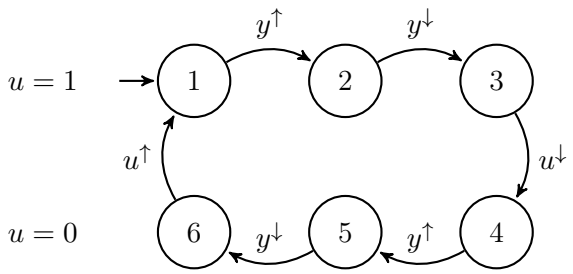
Solutions

Solution to exercise 1.1 The automaton will have five states, one for each possible value of x (where x represents the number of items stored in the buffer). If we call the event that represents the input signal of adding an item $u(t_k) = 1$ as "put", and call the event that represents the input signal of removing an item $u(t_k) = -1$ as "get", the model becomes as follows:

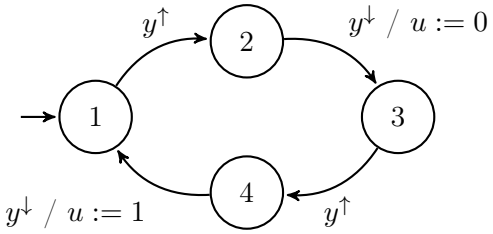


Solution to exercise 1.2

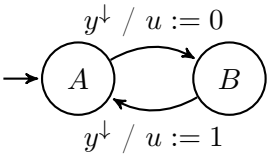
a) The process should be run in the following sequence:



This will cause the temperature to drift back and forth over the acceptable temperature interval. The **extended finite automata** representation is given below. The control signal u is changed when there is a falling edge in y.



Actually, we can ignore the rising edge of y entirely and *only* observe the falling edge of y. This gives us a more compact model.



b) Here are the boolean state equations for the ordinary finite state automata model.

	stay in state	come from other state	init
$x_1^+ =$	$(x_1 \wedge \neg y^{\uparrow})$	$\vee(x_6 \wedge u^{\uparrow})$	$\vee init$
$x_2^+ =$	$(x_2 \wedge \neg y^{\downarrow})$	$\vee(x_1 \wedge y^{\uparrow})$	
$x_3^+ =$	$(x_3 \wedge \neg u^{\downarrow})$	$\vee(x_2 \wedge y^{\downarrow})$	
$x_4^+ =$	$(x_4 \wedge \neg y^{\uparrow})$	$\vee(x_3 \wedge u^{\downarrow})$	
$x_5^+ =$	$(x_5 \wedge \neg y^{\downarrow})$	$\vee(x_4 \wedge y^{\uparrow})$	
$x_6^+ =$	$(x_6 \wedge \neg u^{\uparrow})$	$\vee(x_5 \wedge y^{\downarrow})$	
$u^+ =$	$x_1^+ \vee x_2^+ \vee x_3^+$		

Here are the boolean state equations for the four-state EFA.

	stay in state	come from other state	init
$x_1^+ =$	$(x_1 \wedge \neg y)$	$\vee(x_4 \wedge \neg y)$	$\vee init$
$x_2^+ =$	$(x_2 \wedge y)$	$\vee(x_1 \wedge y)$	
$x_3^+ =$	$(x_3 \wedge \neg y)$	$\vee(x_2 \wedge \neg y)$	
$x_4^+ =$	$(x_4 \wedge y)$	$\vee(x_3 \wedge y)$	
$u^+ =$	$(u \wedge \neg(x_2 \wedge y^{\downarrow}))$	$\vee(x_4 \wedge y^{\downarrow})$	
	or		
$u^+ =$	$(u \wedge \neg x_3)$	$\vee x_1$	

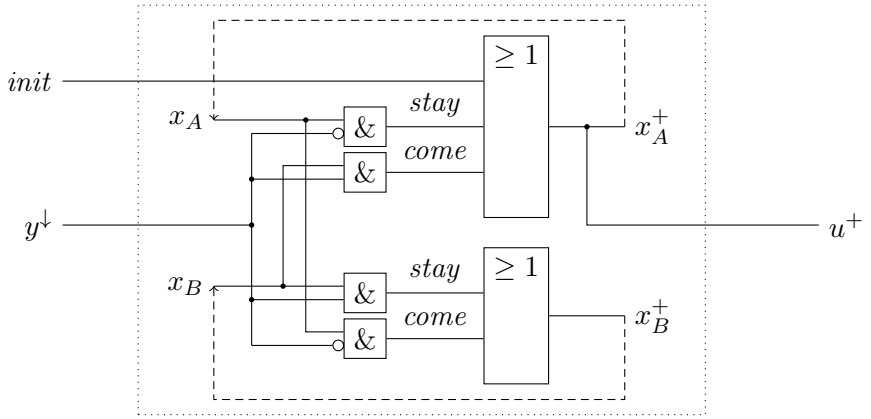
In the last line u^+ stays high if it was high already and we are not coming into state x_3, or if we are coming into state x_1. The other expression for u^+ has conditions for transitions explicitly.

We actually do not need to look at the edges in this case. Since the value of y is constant throughout the entire stay in a state, we can use that value instead of the edge for detecting when we enter the state.

This is not the case if we instead choose the two-state EFA model. Then we must rely on the edges, since y may have different values *within* each state. Here are the boolean state equations for the two-state EFA, let $y\downarrow$ represent a signal that is high for a short period when there is a falling edge in y and low otherwise.

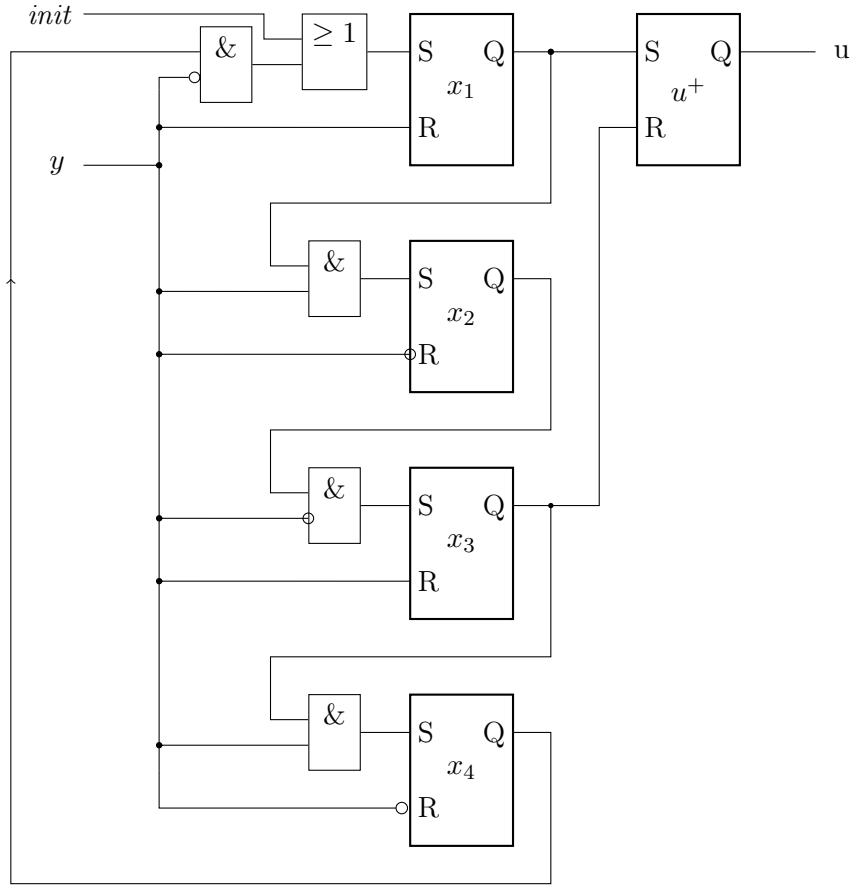
$$
\begin{aligned}
x_A^+ &= (x_A \wedge \neg y^{\downarrow}) && \vee (x_B \wedge y^{\downarrow}) && \vee init \\
x_B^+ &= (x_B \wedge \neg y^{\downarrow}) && \vee (x_A \wedge y^{\downarrow}) \\
u &= x_A
\end{aligned}
$$

This can be drawn as a circuit:



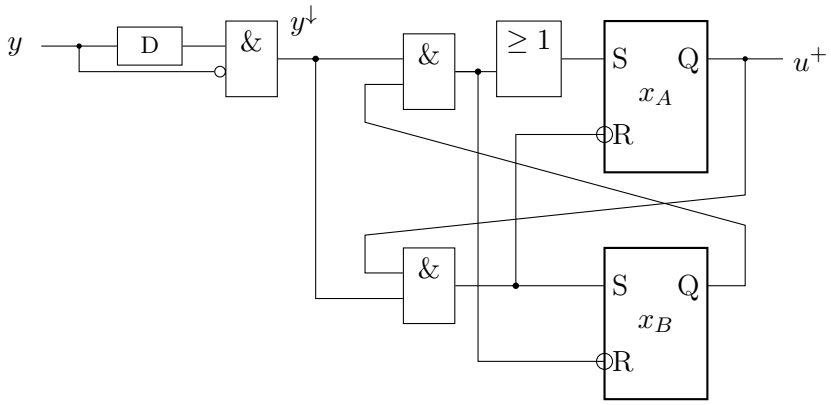
c) SR latches (flip-flops) for each state will have the same signal for the S input as the coming from other state expression in boolean state equation, and R input as a negated staying in a state expression ("reset if not staying").

First, the four state EFA variant. Again we're assuming that *init* only occurs when the system is in the right state (y is low and the tank is cold).



Below we show the function blocks for the two-state EFA.

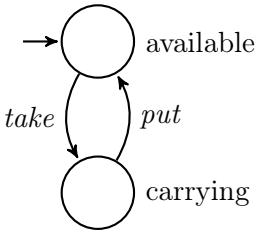
First we need to detect the negative edges of y. We can do this using a small time delay (this can be realized using a series of inverters, for example), and compare the delayed signal with the (inverse of) current signal. This will give a pulse as long the delay when a negative edge occurs. The block labeled D below is such a time delay.



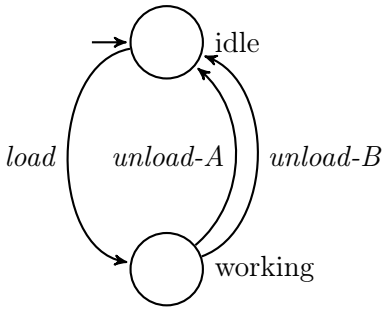
Since this is a two state automaton, what we want is just a flip-flop that toggles on negative flanks. There are much simpler ways of getting this effect, but we'll stick to simple SR blocks here.

Solution to exercise 1.3 We start by creating a formal model of the system.

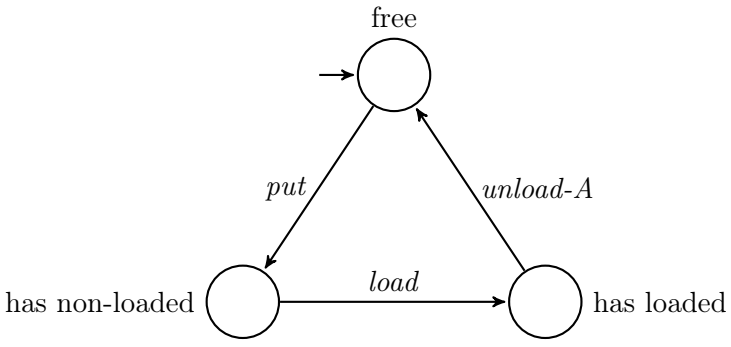
The robot can be modeled with a two-states automaton. One state will correspond to the state when the robot is *available*, and another one to the state when robot is *carrying* the part. To go from the available state to carrying, the robot has to *take* the part. To go back, robot has to *put* the part:



The machine can be modeled with two states as well: *idle* and *working*. The machine has to *load* the part (that was initially put onto machine) to go to the working state. From the working state it can unload the part to the buffer A or the buffer B to get back to the idle state.

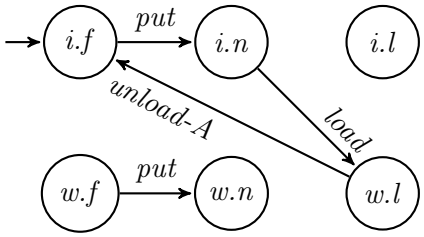


The specification has to ensure that event *load* is executed only after event *put*, event *unload-B* is forbidden, and event *put* is executed only after event *unload-A*. The automaton will have three states: before the *put*-event, when the machine is *free*, after *put*, but before *load*, when machine *has a non-loaded* part, and after *load* before *unload-A* when machine *has a loaded* part:

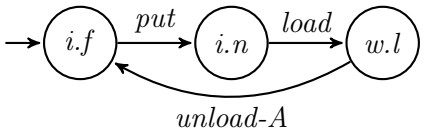


forbidden events: *unload-B*

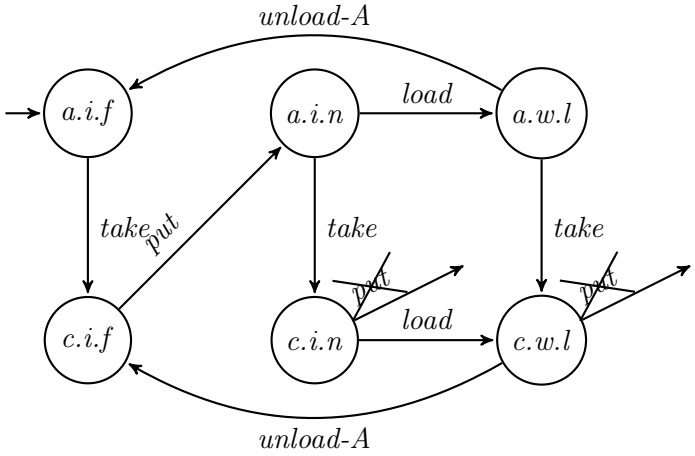
Then we combine or *synchronize* all three automata (more about synchronization in Chapter 3). We can do it in any order. We start by combining the automata of the machine and the specification (*i=idle*, *w=working*, *f=free*, *n=has non-loaded*, *l=has loaded*):



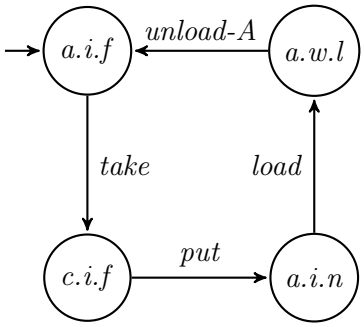
There are six possible combinations of states, but only three of them are reachable. Unreachable states can be ignored:



Then we combine this result with the robot automaton:



There are two states *c.i.n* and *c.w.l* that correspond to the state of the robot when it carries the part, and when it can put the part into machine. But, in these states *c.i.n* and *c.w.l*, putting the part is forbidden by synchronizing with the specification. If the robot puts the part (which is uncontrollable), it will break the specification. Since it is usually not possible to change the specification, we should prevent the system from entering the states that can lead to breaking the specification. So our safety device should just prevent entering the problematic states:



The safety device we created is called *supervisor*, and the process of its creation is called *synthesis*. Chapter 7 is devoted to synthesis.

Having a formal model of a supervisor, we can give boolean state equations for it or draw a boolean circuit that can be implemented in hardware (Chapter 5 has more about implementation, and Chapter 2 provides background about boolean formulas).

Inputs of the circuit will be the events that occur in the system, *take*, *put*, *load*, *unload-A*, *unload-B*, and the outputs are the signals that indicate which of the (controllable) events *take*, *load*, *unload-A*, *unload-B* are enabled (*unload-B* is always disabled). The supervisor indicates, like a traffic light, what is fine to do now, and what is not.

The boolean state equations will look as following:

	stay in state	come from other state	init
$x^+_{aif} =$	$(x_{aif} \wedge \neg take) \vee$	$(x_{awl} \wedge \textit{unload-A}) \vee$	$init$
$x^+_{cif} =$	$(x_{cif} \wedge \neg put) \vee$	$(x_{aif} \wedge take)$	
$x^+_{ain} =$	$(x_{ain} \wedge \neg load) \vee$	$(x_{cif} \wedge put)$	
$x^+_{awl} =$	$(x_{awl} \wedge \neg \textit{unload-A}) \vee$	$(x_{ain} \wedge load)$	
$enabled^+_{take} =$	x^+_{aif}		
$enabled^+_{load} =$	x^+_{ain}		
$enabled^+_{\textit{unload-A}} =$	x^+_{awl}		
$enabled^+_{\textit{unload-B}} =$	$false$		

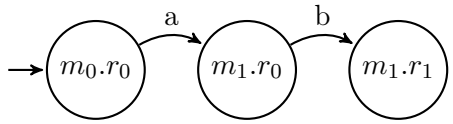
The state is active at the next step in one of three cases:

- the state is initial and now is the first cycle (signal *init* is active);
- the state is active and current event is not among the ones that lead out of the state;
- some other state is active and currently active event leads from that other state to this one.

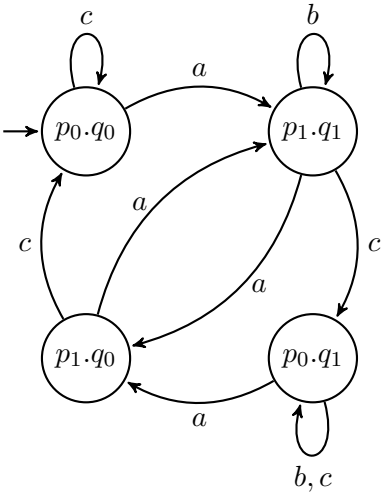
These equations can be readily converted to schematics and implemented in hardware.

Solution to exercise 1.4 The difficult part of the synchronous composition is seeing when an event is *disabled*. Remember that *all* automata (in which the event occurs) must agree on an event for it to occur in the composition.

a) $M||R =$

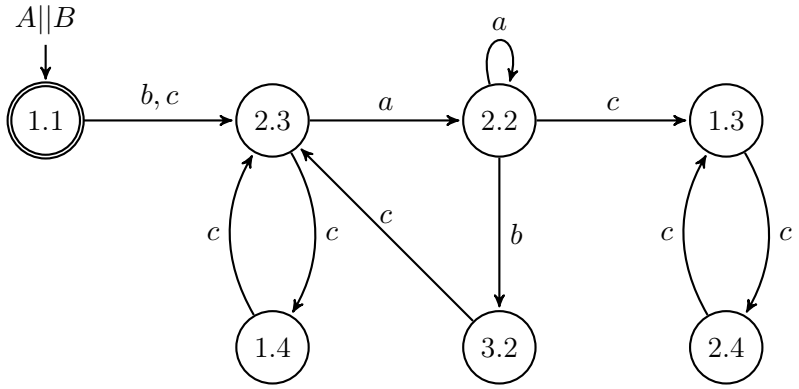


b) $P||Q =$



Note that P does not care about b, and Q does not care about c.

c)



Unreachable states, such as 4 in A, can of course be removed altogether.

Solution to exercise 1.5

a) The total specification $Sp = Sp_0||Sp_1$ is shown in Fig. 1.7. The problem is that

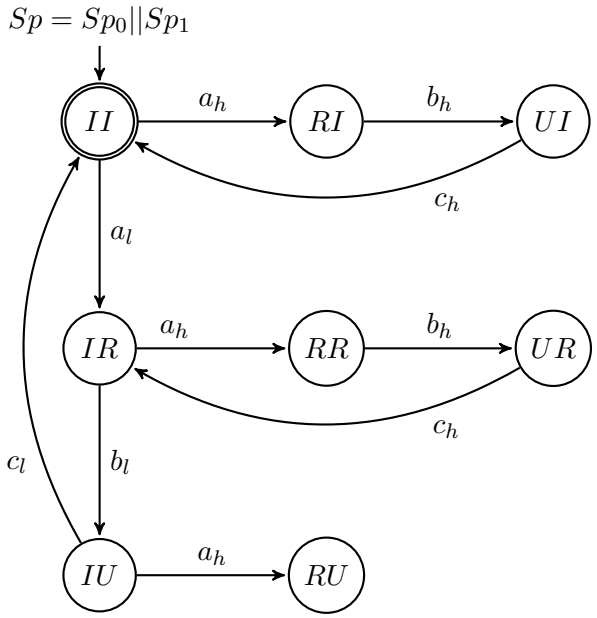


Figure 1.7: Total specification for Mars Pathfinder. State RU is a deadlock.

when the low-priority process uses the information bus (state U) and the high-priority process requests the same bus (event a_h), a deadlock occurs (state RU). The low-priority task cannot be completed, because the scheduler does not allow c_l to occur before the high-priority task has been completed. But the high-priority task cannot even use the bus because it is used exclusively by the low-priority task.

The solution is to avoid that the high-priority task requests the information bus (event a_h) until the low-priority task has completed its operation (event c_l). This is achieved by removing the deadlock state RU from the automaton Sp in Fig. 1.7, which after this modification can be used as a controller. The removal of the deadlock state is the controller synthesis step, which results in a correct control function.

b) Specification $Sp' = Sp_0||Sp_1'$ is shown on Fig. 1.8. The difference is that now the low priority task is allowed to finish even though the high priority task has made a

request. Apparently, this is a better solution than restricting the high priority task from making the request. However, this is cheating: changing the specification is not something that we can do in general. Usually we assume that the specification is correct and make the best out of the situation by changing the *plant* through control.

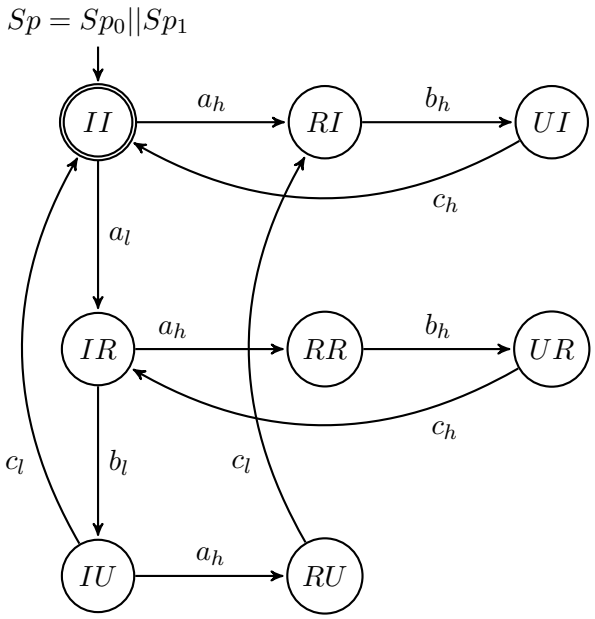
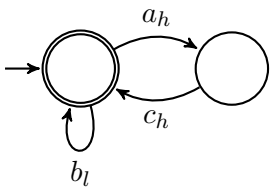


Figure 1.8: Adapted specification for Mars Pathfinder. State RU is no longer a deadlock.

c) This one is pretty good:



It allows the low priority task to make requests even when the high priority task is "active". It will not grant the requests, of course, until the high-priority task is finished.

Solution to exercise 1.6

a) For each of the seven states we assign a variable x_i. The update of the states in each scan-cycle is based on the current state and the "event signals". Either the system stays in the current state or it changes state. Supposedly, only one state can be active at a time. It is assumed that when the *init* signal starts the controller,

the system is in the initial state.

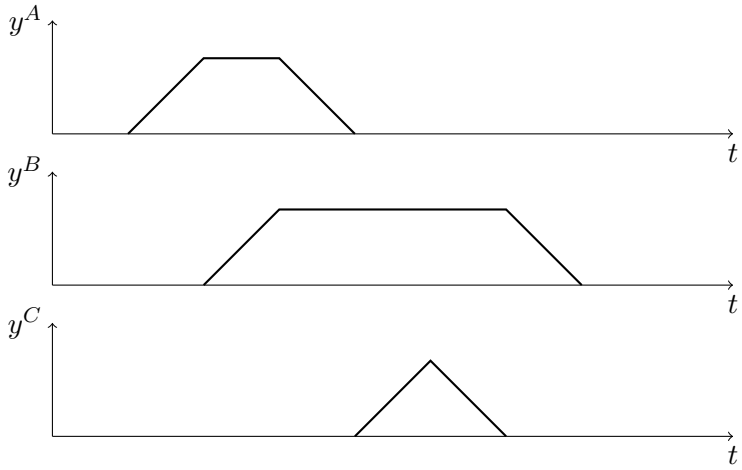
	stay in state	come from other state	init
x^+_{II}	$=(x_{II} \wedge \neg a_l \wedge \neg a_h)$	$\vee\ (x_{IU} \wedge c_l) \vee (x_{UI} \wedge c_h)$	$\vee\ init$
x^+_{IR}	$=(x_{IR} \wedge \neg a_h \wedge \neg b_l)$	$\vee\ (x_{II} \wedge a_l) \vee (x_{UR} \wedge c_h)$	
x^+_{IU}	$=(x_{IU} \wedge \neg c_l)$	$\vee\ (x_{IR} \wedge b_l)$	
x^+_{RI}	$=(x_{RI} \wedge \neg b_h)$	$\vee\ (x_{II} \wedge a_h)$	
x^+_{UI}	$=(x_{UI} \wedge \neg c_h)$	$\vee\ (x_{RI} \wedge b_h)$	
x^+_{RR}	$=(x_{RR} \wedge \neg b_h)$	$\vee\ (x_{IR} \wedge a_h)$	
x^+_{UR}	$=(x_{UR} \wedge \neg c_h)$	$\vee\ (x_{RR} \wedge b_h)$	

b) Nothing, which is bad. The controller does not detect (fails to *observe*) a state change in the plant so they become "out of sync". When the plant wants to execute its second event, the controller is still expecting the first event.

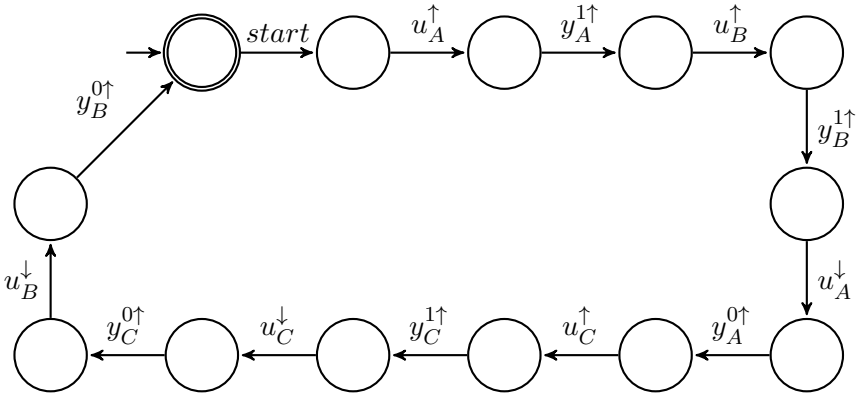
c) The controller can monitor events and prevent certain signals in the system from occurring (for example filter requests). When x_{RI} or x_{UI} is high, the controller must "stop" a_l from going high (it must deny requests from the low priority task). When x_{IU} is high, the controller must "stop" a_h.

Solution to exercise 1.7

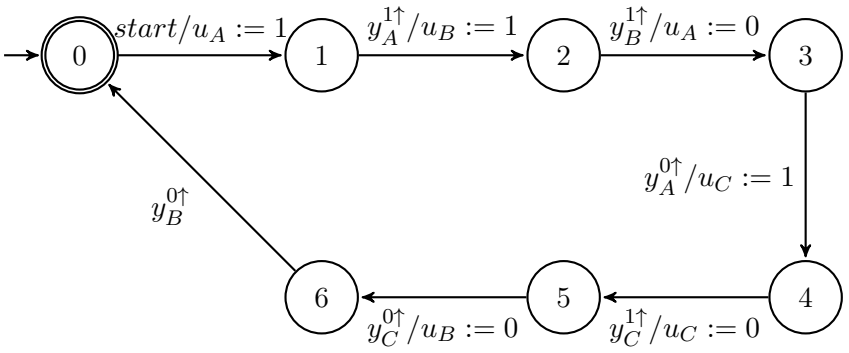
a) We assume a constant extraction and retraction speed. The time-distance diagram for all three piston rods are as shown below. When piston A is fully extracted, B starts to extract. When piston B reaches its final position, A retracts. The drilling is performed once A is fully retracted and is characterized as one continuous extraction-retraction movement of piston C. At its end B retracts.



b) The above plots correspond to the following desired sequence:



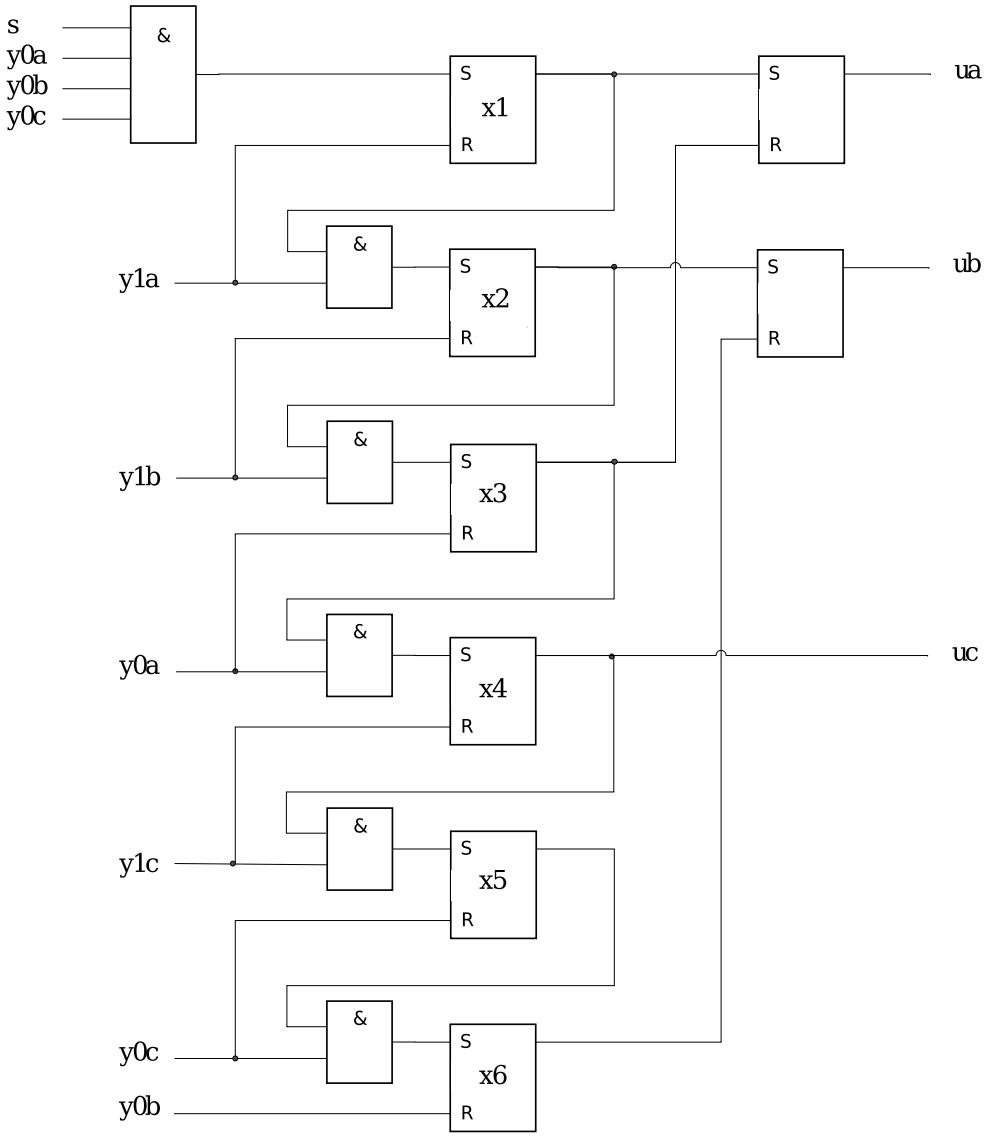
The resulting EFA model is shown below.



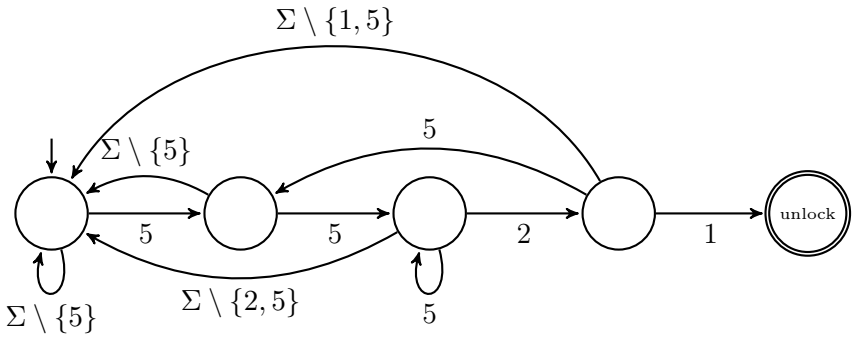
c) The initial state is uniquely defined by the input signals. Therefore we don't need to implement it as an internal state.

stay in state	come from other state	init
$x_1^+ = (x_1 \wedge \neg y_A^{1\uparrow})$	$\vee (start \wedge y_A^{0\uparrow} \wedge y_B^{0\uparrow} \wedge y_C^{0\uparrow})$	
$x_2^+ = (x_2 \wedge \neg y_B^{1\uparrow})$	$\vee (x_1 \wedge y_A^{1\uparrow})$	
$x_3^+ = (x_3 \wedge \neg y_A^{0\uparrow})$	$\vee (x_2 \wedge y_B^{1\uparrow})$	
$x_4^+ = (x_4 \wedge \neg y_C^{1\uparrow})$	$\vee (x_3 \wedge y_A^{0\uparrow})$	
$x_5^+ = (x_5 \wedge \neg y_C^{0\uparrow})$	$\vee (x_4 \wedge y_C^{1\uparrow})$	
$x_6^+ = (x_6 \wedge \neg y_B^{0\uparrow})$	$\vee (x_5 \wedge y_C^{0\uparrow})$	
$u_A^+ = x_1^+ \vee x_2^+$		
$u_B^+ = x_2^+ \vee x_3^+ \vee x_4^+ \vee x_5^+$		
$u_C^+ = x_4^+$		

d) An implementation of the control function with SR function blocks:

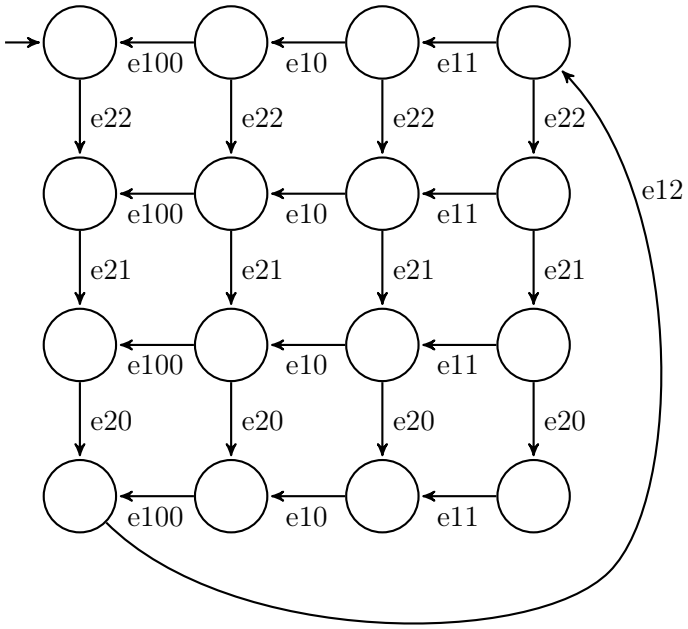


Solution to exercise 1.8



The symbol Σ represents the "alphabet", $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $\Sigma \setminus \{2, 5\}$ means "Σ minus the events 2 and 5" ($\setminus$ is the symbol for "set minus").

Solution to exercise 1.9 Synchronise the eight (relabelled) automata and you get the following:


e100
e10
e11
e22
e22
e22
e22
e12
e100
e10
e11
e21
e21
e21
e21
e100
e10
e11
e20
e20
e20
e20
e100
e10
e11

Chapter 2

Discrete Mathematics

Equivalence relations:

$$
\begin{array}{lrcl}
E_1 & \neg\neg p & \Leftrightarrow & p \\
E_2 & \neg(p \vee q) & \Leftrightarrow & \neg p \wedge \neg q \\
E_3 & \neg(p \wedge q) & \Leftrightarrow & \neg p \vee \neg q \\
E_4 & p \vee q & \Leftrightarrow & q \vee p \\
E_5 & p \wedge q & \Leftrightarrow & q \wedge p \\
E_6 & p \vee (q \vee r) & \Leftrightarrow & (p \vee q) \vee r \\
E_7 & p \wedge (q \wedge r) & \Leftrightarrow & (p \wedge q) \wedge r \\
E_8 & p \wedge (q \vee r) & \Leftrightarrow & (p \wedge q) \vee (p \wedge r) \\
E_9 & p \vee (q \wedge r) & \Leftrightarrow & (p \vee q) \wedge (p \vee r) \\
E_{10} & p \vee p & \Leftrightarrow & p \\
E_{11} & p \wedge p & \Leftrightarrow & p \\
E_{12} & p \vee \mathbb{F} & \Leftrightarrow & p \\
E_{13} & p \wedge \mathbb{T} & \Leftrightarrow & p \\
E_{14} & p \vee \mathbb{T} & \Leftrightarrow & \mathbb{T} \\
E_{15} & p \wedge \mathbb{F} & \Leftrightarrow & \mathbb{F} \\
E_{16} & p \vee \neg p & \Leftrightarrow & \mathbb{T} \\
E_{17} & p \wedge \neg p & \Leftrightarrow & \mathbb{F} \\
E_{18} & p \vee (p \wedge q) & \Leftrightarrow & p \\
E_{19} & p \wedge (p \vee q) & \Leftrightarrow & p \\
E_{20} & p \to q & \Leftrightarrow & \neg p \vee q \\
E_{21} & \neg(p \to q) & \Leftrightarrow & p \wedge \neg q \\
E_{22} & p \to q & \Leftrightarrow & \neg q \to \neg p \\
E_{23} & p \to (q \to r) & \Leftrightarrow & (p \wedge q) \to r \\
E_{24} & \neg(p \leftrightarrow q) & \Leftrightarrow & p \leftrightarrow \neg q \\
E_{25} & p \leftrightarrow q & \Leftrightarrow & (p \to q) \wedge (q \to p) \\
E_{26} & p \leftrightarrow q & \Leftrightarrow & (p \wedge q) \vee (\neg p \wedge \neg q)
\end{array}
$$

Implication relations:

$$
\begin{array}{lrcl}
I_1 & p \wedge q & \Rightarrow & p \\
I_2 & p \wedge q & \Rightarrow & q \\
I_3 & p & \Rightarrow & p \vee q \\
I_4 & q & \Rightarrow & p \vee q \\
I_5 & \neg p & \Rightarrow & p \to q \\
I_6 & q & \Rightarrow & p \to q \\
I_7 & \neg(p \to q) & \Rightarrow & p \\
I_8 & \neg(p \to q) & \Rightarrow & \neg q \\
I_9 & \neg p \wedge (p \vee q) & \Rightarrow & q \\
I_{10} & p \wedge (p \to q) & \Rightarrow & q \\
I_{11} & \neg q \wedge (p \to q) & \Rightarrow & \neg p \\
I_{12} & (p \to q) \wedge (q \to r) & \Rightarrow & p \to r \\
I_{13} & (p \vee q) \wedge (p \to r) \wedge (q \to r) & \Rightarrow & r
\end{array}
$$

Exercise 2.1 (Truth table calculation)**.** Using truth tables, decide whether the following two propositions are equivalent.

- $(A \vee B) \wedge \neg(A \wedge B)$
- $(A \wedge \neg B) \vee (B \wedge \neg A)$

Exercise 2.2 (Transitivity or $\to$)**.** Show that $(p \to q) \wedge (q \to r) \Rightarrow (p \to r)$ using: (1) truth table and (2) equivalence relations. Note that this is implication relation I_{12}.

Exercise 2.3 (Proof by contradiction)**.** Show that $(p \vee q) \wedge (p \to r) \wedge (q \to r) \Rightarrow r$, that is I_{13} in the table. Use a proof by contradiction.

Exercise 2.4 (Distributivity of $\cup$ and $\cap$)**.** Show the following using membership tables:

a) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$,

b) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

Exercise 2.5 (Membership table calculations)**.** Using a membership table, show that $\sim(\sim((A \cup B) \cap C) \cup \sim B) = B \cap C$.

Exercise 2.6 (More set theory)**.**

a) Is it *always* the case that $A \times B \neq B \times A$?

b) Show that $(A \cap B) \times C = (A \times C) \cap (B \times C)$ using the definitions of $\cap$ and $\times$.

Solutions

Solution to exercise 2.1 We start off with the truth tables for $A \vee B$ and $A \wedge B$ and then apply $\neg$ and $\wedge$ to these to form the first proposition.

A	B	$A \vee B$	$A \wedge B$	$\neg(A \wedge B)$	$(A \vee B) \wedge \neg(A \wedge B)$
F	F	F	F	T	F
F	T	T	F	T	T
T	F	T	F	T	T
T	T	T	T	F	F

Then we do the same for the second proposition.

A	B	$\neg A$	$\neg B$	$A \wedge \neg B$	$B \wedge \neg A$	$(A \wedge \neg B) \vee (B \wedge \neg A)$
F	F	T	T	F	F	F
F	T	T	F	F	T	T
T	F	F	T	T	F	T
T	T	F	F	F	F	F

Clearly, the two expressions to the far right in the tables are identical (and based on the same A and B values) which is what we wanted to show.

Solution to exercise 2.2 a) We build one big truth table, starting with p, q and r:

p	q	r	$p \to q$	$q \to r$	$(p \to q) \wedge (q \to r)$	$p \to r$	$(p \to q) \wedge (q \to r) \to (p \to r)$
F	F	F	T	T	T	T	T
F	F	T	T	T	T	T	T
F	T	F	T	F	F	T	T
F	T	T	T	T	T	T	T
T	F	F	F	T	F	F	T
T	F	T	F	T	F	T	T
T	T	F	T	F	F	F	T
T	T	T	T	T	T	T	T

b) Start from the given expression, but with $\to$ instead of $\Rightarrow$. Apply the equivalence relations, trying to condense the expression into a tautology:

$$
\begin{array}{cc}
(p \to q) \wedge (q \to r) \to (p \to r) & \overset{4\times E_{20}}{\Longleftrightarrow} \\
\neg((\neg p \vee q) \wedge (\neg q \vee r)) \vee (\neg p \vee r) & \overset{1\times E_3 + 2\times E_2}{\Longleftrightarrow} \\
(p \wedge \neg q) \vee (q \wedge \neg r) \vee (\neg p \vee r) & \overset{E_4+E_6}{\Longleftrightarrow} \\
(\neg p \vee (p \wedge \neg q)) \vee (r \vee (q \wedge \neg r)) & \overset{2\times E_9}{\Longleftrightarrow} \\
((\neg p \vee p) \wedge (\neg p \vee \neg q)) \vee ((r \vee q) \wedge (r \vee \neg r)) & \overset{2\times(E_{16}+E_{13})}{\Longleftrightarrow} \\
(\neg p \vee \neg q) \vee (r \vee q) & \overset{E_4+E_6}{\Longleftrightarrow} \\
(q \vee \neg q) \vee (\neg p \vee r) & \overset{E_{16}}{\Longleftrightarrow} \\
\mathbb{T} \vee (\neg p \vee r) & \overset{E_{14}}{\Longleftrightarrow} \\
\mathbb{T} & \text{Done!}
\end{array}
$$

Solution to exercise 2.3 To show that $(p \vee q) \wedge (p \to r) \wedge (q \to r) \Rightarrow r$ we assume that $(p \vee q) \wedge (p \to r) \wedge (q \to r)$ is true and that r is false and show that this leads to a contradiction ($\Rightarrow \mathbb{F}$).

$$
\begin{array}{cc}
(p \vee q) \wedge (p \to r) \wedge (q \to r) \wedge \neg r & \overset{E_8}{\Longleftrightarrow} \\
((p \wedge (p \to r) \wedge (q \to r)) \vee (q \wedge (p \to r) \wedge (q \to r))) \wedge \neg r & \overset{2\times I_{10}}{\Longrightarrow} \\
((r \wedge (q \to r)) \vee (r \wedge (p \to r))) \wedge \neg r & \overset{2\times I_1}{\Longrightarrow} \\
(r \vee r) \wedge \neg r & \overset{E_{10}+E_{17}}{\Longleftrightarrow} \\
\mathbb{F} & \text{Done!}
\end{array}
$$

Solution to exercise 2.4 a) For a given element in the universal set, there are eight possibilities. These are the eight rows in the "membership tables" below. The three leftmost columns show the membership of such an element (1 for included, 0 for not included) in the sets A, B, and C, respectively:

A	B	C	$B \cup C$	$A \cap (B \cup C)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	$A \cap B$	$A \cap C$	$(A \cap B) \cup (B \cap C)$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

The rightmost columns in the two tables are identical!

b) Similar reasoning yields:

A	B	C	$B \cap C$	$A \cup (B \cap C)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A	B	C	$A \cup B$	$A \cup C$	$(A \cup B) \cap (B \cup C)$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

The rightmost columns in the two tables are identical!

Solution to exercise 2.5 Starting from A, B and C, and using D as a name for the intermediate result $(A \cup B) \cap C$, to simplify the table:

A	B	C	$A \cup B$	$D := (A \cup B) \cap C$	$\sim D$	$\sim B$	$\sim D \cup \sim B$	$\sim(\sim D \cup \sim B)$	$B \cap C$
0	0	0	0	0	1	1	1	0	0
0	0	1	0	0	1	1	1	0	0
0	1	0	1	0	1	0	1	0	0
0	1	1	1	1	0	0	0	1	1
1	0	0	1	0	1	1	1	0	0
1	0	1	1	1	0	1	1	0	0
1	1	0	1	0	1	0	1	0	0
1	1	1	1	1	0	0	0	1	1

The rightmost two columns are identical, which is what we wanted.

Solution to exercise 2.6

a) No, for example, if $A = \emptyset$, then $A \times B = \emptyset = B \times A$. This is irrespective of B. (The same applies if $B = \emptyset$) Another example is if $A = B$.

b) By definition, $(A \cap B) \times C = \{\langle x, y\rangle \mid (x \in A \wedge x \in B) \wedge y \in C\}$. Also by definition, $(A \times C) \cap (B \times C) = \{\langle x, y\rangle \mid (x \in A \wedge y \in C) \wedge (x \in B \wedge y \in C)\}$. Just rearranging the terms $(E_5 + E_7)$ and eliminating (E_{11}) one occurrence of $y \in C$, which appears twice, gives us

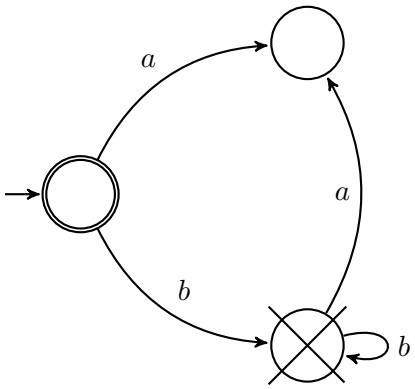
$$(A \times C) \cap (B \times C) = \{\langle x, y\rangle \mid (x \in A \wedge x \in B) \wedge y \in C\} = (A \cap B) \times C.$$

Chapter 3

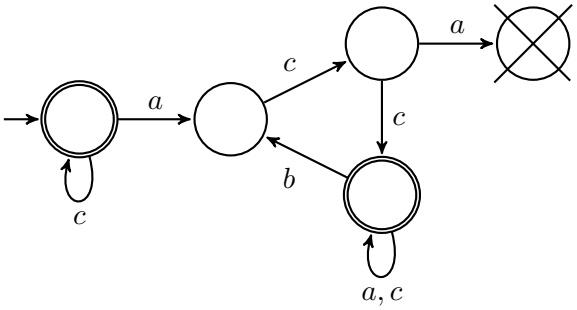
Formal Models

Exercise 3.1 (From automata to formal language)**.** What is the language, marked language and forbidden language of the following automata?

a)



b)



Exercise 3.2 (Languages to automata)**.** Draw (deterministic) automata with the following languages, use as few states as possible.

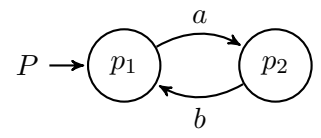
a)

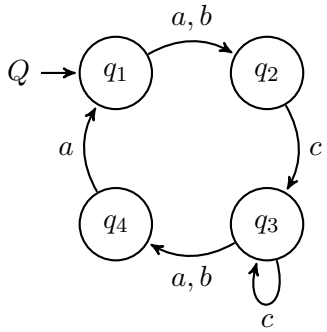
$$\begin{aligned} L(A_1) &= \overline{a^*(b^+c + c)b^+}, \\ L^m(A_1) &= a^*b^+, \\ L^x(A_1) &= \emptyset. \end{aligned}$$

b)

$$L(A_2) = \overline{a^*b(ca^*b)^*a},$$
$$L^m(A_2) = (a^*bc)^*, \text{ (This one is tricky!)}$$
$$L^x(A_2) = \emptyset.$$

Exercise 3.3 (Synchronous composition = language intersection)**.** Consider the two automata below.





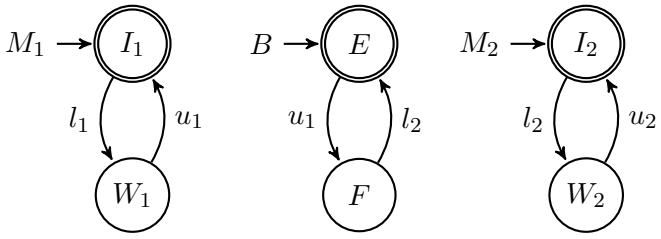
Consider both the case where $c \in \Sigma_P$ and the case where $c \notin \Sigma_P$.

a) Calculate the synchronous compositions.

b) Calculate the language intersections (remember that a language is a set of strings, and language intersection is thus an intersection of these sets).

c) Verify that automata of synchronous composition correspond to the intersection languages.

Exercise 3.4 (Associativity and commutativity)**.** Two machines, M_1 and M_2, are connected by a one-place buffer B. M_1 unloads stuff into the buffer, and M_2 loads stuff from the buffer.



a) Check whether $(M_1||B)||M_2 = M_1||(B||M_2)$.

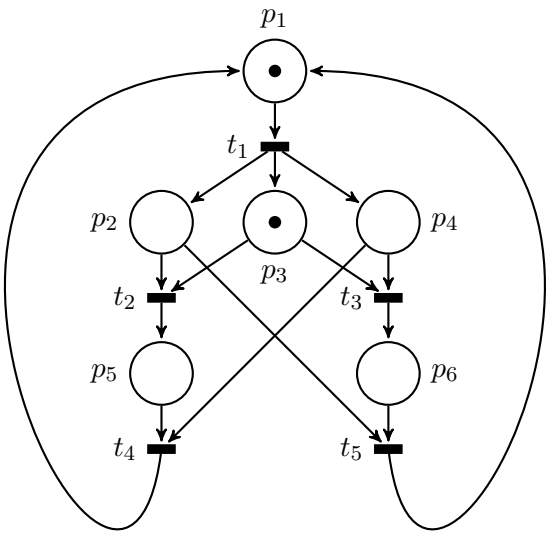
b) Check whether $M_1||B = B||M_1$.

Exercise 3.5 (Petri net reachability graph)**.** Construct the reachability graph for the Petri net below. Answer the following questions:

a) is the net bounded?

b) is it deadlock-free?

c) is it live?



Solutions

Solution to exercise 3.1

a)

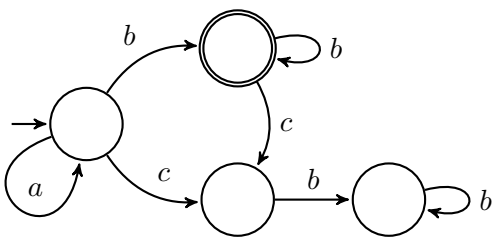
$$\begin{aligned} L(G_1) &= \overline{a + b^+ a}, \\ L^m(G_1) &= \{\epsilon\}, \\ L^x(G_1) &= b^+ + b^+ a. \end{aligned}$$

b)

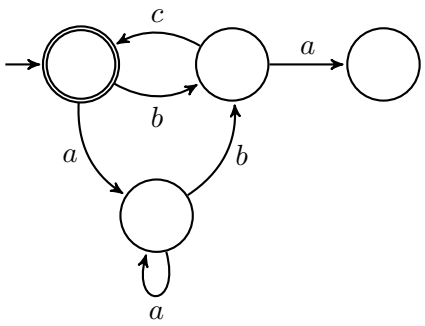
$$\begin{aligned} L(G_2) &= \overline{c^* ac(c(a+c)^* bc)^* a}, \\ L^m(G_2) &= c^* + acc(a + c + bcc)^*, \\ L^x(G_2) &= (c^* ac(c(a+c)^* bc)^* a). \end{aligned}$$

Solution to exercise 3.2

a)

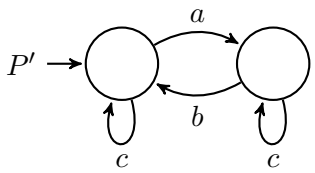


b) If you got a much simpler solution, make sure that the marked language is represented correctly (note that the language is not prefix-closed, which is denoted by an absence of the overline over the expression).

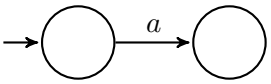


Solution to exercise 3.3 If two automata have the same alphabet, then the language of their synchronous composition is the same as the language intersection. If the alphabets are unequal, the result of the intersection may come as a surprise. It is very important to understand that when you perform language intersection, you are implicitly *assuming* that the alphabets are *equal*.

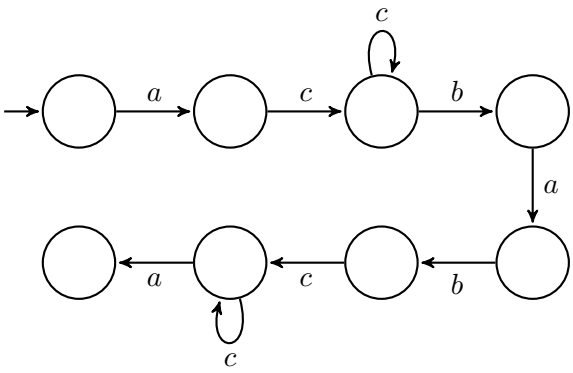
Since this is dangerous stuff, it is best to always be very careful with the alphabets. To get the right languages when alphabets are not equal, start by making alphabets equal. Calculate the union alphabet (union of the alphabets). If some automaton has less events in its alphabet than the union alphabet has, add *add self loops* for the missing events to all states of the automaton. This way automaton P, in case when its alphabet is $\Sigma_P = \{a, b\}$, will be converted to P':



a) Synchronous composition in case when alphabet of P includes event c ($\Sigma_P = \{a, b, c\}$):



Synchronous composition in case when alphabet of P does not include event c ($\Sigma_P = \{a, b\}$), $P||Q = P'||Q$:



b) Language intersection in case when alphabet of P includes event c ($\Sigma_P = \{a, b, c\}$):

$$L(P) \cap L(Q) = \overline{(ab)^*} \cap \overline{((a+b)c^+(a+b)a)^*} = \overline{a} = \{\epsilon, a\}$$

Language intersection in case when alphabet of P does not include event c ($\Sigma_P = \{a, b\}$). Since c is not in the alphabet of P, it is allowed to appear anywhere. We make it explicit by rewriting the language:

$$L(P) = \overline{(ab)^*} = \overline{(c^*ac^*b)^*} = L(P')$$

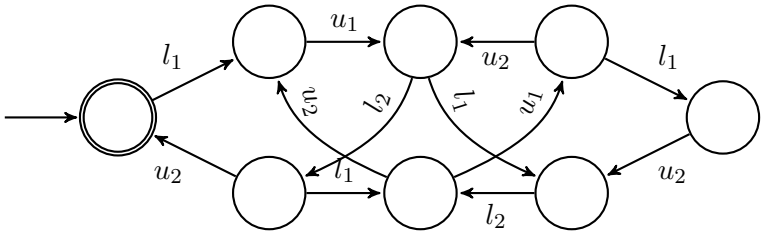
Then the intersection will be as following:

$$L(P) \cap L(Q) = L(P') \cap L(Q) = \\ = \overline{(c^*ac^*b)*} \cap \overline{((a+b)c^+(a+b)a)^*} = \overline{ac^+babc^+a}.$$

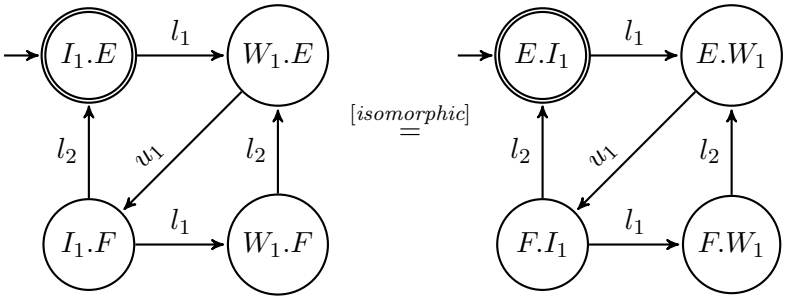
c) Comparing resulting languages and automata, it is easy to see that they accept the same strings.

Solution to exercise 3.4

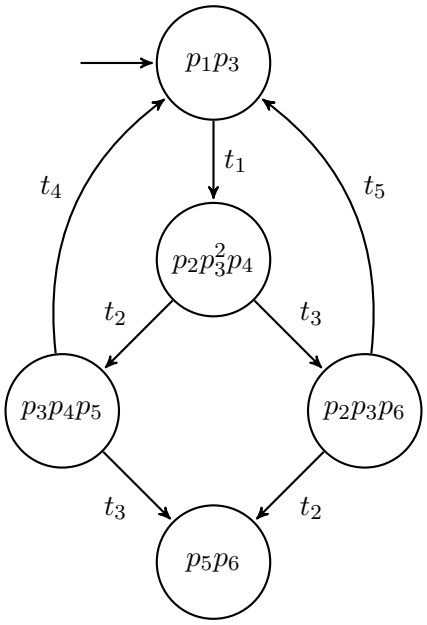
a) Yes, the synchronous composition, $||$, is *associative*. The synchronous composition (we can skip the parentheses) $M_1||B||M_2$ is:



Other than the state names (that are usually composed of the state names of the automata in the order they appeared) there is no difference. This is what we normally mean by the equality sign, $=$, when it comes to automata. In complicated words, $M_1||B$ is *isomorphic* to $B||M_1$ up to the renaming of states.



Solution to exercise 3.5 Reachability graph (the state $p_2p_3^2p_4$ represents one token in the place p_2, one in p_4, and two tokens in p_3):



a) The net is bounded (it has a finite number of states in its reachability graph).

b) The net has a deadlock state p_5p_6.

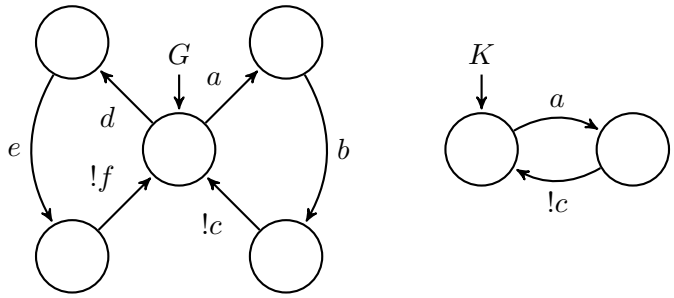
c) There are several levels of liveness. The net has the same level of liveness as all of its transitions. A transition is *dead* if and only if it can never fire (L_0-live). A transition is *potentially fireable* iff it may fire sometime. Transition can also fire arbitrary and infinitely often. A transition is *live* (L_4-live) iff it may always fire (from every state of the reachability graph the transition is potentially fireable).

   Each transition in this graph can be fired infinitely often from the initial marking, but there is a marking where no more transitions can be fired. So the net is "arbitrarily often"-live.

Chapter 4

Modelling and Specification

Exercise 4.1 (Plant with two loops)**.** Consider the plant G and the specification K below. a and d are two different commands for the plant, and c and f are two possible responses (uncontrollable, indicated by an exclamation mark !). It is desired that only the abc part of the plant is used.

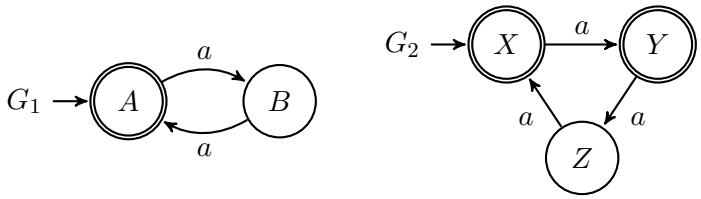


a) Calculate the synchronous composition $G||K$.

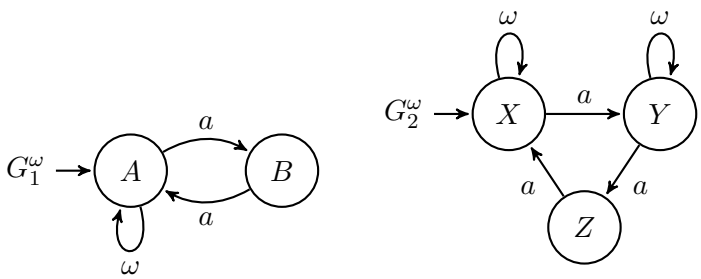
b) What is the problem with K?

Exercise 4.2 (Marked and forbidden states in synchronous composition)**.** This task aims to show that in synchronous composition, marking behaves just like events. In fact, marking is just a special case of an event.

Consider the two automata below.



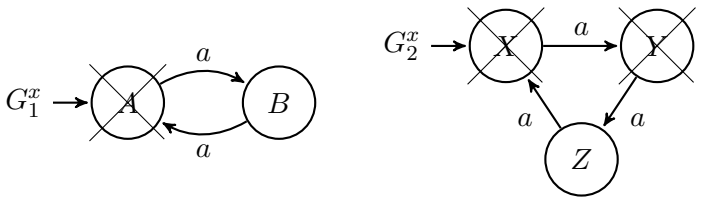
Let the states that are marked instead have a self-loop on the event ω that we call the "marking event". That is, we have the following two automata:



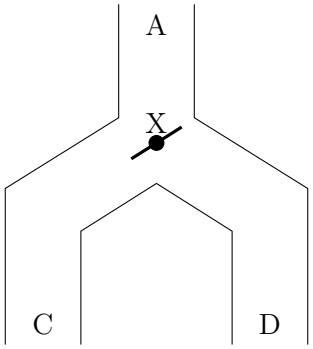
a) Calculate $G_1^\omega || G_2^\omega$.

b) Let all states in the synchronous composition that have self-loops on ω be marked instead. Compare the result to $G_1 || G_2$.

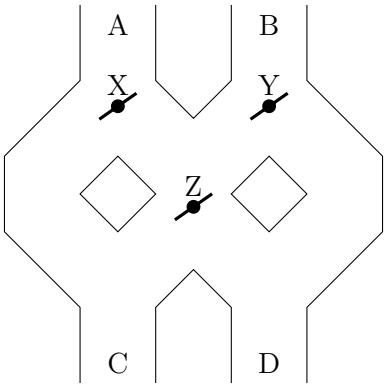
c) Is there a similar way to view *forbidden* states? Calculate the synchronous composition of the following two automata:



Exercise 4.3 (Pachinko game)**.** Consider the simplified Pachinko game shown to the left below. A ball is dropped in at A. The lever x causes the ball to either go left or right. In the state shown, the ball will bounce to the left, thus exiting at C. As the ball bounces off the lever, x will change state so that the next ball will take the opposite branch. Model this as a Petri net.



Then use the components developed for the simple case to model the more complex case.



Exercise 4.4 (Student and professor)**.** Consider a system consisting of a student and a professor. The student works with an assignment, hands it in, and waits for it to be passed or failed by the professor. If the assignment is passed, she cheers, and if it is failed, she sobs. The professor collects the hand ins and passes or fails them. She also mutters each time a hand in is failed.

a) Give a Petri net modelling the student and another one modelling the professor.

b) Synchronize the two nets from a) to a model for the entire system.

c) Give the corresponding state automaton.

Exercise 4.5 (Man, wolf, goat, cabbage)**.** A man together with a wolf, a goat and a cabbage head is on the left bank of a river. There is a boat large enough to carry the man and only one of the other three (yes, it is a large cabbage head!). The man and his company want to cross the river to the right shore, and the man can ferry each across, one at a time. However, if the man leaves the wolf and goat unattended on their shore, the wolf will surely eat the goat. Similarly, if the goat and the cabbage are left alone, the goat will eat the cabbage.

This "system" has a number of states denoting who is where. For instance, if the goat is on the left shore and the others are on the right, the state can be denoted $G - MWC$. The initial state is $MWGC - 0$. The state can change only on the occurrence of four events representing that the man and one of the others (or the man alone) crosses the river in the boat. The events are m, mw, mg and mc, representing that the man alone, the man and the wolf, the man and the goat, and the man and the cabbage cross the river, respectively. Let the same event denote crossing the river in either direction.

a) Model the system as an automaton. It has 16 states, some of which are forbidden and one is marked. Indicate these. Find the largest (in number of states) solution to the problem of getting from the initial state (the left shore) to the marked state without passing through any forbidden state.

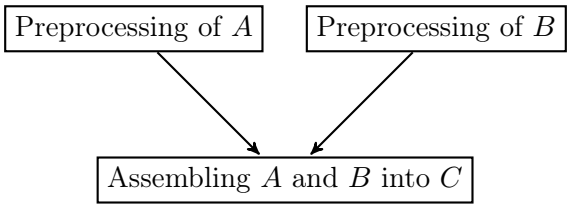
b) Model the system as four automata—one each for the man, wolf, goat and cabbage head—the synchronous composition of the four should have 16 states just like the solution in a). Hint: you must now use *different* events for the different directions of ferrying. Use event names mb, mwb, mgb and mcb for the ferrying in the "backwards" direction. Ignore the forbidden states for now.

c) *In the modular model*, add uncontrollable events that specify the forbidden states (as uncontrollable states). Hint: let the previous four automata be plants and add a specification that always disables the uncontrollable events representing "wolf eating goat", "goat eating cabbage". Is it enough with two uncontrollable events?

Exercise 4.6 (Reader and writer)**.** In a system, we have two writer-processes and three reader-processes. All of these use a mutual critical region. The processes read and write continuously, and then acquire the region, use it and release it. Only one writer-process can be allowed in the region at a time, provided that no reader is there. The reader-processes, on the other hand, can all read simultaneously, provided that no one is writing. The critical region must therefore be subject to mutual exclusion; one writer-process or up to three reader-processes at a time.

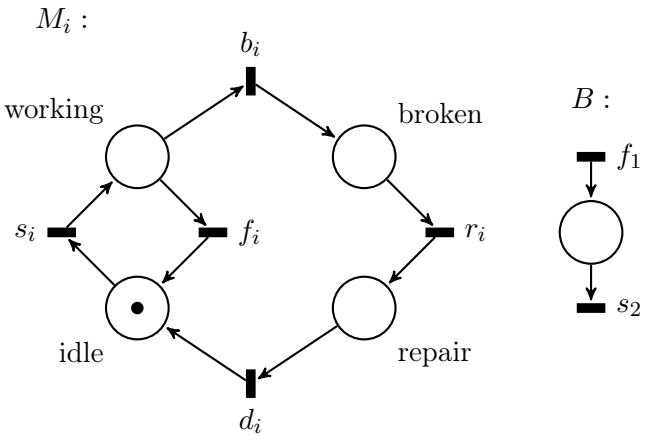
Draw a Petri net that models this.

Exercise 4.7 (Processing and assembling)**.** Work-pieces of two different types A and B arrive to an assembly cell. They are first preprocessed, and then assembled together to a product C. There are two tables in the cell and two robots. The tables are used for preprocessing. Preprocessing of the part A is performed on the *Table A*, and preprocessing of the part B on the *Table B*. Only one work-piece at a time can be processed at each table. Preprocessing is performed by a robot. The two robots are exactly the same, each robot can preprocess either part. After preprocessing two part are assembled together to a product C. For building a C product, one A and one B product is required. Assembling is performed by any of the two robots, without using any of the tables.



Formulate a Petri net for this cell. It could be worthwhile to consider modelling each robot by itself and then compose the cell model from these Petri nets. Could the robots be instantiations of a generic "preprocessing and assembly" robot?

Exercise 4.8 (Breakdown and repair)**.** A manufacturing cell consists of two machines M_1 and M_2 with a buffer B in-between them. Output from M_1 goes to B and then M_2 takes work-parts from there. The models are given below. The machines are instantiations of the same generic machine model M_i. The s_i event represents starting of machine i. The f_i event represents the finishing and ejection of the work-part. While processing the work part the machine can break down, b_i, and then has to wait to be repaired, r_i. When it has been repaired it becomes idle again, d_i. Work-pieces are stored in B automatically as M_1 finishes, and taken out of B as M_2 starts.



Formulate sub-specifications for the following cases:

a) The buffer is not supposed to hold more than three work-pieces.

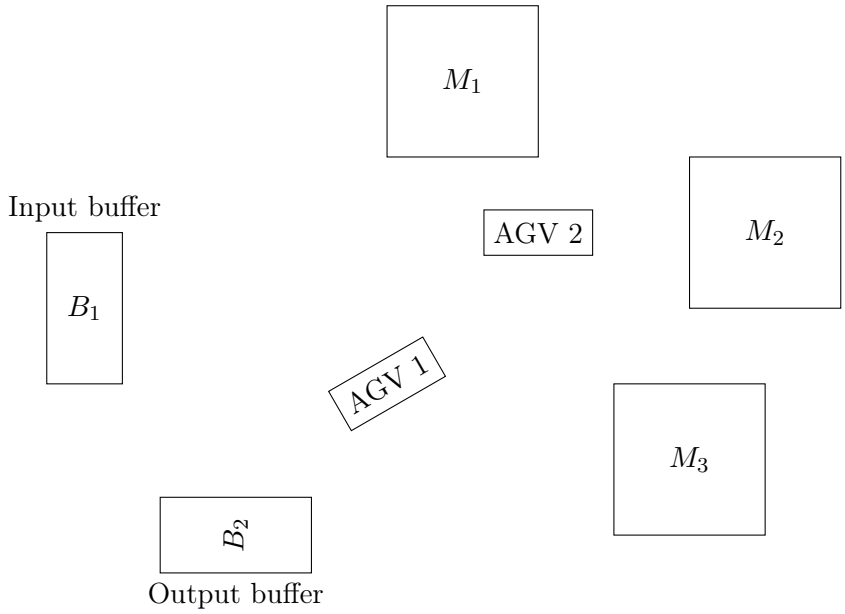
b) Only one machine can be repaired at a time.

c) If both machines are broken, M_2 is to be repaired first.

d) M_1 may not start if M_2 is broken.

Use the sub-specifications and the models above to formulate a total Petri net specification by synchronization.

Exercise 4.9 (Two AGV cell)**.** In a manufacturing cell there are three multi-operational machines, M_1, M_2 and M_3, served by two AGVs (automatically guided vehicles). Work-pieces are fetched from an input-buffer B_1 by an AGV transported to and loaded into a machine. After processing one of the AGVs unload the work-piece from the machine and transports it to the output buffer B_2. The two AGVs act as mutual resources shared between the machines. The output buffer can be considered to have infinite capacity. Formulate Petri net models for the components of this system. Formulate also a specification describing work-pieces entering at B_1 (event b_1) and being transported to any of the tree machines, processed and then unloaded and moved to the output buffer. Finally synchronize all the nets to generate S_0.



Note that the AGVs can be modelled by a single net with one token for each individual AGV. This is possible, since the AGVs have no individual characteristics that need to be distinguished. It could be worthwhile to consider also a system where this was not the case. If each AGV can serve only some of the machines or buffers.

Exercise 4.10 (The dining philosophers)**.** Three philosophers sit around a table. The only thing these philosophers can do is think and eat, and while they think they do not eat, and vice versa. In addition, the philosophers eat with chopsticks and they need two to be able to eat. Three chopsticks lay on the table, one between each philosopher. The philosophers can neither pick up nor put down both chopsticks simultaneously, but have to pick up or put down one on either side before the other. When a philosopher has picked up one chopstick, he does not put it down again until he has eaten.

Draw a Petri net that models the philosophers and their thinking and eating with the chopsticks.

Solutions

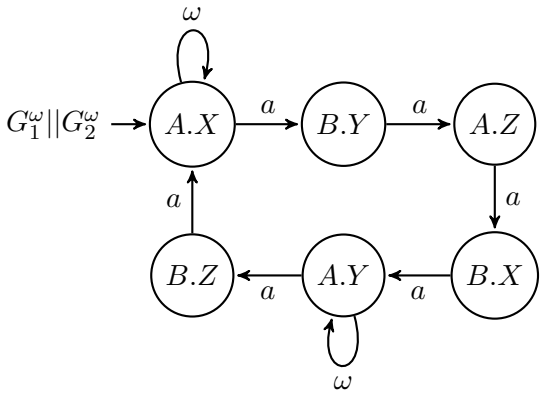
Solution to exercise 4.1

a) Perhaps surprisingly, $G||K = G$. The specification K does not restrict G at all! K only really says that a and c should alternate, which they do even if the plant sometimes goes through the def-loop.

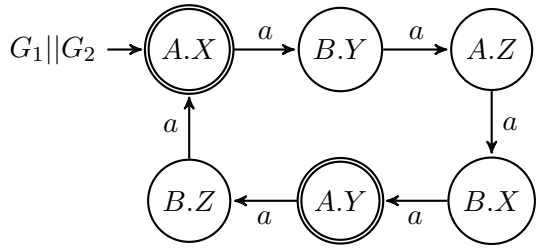
b) Since nothing else has been said, we have to assume that the alphabet of K is $\Sigma_K = \{a, c\}$ which means that it allows d, e, and f (and b) to occur at any time. Presumably, the intention was for $\Sigma_K = \{a, c, d\}$ which would have given the desired behavior in the plant.

Solution to exercise 4.2

a)



b) Replacing the selfloops with markings gives the automaton below:

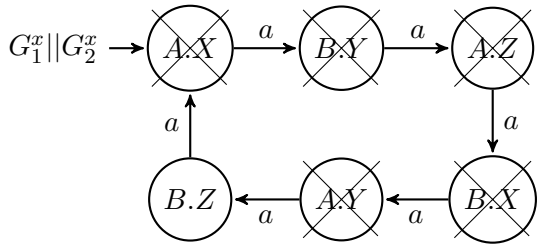


The result is the same as the synchronous composition of the original automata. As expected, in the composition, the markings show up in the states representing state combinations where the original automata are all marked (as in AND), i.e. the cross product

$$Q^m_{G_1||G_2} = Q^m_{G_1} \times G^m_{G_2} = \{A\} \times \{X, Y\} = \{A.X, A.Y\}.$$

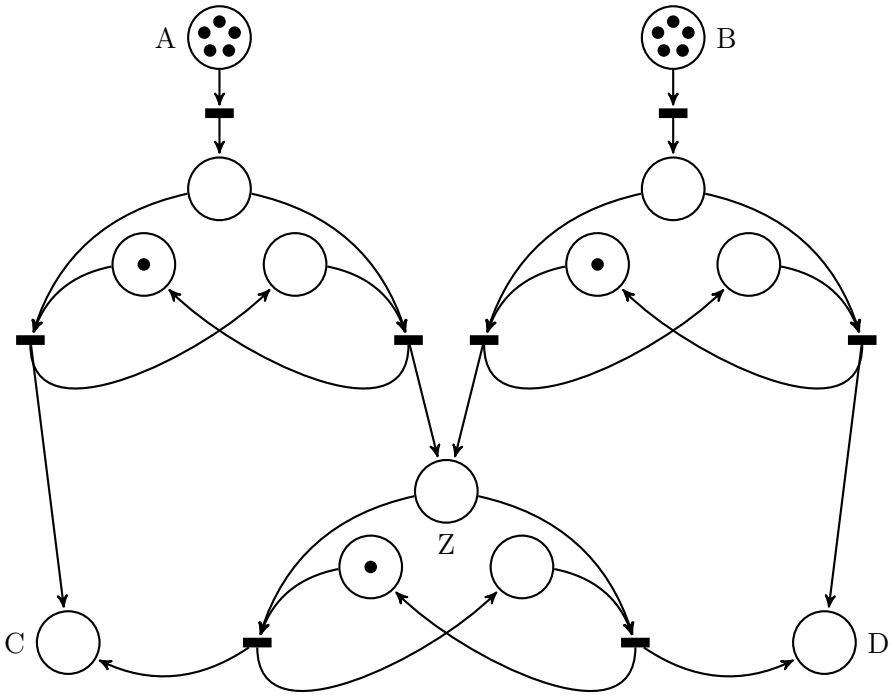
c) Forbidden states behave differently. The forbidden states in the composition are state combinations where at least one of the original automata is in a forbidden state (as in OR), i.e. the union of two cross products:

$$\begin{aligned} Q^x_{G_1||G_2} &= (Q^x_{G_1} \times Q_{G_2}) \cup (Q_{G_1} \times Q^x_{G_2}) = \\ &= (\{A\} \times \{X, Y, Z\}) \cup (\{A, B\} \times \{X, Y\}) = \\ &= \{A.X, A.Y, A.Z, B.X, B.Y\}. \end{aligned}$$



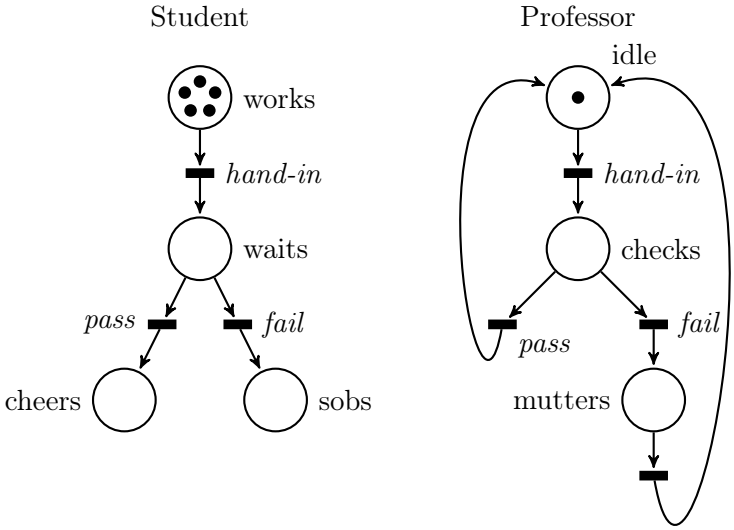
To represent the forbidden states using events, we would need different events for the forbidden states for different automata.

Solution to exercise 4.3 The following is a Petri net solving the whole problem. A fixed number of balls is located in places A and B. The balls exiting to the place C.



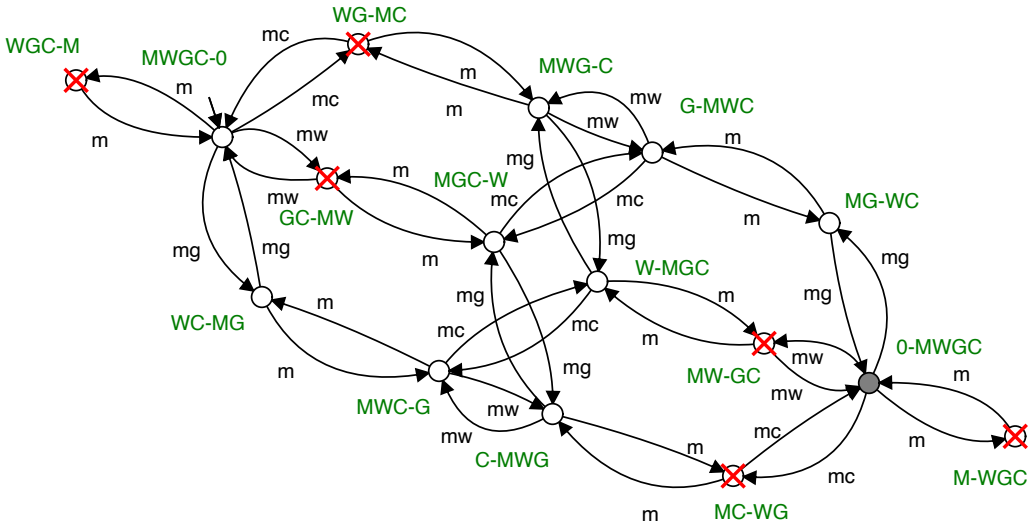
Another possibility is to represent inputs and outputs of the game as transitions instead of places.

Solution to exercise 4.4 One possible solution would be like the following:

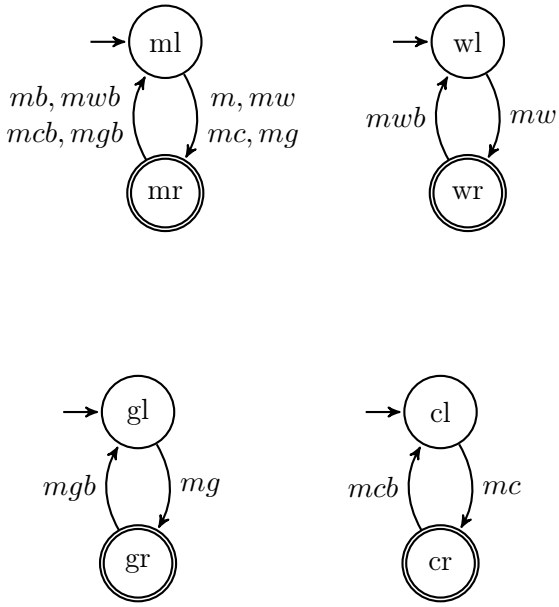


Solution to exercise 4.5

a) There are six forbidden states. If these are removed, it is clear that for example the sequence mg, m, mc, mg, mw, m, mg will solve the problem (reach the marked state).

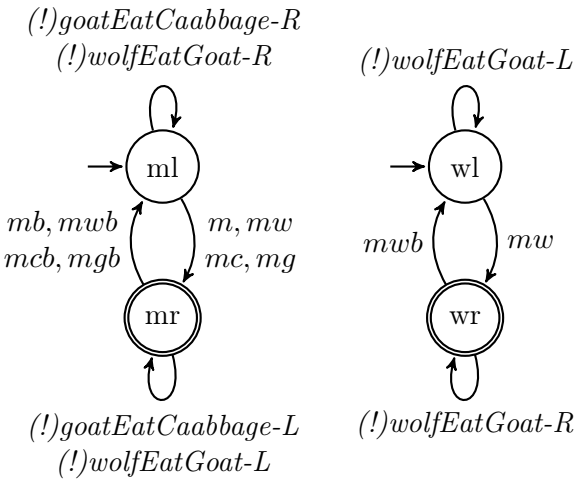


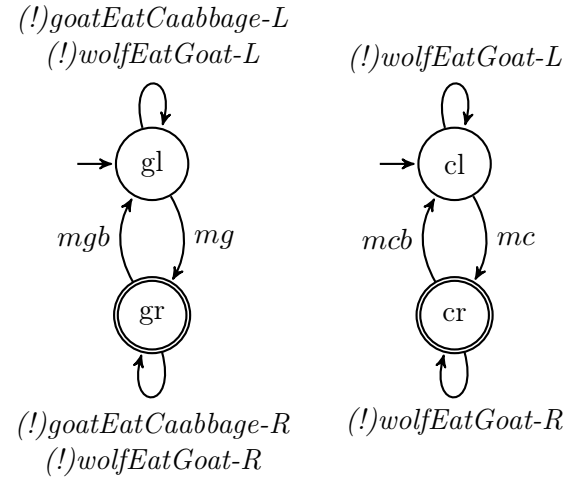
b) The *modular* model looks like this.



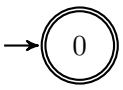
This model does not have any forbidden states.

c) To be able to pinpoint the forbidden state combinations exactly, we need different events for things being eaten at different shores of the river. So, using the event names *wolfEatGoatR* and *wolfEatGoatL* for wolf eating goat at right and left shore, respectively, and *goatEatCabbageR* and *goatEatCabbageL* correspondingly, the models become as follows.



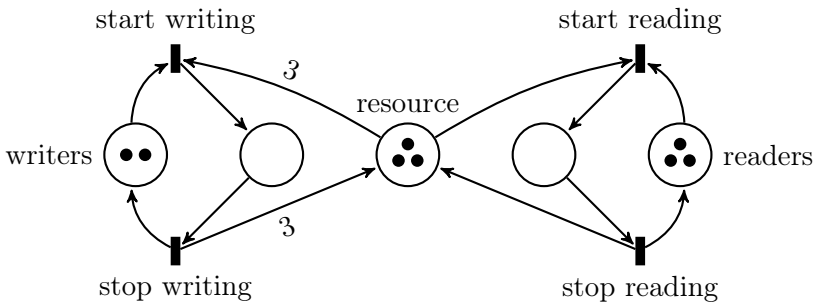


The specification is simply a one state automaton. Note that all the eat-events are uncontrollable and in the alphabet of the specification.

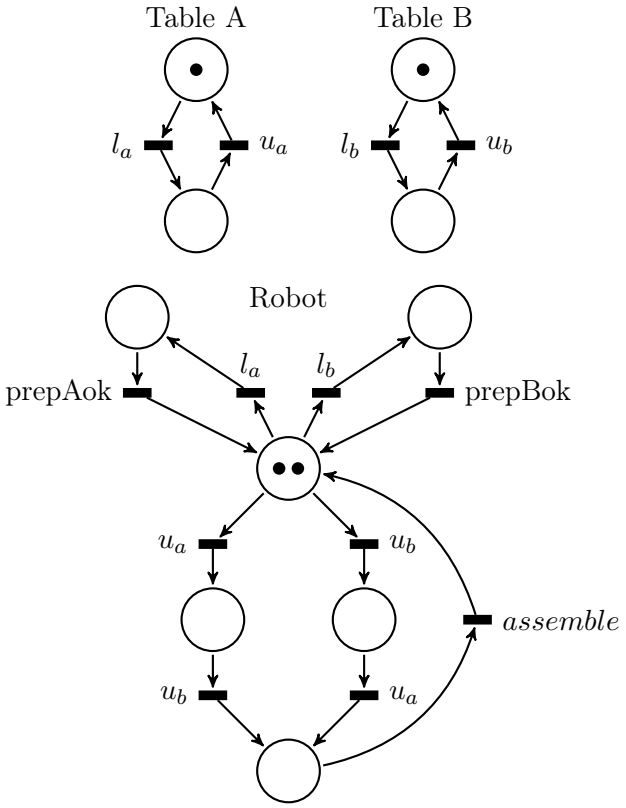


The synchronous composition of these five automata has 16 states of which six are uncontrollable and one is marked.

Solution to exercise 4.6 There can, of course, be drawn many Petri nets solving this problem, but this one is perhaps the simplest. There is a place that represents a resource. Writers are represented by two places, one of which represents the number of waiting writers, and another one the number of writing writers. Similar two places are used for readers. Moving a writer marker from waiting to writing place consumes all three markers of the resource, preventing the other writer and readers from using it. Readers consume one marker of the resource, allowing other readers to join in reading, but preventing any writer from starting writing.



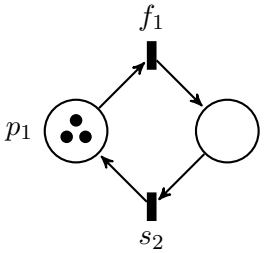
Solution to exercise 4.7



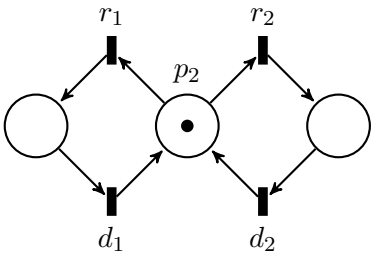
Note that in this model one robot can load *Table A* (l_a) and the other can immediately (try to) unload it (u_a) before the work-piece has been fully prepared. This is probably catastrophic, and must be avoided during production. However, it is possible in the uncontrolled system. Note also that one robot can pick up a prepared A while the other picks up a prepared B. In this state, the system will deadlock, each robot waiting for a piece of the other type but with no one to serve it. Again, this must be avoided during production.

Solution to exercise 4.8

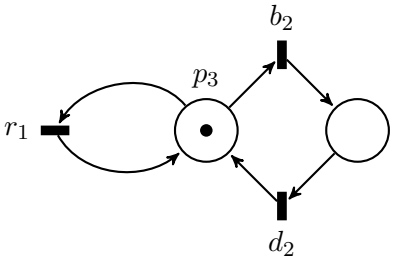
a) No more than three work-pieces in the buffer:



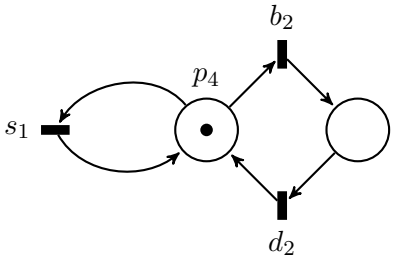
b) One repair at a time:



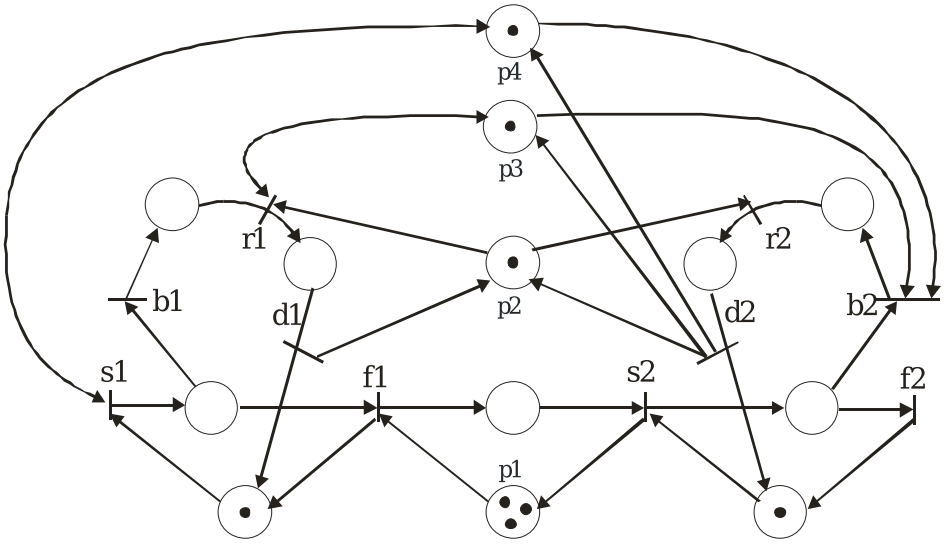
c) M_2 repaired first:



d) M_1 start only if M_2 is functional:



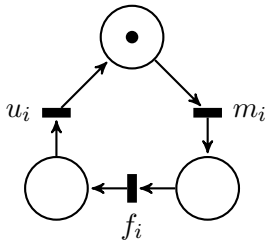
A total specification is achieved by synchronizing the sub-specifications above with the process models. This results in the Petri net below.



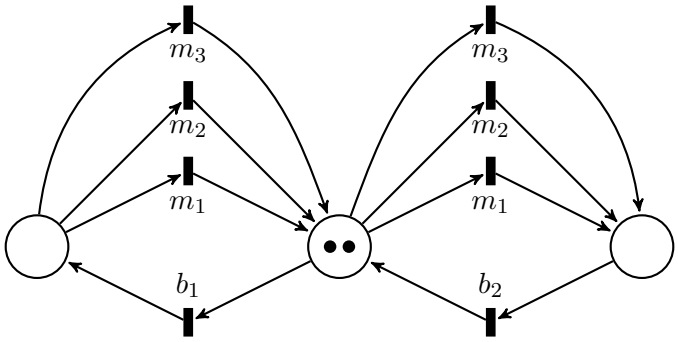
Note that the model of the buffer is equal to the first sub-specification. Thus, whether we want to view the buffer as resource or as a specification is not entirely clear. In fact, in many examples the buffer is viewed as specification only. In the composed net only one instance of the buffer has been included, though two instances could have been, one representing the buffer resource and one representing the buffer specification. This would have added nothing but complexity to the model, though. Some other, similarly redundant, places have also been removed for clarity.

Solution to exercise 4.9

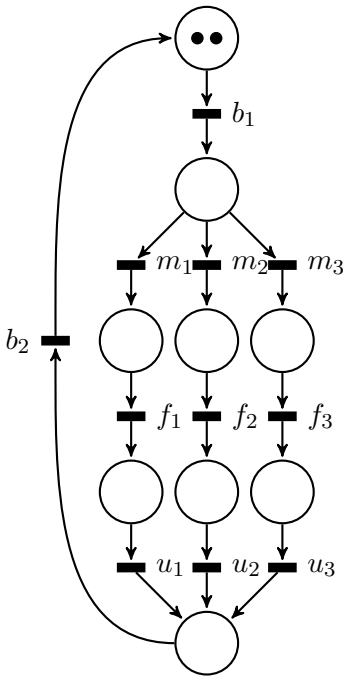
Machine:



AGVs:



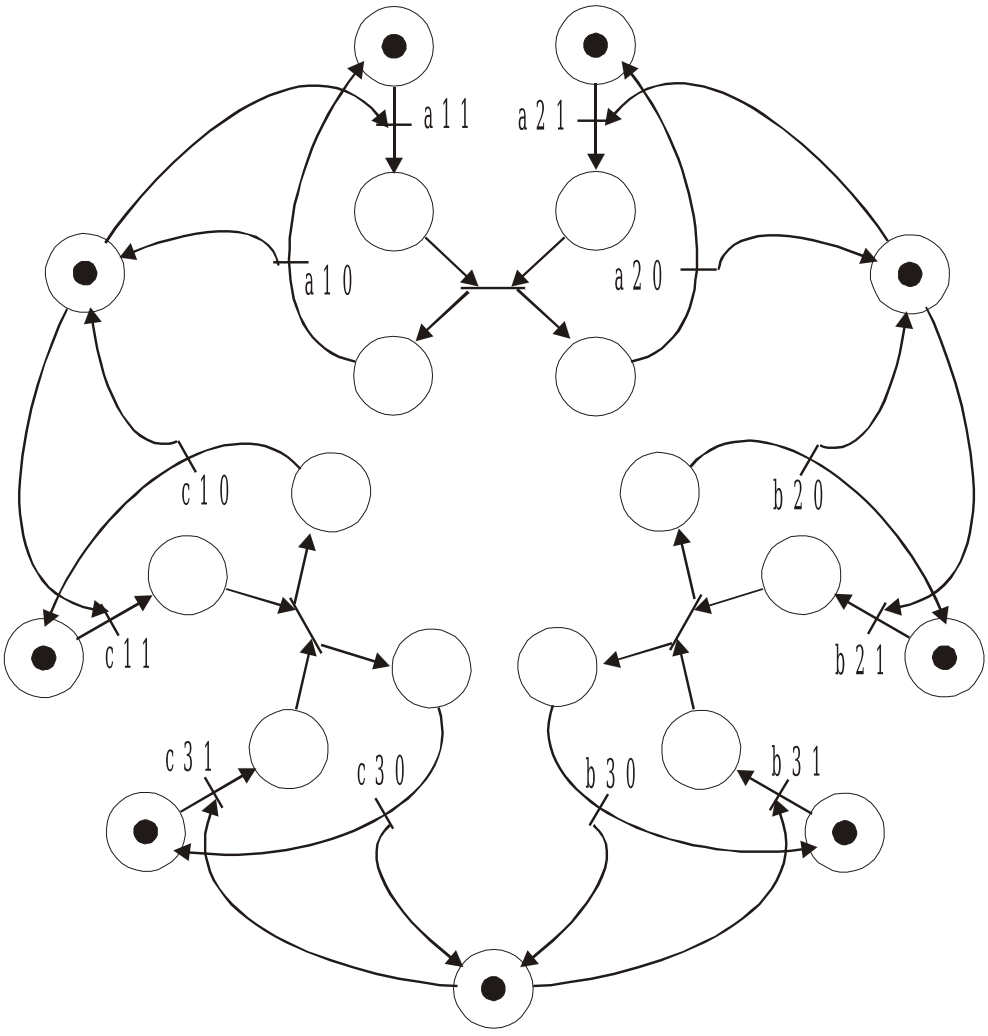
Specification:



The three machine models can be instantiated from M_i above. The two AGVs can be modelled as a single Petri net with two tokens. The specification specifies that a

product should first be fetched at the input buffer (b_1), transported to and loaded into either of the machines (m_1, m_2, m_3), processed until finished (f_1, f_2, f_3), unloaded (u_1, u_2, u_3) and transported to the output buffer (b_2). Note that the f_i are not necessary in the specification, since once loaded the machines are incapable of being unloaded unless they have finished their processing. Whether this models reality in enough detail or not, depends on the application. It could be that once loaded, the machines automatically begins processing. Note also that this model does not decide which machine is to be loaded until the work-piece has been picked up at the input buffer. Again, this may not model reality in enough detail. Furthermore, the AGV model allows the two AGVs to load and unload from and to the buffers simultaneously. As always, this may or may not be correct. The description of the system gives us no clues, though.

Solution to exercise 4.10 The philosophers are named a, b and c, and the chopsticks are numbered 1, 2 and 3. Zero means putting a chopstick down, whereas 1 means picking one up. Thus, the event $a11$ represents that philosopher a picks up chopstick number 1, while the event $b30$ represents that philosopher b puts down chopstick number 3.



Chapter 5

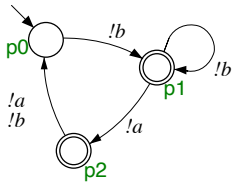
Implementation

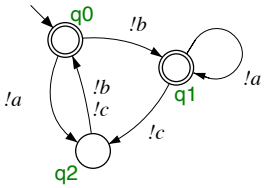
This chapter is no longer included in the course.

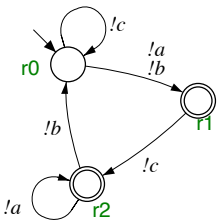
Chapter 6

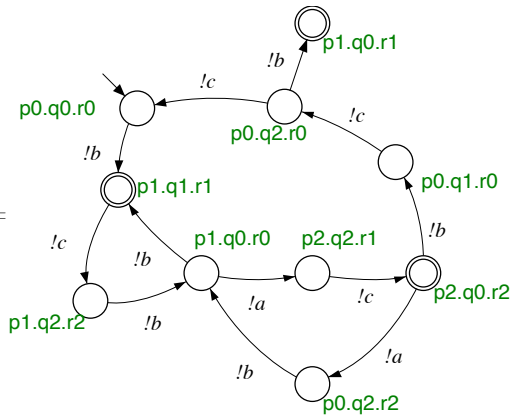
Verification

Exercise 6.1 (Specifications & verification)**.** Below are three automata, P, Q, and R, and their synchronous composition, $P \parallel Q \parallel R$. All three automata can be considered as *plants* and all events as *uncontrollable*.

$P =$ 

$Q =$ 

$R =$ 

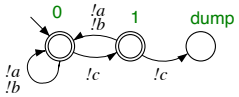
$P \parallel Q \parallel R =$ 

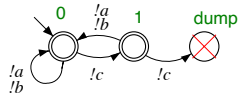
The task is to examine some properties & specifications. First, you should understand them, and then you should check if they are satisfied by the system...

a) Is $P \parallel Q \parallel R$ deadlock free?

b) Is $P \parallel Q \parallel R$ blocking or nonblocking?

c) Consider the following two specifications.

$K_1 =$ 

$K_1^x =$ 

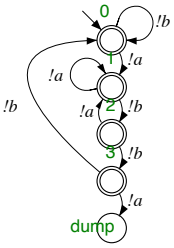
i) In words, what does the specification K_1 say?

ii) Is there any difference between K_1 and K_1^x in practice?

iii) Is this specification satisfied by $P \parallel Q \parallel R$? Why/why not?

iv) Change the specification so that instead of causing blocking when there is a problem, it causes a *controllability problem*.

d) Consider the following specification.

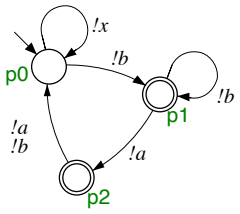
$K_2 =$ 

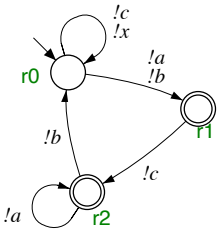
i) In words, what does the specification say?

ii) Is this specification satisfied by...

- P? Why/why not?
- Q? Why/why not?
- R? Why/why not?
- $P \parallel Q \parallel R$? Why/why not?

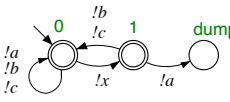
e) For the next specification, we want to point out a particular state in the plant. Therefore we change the plant model for P and R into the following...

$P' =$ 

$R' =$ 

That is, we added selfloops on the new, uncontrollable, event x in the initial states.

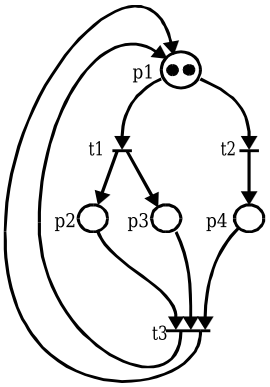
Now, consider following specification.

$K_3 =$ 

i) In words, what does the specification say? (This one is tricky!)

ii) Is this specification satisfied by $P' \| R'$? By $P' \parallel Q \parallel R'$? Why/why not?

Exercise 6.2 (Reachability graph II (from [Fab04]))**.** Consider the following Petri net.



a) Verify, using a reachability graph, that this Petri net is bounded and that a deadlock can occur.

b) Modify the net so that the deadlock is avoided and verify the modified net. No part of the original net may be altered and the new net must be bounded.

c) Deadlock may be avoided by adding logical conditions such that a certain transition may not be fired when the marking vector has a specific value corresponding to a certain condition. Formulate such logical conditions, using the reachability graph of a).

Exercise 6.3 (The stick-picking game (from [Fab04]))**.** Two people named A and B are playing a simple game. A number of sticks are lain out on the ground and the players

take turns in picking up one, two or three sticks. Note that at least one stick must be picked. The player that ends up with the last stick has lost the game. Player A is always the one that starts picking sticks.

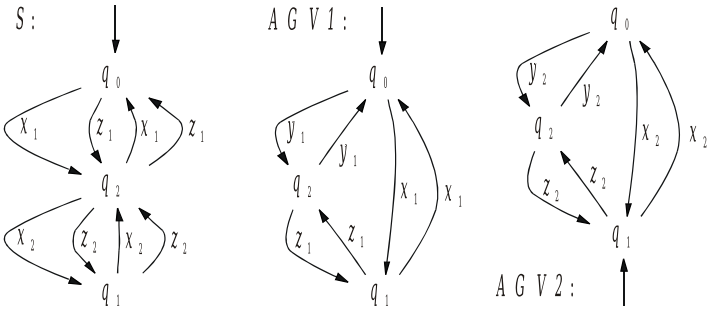
a) Model this game as a Petri net, with an initial arbitrary number of sticks (and weighted arcs).

b) For an initial number of seven sticks, give the reachability graph of the Petri net.

c) For the seven sticks game, specify that the A player is to win and the B player is to loose. Remember that the player left with only the final stick to pick, is the loser.

d) Is this a static or dynamic specification? Note that now is a good time to determine the set of uncontrollable events. (Hint, since we are to guarantee that A wins, we should view the game from A's perspective.)

Exercise 6.4 (Controllability and verification (SKS 000306, U1))**.** Two automated guided vehicles (AGVs), are cooperating on a small factory floor. For controlling the AGVs, the floor is divided into three zones, named 0, 1, and 2. The zones are all adjacent so an AGV can move from its current zone to any other zone at all times. Cameras in the roof identify the AGVs as they cross the zone borders and sends this information (as an event) to a central supervisor, S. The same event is sent regardless of the direction in which an AGV is moving between two zones, but these events are different for different AGVs. The task of the supervisor is to prevent the two AGVs from simultaneously occupying zone 0 and zone 2. On the other hand, the AGVs *must* be able to occupy zone 1 simultaneously as they need to cooperate for a particular task there. Automata models of the AGVs and the supervisor are given below. Initially, AGV1 is in zone 0 and AGV2 in zone 1.



The events y_1 and y_2 are uncontrollable, and all others controllable. Note that the alphabets of the automata only consist of the events shown explicitly. Especially, S only has a subset of the events of the two AGVs.

a) Check that S guarantees that there is never more than one AGV in zone 0 and in zone 2. (2p)

b) Check whether S allows both AGVs to be in zone 1 simultaneously. (1p)

c) Verify that S is controllable. (2p)

Exercise 6.5 (Modelling using Petrinets (HBS 060520, U8))**.** The industrial robots, R_1, R_2, and R_3, are working in a cell. In the cell there are three tools, T_1, T_2, and T_3, of which each robot needs *two* in order to perform its work. All three tools are identical so it does not matter which two tools are used by a robot. The robots can only pick up one tool at a time. Each robot returns both tools simultaneously when it is done. If a robot has picked up a tool, it won't return it until it has picked up the other tool and performed its work.

a) Model the system using ordinary Petrinets.

b) Show that the system has a deadlock.

c) Suggest a way of guaranteeing that this deadlock will not occur, without introducing new deadlocks and without significantly limiting the function of the cell. Add your control logic directly into the original Petrinet, but make sure that it is clear what represents the uncontrolled model and what represents the supervisor function.

Solutions

Solution to exercise 6.1

a) No, the state p1.q0.r1 is a deadlock!

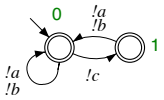
b) It is nonblocking—it is possible to reach a marked state from all reachable states. This includes the deadlocked state (since it is also marked)! If this came as a surprise, think through this again to make sure you understand the difference between deadlock and blocking.

c) i) c must never occur twice in a row (without a or b occurring in between).

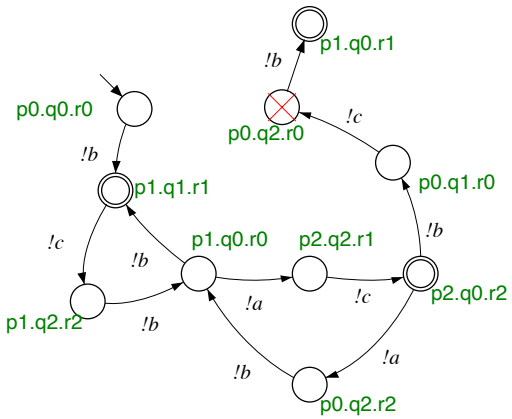
ii) There is no practical difference, K_1^x is a bit clearer since it explicitly points out that the blocking state is forbidden.

iii) This specification is *not* satisfied. For example, the string *bcbacbcc*, which is in the language of $P \| Q \| R$, ends with two cs in a row... this causes a blocking situation since the specification can never reach a marked state.

iv) To cause a controllability problem on the second c in a row, we just need make sure that the specification tries to *disable* the second c. Here, we can do this by simply removing the transition to the deadlocked state:

$K_2' =$ 

The synchronous composition of $P \parallel Q \parallel R$ and the specification is shown below, with the uncontrollable state forbidden.



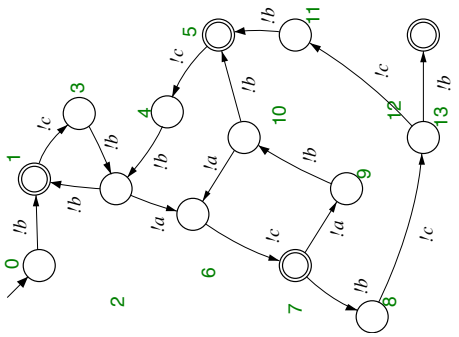
That is, we have found two ways of expressing the exact same specification. K_2 specifies the "bad behaviour" by causing blocking, and K_2' through a controllability problem.

d) i) It says that the sequence *abba* must never occur, ignoring any occurrences of c.

ii) The specification is violated by all three of P, Q and R (yes, R too since c events are ignored (not in the alphabet of the specification)).

- In P, the sequence *babba* (for example) leads to a blocking state in K_2.
- In Q, the sequence *abba* (for example) leads to a blocking state in K_2.
- In R, the sequence *acbbca* (for example) leads to a blocking state in K_2.

However, strange as it may seem, the specification *is* satisfied by $P \parallel Q \parallel R$. This can be decided by visual inspection or by composing $P \parallel Q \parallel R$ with the specification, resulting in the following automaton, which is nonblocking (right?).



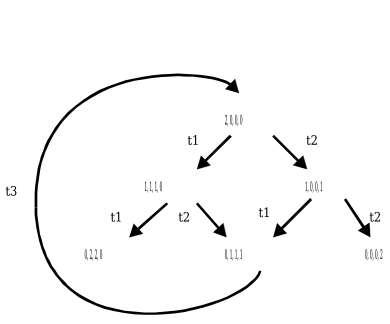
e) i) When P and R are in their initial states, a must not occur.

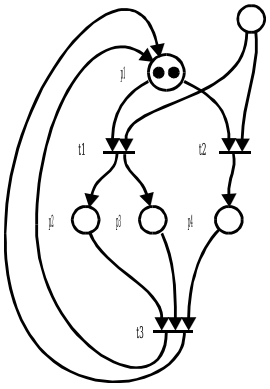
ii) Actually, a is not even enabled in the initial state of P, p0, so it can not occur in the initial state of $P' \parallel R'$ either. Since the specification is satisfied by $P' \parallel R'$ it must also be satisfied by $P' \parallel Q \parallel R'$; remember, synchronous composition is language restriction, so no new transitions are introduced.

Maybe that task was too hard... well... it builds character!

Solution to exercise 6.2

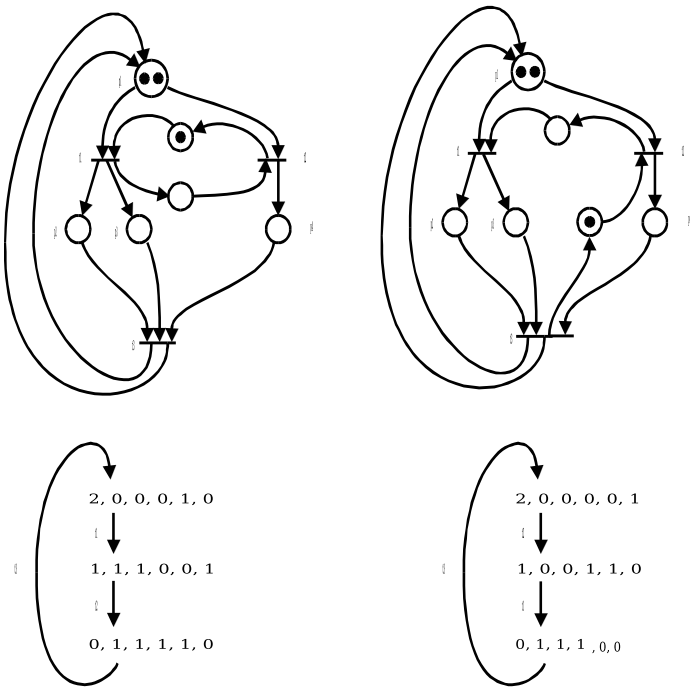
a) The reachability graph of the net can be drawn as below, left. As we can see, if t_1 is fired in (1,1,1,0) or t_2 is fired in (1,0,0,1), the net will deadlock. As is also obvious, no place is ever marked by more than two tokens, and therefore the net is bounded. Of course, to remove the deadlock we must prevent t_1 or t_2 from firing in (1,1,1,0) or (1,0,0,1), respectively. This can be done in several ways.



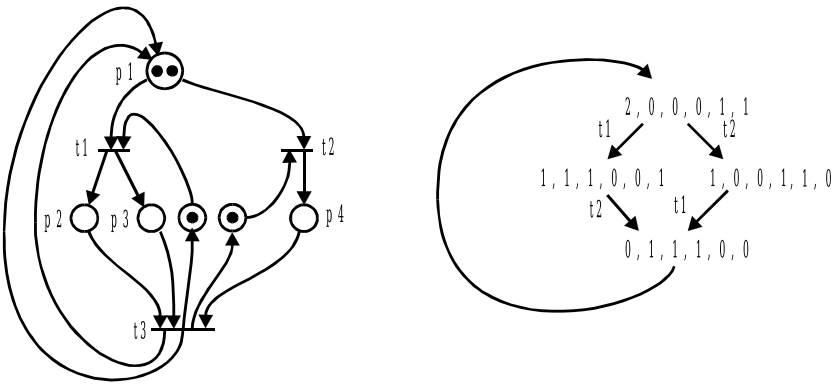


b) Surprisingly (perhaps), the net above to the right is a solution, given the problem formulation. The deadlock is removed, but only at the price of introducing a new one. This can be interpreted as very restrictive control. In order to prevent something bad from occurring, the controller allows nothing to occur. Such a solution is not uncommon, though, when it is generated by a computer algorithm (that is, synthesised). In fact, it can be shown that such a solution is always a valid

one, though not very useful in practice. Such a solution is therefore usually interpreted as the non-existence of a (useful) solution. Usually, something is required to happen, though. In that case, either of the two solutions below satisfies the requirements. The respective reachability graphs are shown underneath each net.



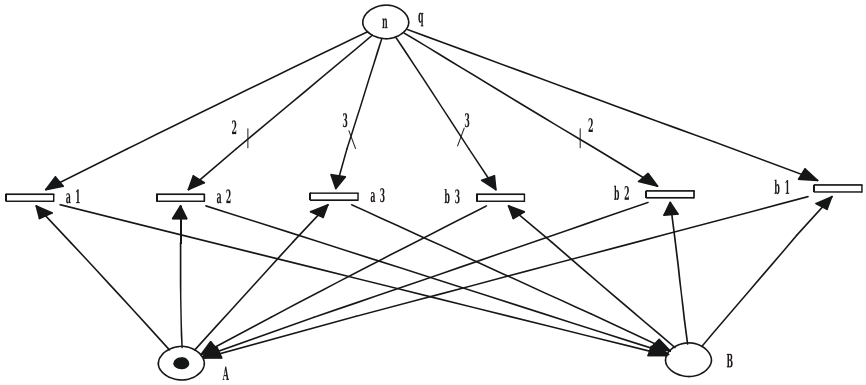
However, even this solution can be improved. For one thing, the two solutions above are not unique, and we would have to choose in some way between which one to consider as the "correct" one. It would be nice if a single unique solution did exist. Then we could consider that solution to be the "correct" one. Fortunately, such a unique solution does exist, and it is less restrictive than either of the two solutions above. This solution, shown below, is said to be the minimally restrictive one, since it only restricts exactly what is necessary to fulfill the specification, and no more. We will always be looking for this solution.



As we can see, only the two blocking branches have been removed, and no restriction is imposed on the firing-order of the t_1 and t_2 transitions.

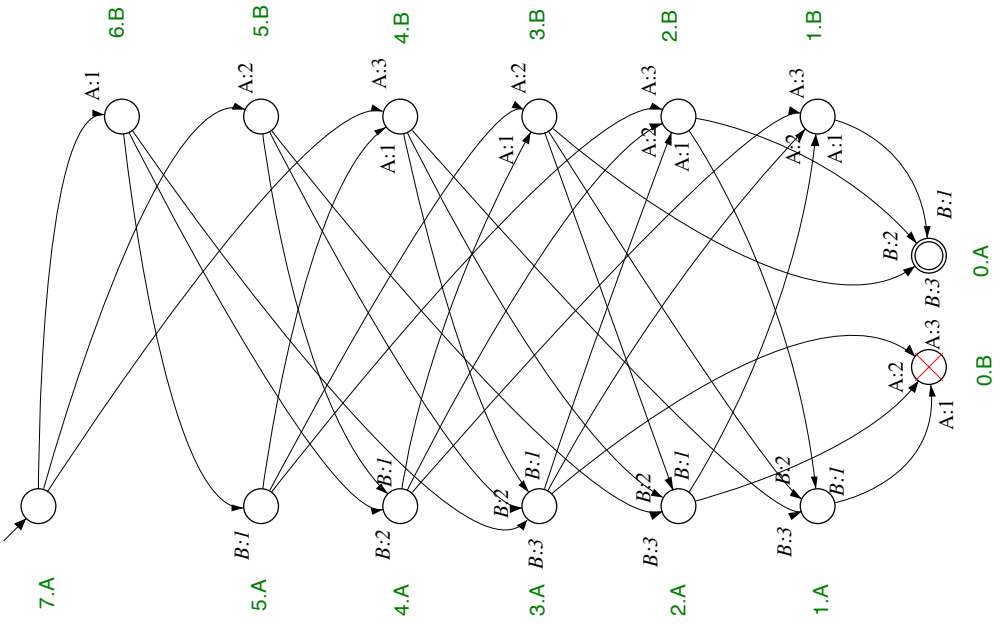
Solution to exercise 6.3

a) The Petri net is shown below. The picking of 1, 2 or 3 sticks is represented by the weighted arcs. A token in the A-place represents that player A is to pick sticks.



b)-c) The reachability graph, with $n = 7$, is given below. The states are named according to which places have how many tokens. Thus, the initial state $7.A$ represents that there are seven tokens in place q and one token in place A. This means that there are seven sticks left and it is A's turn to pick. Similarly, the state $1.B$ represents that there is one stick left and B is to pick, thus, A is winning.

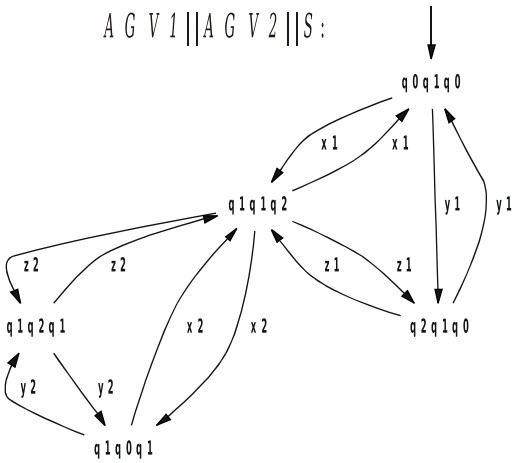
Since we view the game from A's perspective, it is reasonable to consider all moves by B as uncontrollable. A simply has no control over how many sticks B decides to pick.



d) The specification could be expressed solely by marking and forbidding states of the game model (our plant). Thus, this is a static specification with the $0.A$-state desired and $0.B$ forbidden, see above.

Solution to exercise 6.4

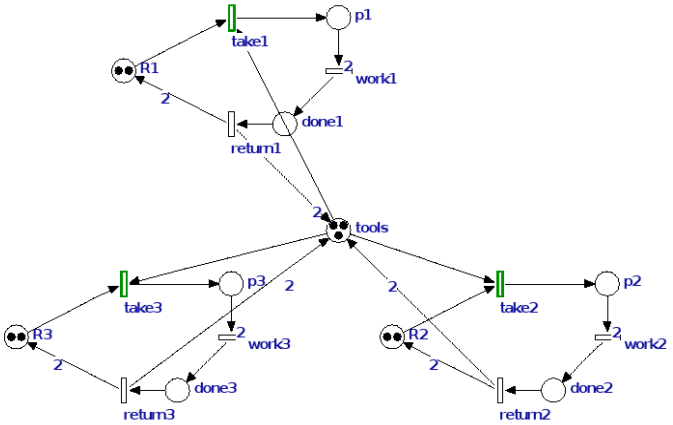
a) Each state of the AGV models can be seen as representing an AGV inside a particular zone. Composing AGV1, AGV2 and S gives the closed loop system. Since no states of the form "q0q0*" or "q2q2*" can be found, the AGVs won't ever occupy the zones simultaneously.

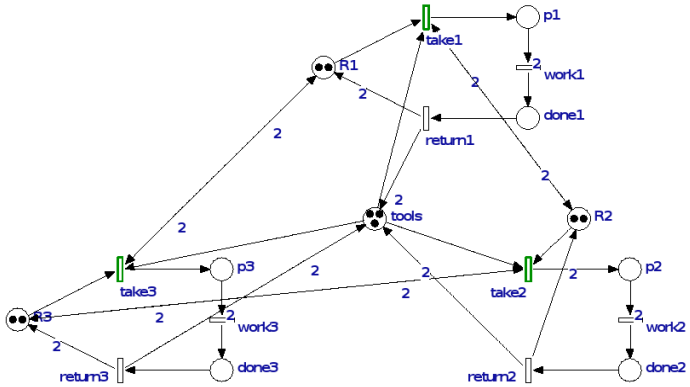


b) As above, it is easy to see that a state of the form $q1q1*$ (there is actually only one—$q1q1q2$) can be reached (from all reachable states, actually).

c) The simplest way is by observing that S has no uncontrollable events at all in its alphabet. This is enough since in order for S to be uncontrollable it must disable an uncontrollable event that the plant can execute but the events that aren't in S's alphabet are *never* disabled (by S).

Solution to exercise 6.5 Below is a Petri net model of this system. Of course, this system deadlocks as soon as each robot has taken one tool. Now, each robot is waiting for another tool to become free, while at the same time greedily holding on to its own tool; thus, the system deadlocks. One way to guarantee that no deadlock exists is to give priority to the robots so that R1 does not take any tool if R2 has taken one, and R2 does not take any tool if R3 has taken one, and R3 does not take any tool if R1 has taken one, see below. It turns out this scheme is also minimally restrictive.





Chapter 7

Controller Synthesis

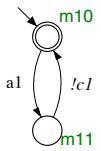
Exercise 7.1 (Man, wolf, goat, cabbage revisited)**.** Consider again the problem of the man with the wolf, the goat and the cabbage head (task 4.5 above). Assume that the wolf is a good swimmer that can instantaneously swim across the river to the other side (just as the man can instantaneously row across). Add two new events to the model, w that represents that the wolf swims across the river (from left to right), and wb that represents that the wolf swims back again. Both w and wb are uncontrollable.

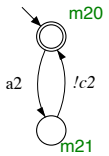
a) Assume that the man cannot stop the wolf from swimming across at all.

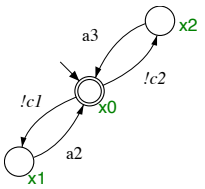
b) Assume that the man *can* hinder the wolf from swimming across if they are on the same side, but not if they are on opposite shores. That is, using selfloops, have the man disable the wolf from swimming away from him.

In both cases, calculate the supervisor that determines how the man should get from one shore to the other with his companions intact. Note that the solution to the original problem considered all events controllable, and is thus a supervisor for that case.

Exercise 7.2 (AGV system (from [Fab04] (adapted from [Won99])))**.** A work-cell consists of two machines M_1 and M_2, together with an automatically guided vehicle AGV. The AGV can be loaded with a work-part from either M_1 (event c_1) or M_2 (event c_2). The part is then transported either to M_2 (event a_2) or to an output buffer (event c_3). Models of the work-cell resources are given below.

$M_1 =$ 

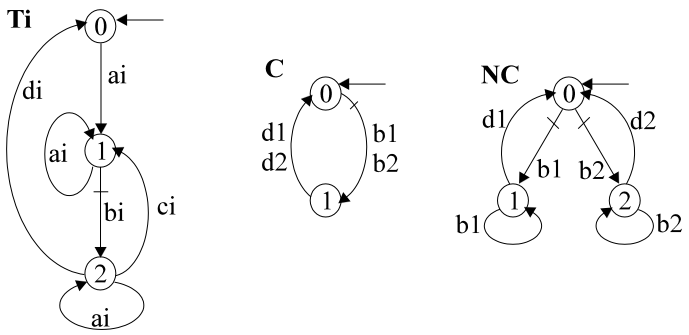
$M_2 =$ 

$AGV =$ 

Compose a model of the entire plant by synchronising the above models, $P = M_1 \parallel M_2 \parallel AGV$.

The uncontrollable events are $\{c_1, c_2\} = \Sigma_u$. Assume that the initial state is specified as the only desired (marked) state. From the plant model and this specification synthesise a controllable and non-blocking supervisor. Observe that the synthesis must be performed iteratively to make the supervisor both controllable and non-blocking.

Exercise 7.3 (Transmitter (from [Fab04] (adapted from [Won99])))**.** A transmitter can be modelled as a three-state automaton, see T_i below. The event a_i represents the reception of a message to be transmitted, b_i represents start of the transmission, c_i represents a timeout when a sent message has not been acknowledged and d_i represents a reset to transmit a new message. An un-acknowledged message is retransmitted after the timeout. New messages received while a message is being sent are discarded.

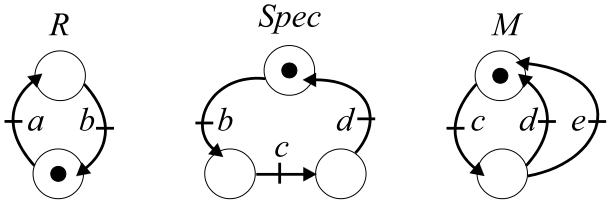


The process T consists of two transmitters, $T = T_1 \parallel T_2$ (9 states), with $\Sigma_c = \{b_1, b_2\}$ and all other events uncontrollable. The transmitters send on a channel that can only be used by one transmitter at a time. Of course, each transmitter must be able to send more than one message; the initial-state must therefore be marked.

Two different supervisors exist for this process, C and NC, see above. Both of these can be regarded as modelling the channel. Note that the supervisors do not have the same alphabet as T; rather $\Sigma_C = \Sigma_{NC} = \{b_1, b_2, d_1, d_2\}$. Both supervisors guarantee, for instance, that if b_1 occurs, then b_2 cannot occur until a d_1 has occurred.

Verify the two supervisors by composing the closed loop systems $T \parallel C$ (11 states) and $T \parallel NC$ (12 states), respectively. Are both controllable? Show and explain why C is not an acceptable supervisor, and why NC *is* acceptable. Show also that T is blocking (find a string that takes T to a blocking state).

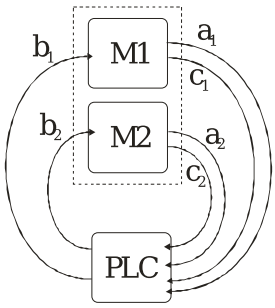
Exercise 7.4 (Robot & Machine (from [Fab04] (adapted from [Giu92]))**.** A robot R and a machine M are to work together, see the figure below. The robot fetches work-parts from some storage (event a) and loads the machine (event b). The machine processes the work-part (event c) and, when finished, places the part on either conveyor 1 (event d) or conveyor 2 (event e). This scheme can be described by the specification below. Naturally, the system should execute this cycle again and again. Thus, the initial state is specified as marked.



The only uncontrollable event is $\Sigma_u = \{b\}$. Generate a supervisor that guarantees the specified behaviour. Since the specification above is partial ($\Sigma_{Spec} \subset \Sigma_P = \Sigma_M \sup \Sigma_R$), a total specification must first be generated. This can be done in the form of a Petri net from which a controllable, minimally restrictive and non-blocking supervisor can be generated.

Exercise 7.5 (Modelling and synthesis (HBS 070120, U8))**.** We have two machines, M_1 and M_2, using a common tool. The machines work independently except for accessing the tool. The machines pick up the tool automatically, but not until they have received permission from the controller (the PLC). The machines ask for permission by setting a signal a_i and the controller answers the machine by in turn setting a signal b_i (i is the number of the machine). The machine then picks up the tool and uses it until it is done, puts the tool back, and then informs the controller that it is done by setting a third signal, c_i.

Note that a machine M_i that has just returned the tool may very well ask for permission to pick it up again immediately. On the other hand it is reasonable to assume that the machining always takes a while and that a machine only requests the tool when it actually has a need for it.



The "protocol" of machine i can be described as follows.

1) Not using tool.

2) Ask for permission—set a_i.

3) Wait for permission—wait for b_i.

4) Use tool.

5) Return tool, set c_i

6) Go to 1.

Initially, the tools is not used by any machine.

a) Formulate a modular discrete event model for the process. The rising edges (change from 0 to 1) of the signals can be seen as the events of the system. State which events are uncontrollable. (2p)

b) Give a modular specification describing that

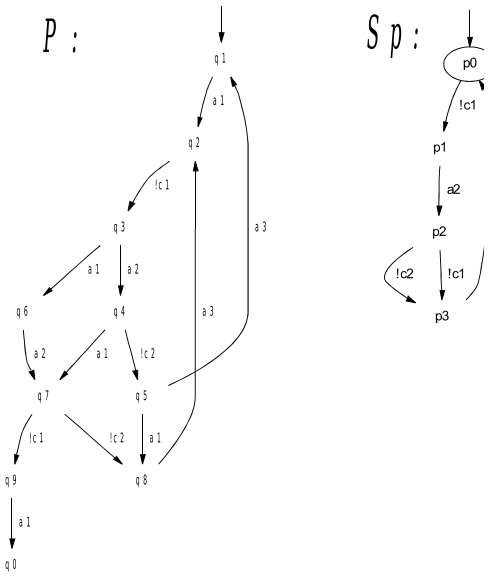
   i) both machines are not allowed to use the tool at the same time, and that

   ii) the machine that asks for permission first is the one that gets access first. (3p)

c) Synthesise a (controllable and nonblocking) supervisor that guarantees that the specifications are satisfied. (2p)

Note that we want automata models describing the specifications in task b). Of course, the supervisor should allow the procedure to repeat over and over.

Exercise 7.6 (Supervisor synthesis (HBS 060901, U6))**.** Below you will find two automata representing a process P and a specification Sp, respectively. The event $a_1 \in \Sigma_P$ is not a member of the alphabet of the specification. The events c_1 and c_2 are uncontrollable, the state p_0 is marked, and all states in P are marked even if this is not indicated in the graph.



a) Decide whether the specification is static or dynamic (relative to the process P). (2p)

b) Generate a nonblocking (and controllable) minimally restrictive supervisor that keeps the closed loop system within the specification. (3p)

Exercise 7.7 (Coffee machine)**.**

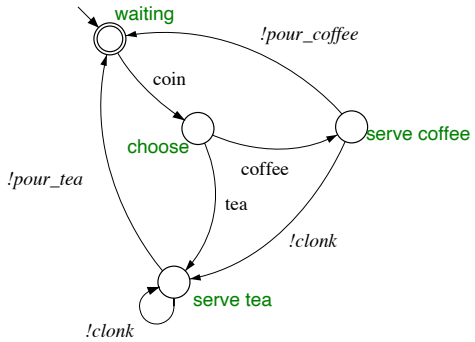
Some problems have been observed with a coffee machine and it has been decided that the user interface should be supplied with a supervisor to ameliorate the strange workings of the machine.

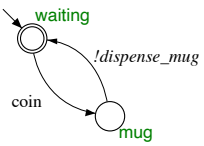
Supposedly, the machine should accept a coin, allow its user to choose either coffee or tea, and then serve the ordered drink in a mug.

However, there are a few problems with the machine.

- It sometimes goes "clonk" and serves the wrong drink.
- It sometimes pours the drink before dispensing a mug.

After careful observation, a modular plant model for the malfunctioning machine is created (the two automata below).





a) Create a specification saying that the drink that is served must be the drink that was ordered.

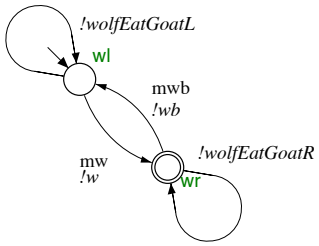
b) Create a specification saying that the mug must be delivered before the beverage.

c) Design a supervisor for the user interface (*coin*, *coffee*, and *tea* are the only controllable events).

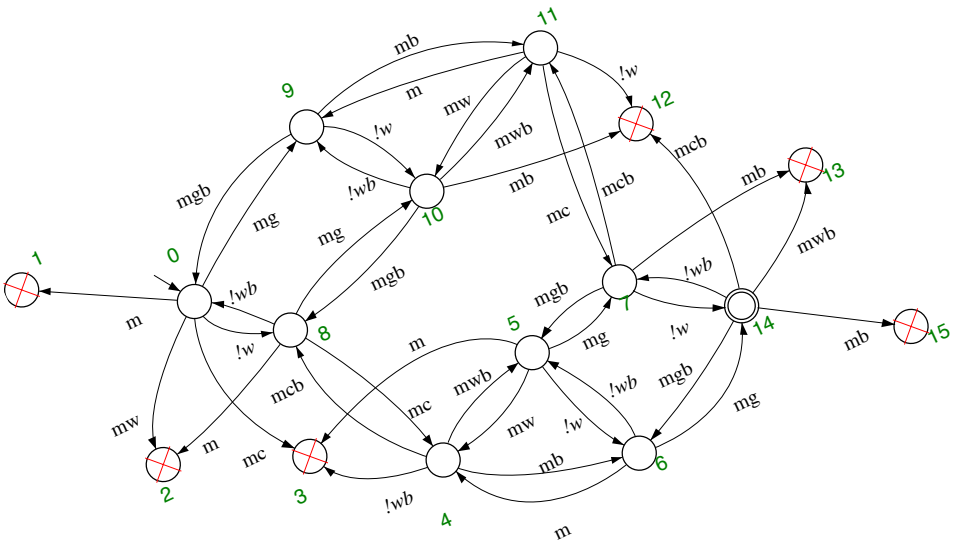
Solutions

Solution to exercise 7.1

a) The only change to the models is in the wolf-automaton. Note the exclamation marks denoting the uncontrollable events.

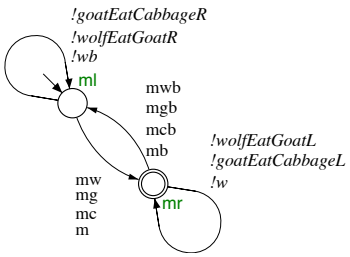


The figure below shows the synchronous composition of the plant (using the new wolf-automaton) and the specification. The uncontrollable states are shown as forbidden. To simplify the figure, all outgoing transitions have been removed from the forbidden states.



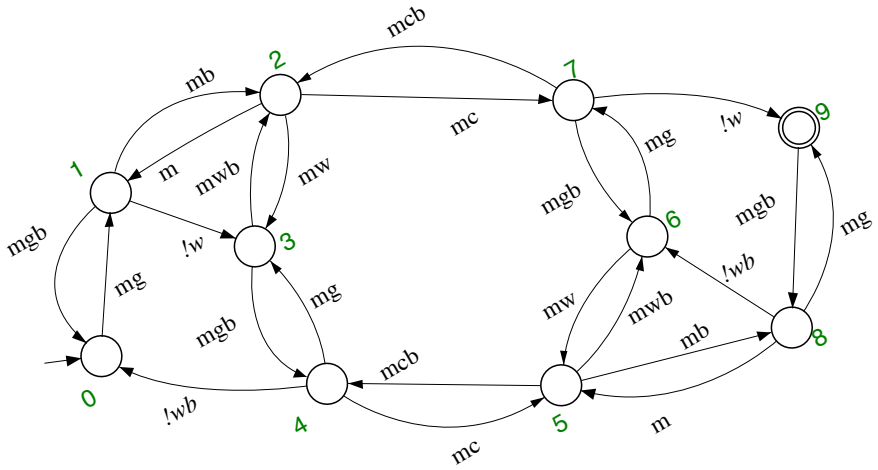
If the man can not control the wolf, there is no way to solve the problem (null supervisor). As soon as the man and the goat are not on the same shore, the wolf can always get to the goat and eat it. In the synchronous composition, this shows up as uncontrollable transitions leading to forbidden states. Especially, to reach the marked state, at some point, the system must be in either state 4 or 11. However, from both of these, there are uncontrollable transitions to forbidden states. The synthesis process will discover this and forbid both of those states. This leaves no way to get across the river without risking the goat being devoured by the wolf.

b) Assuming that the man *can* stop the wolf from leaving the same shore, the problem can be solved. The model for the man looks like this.

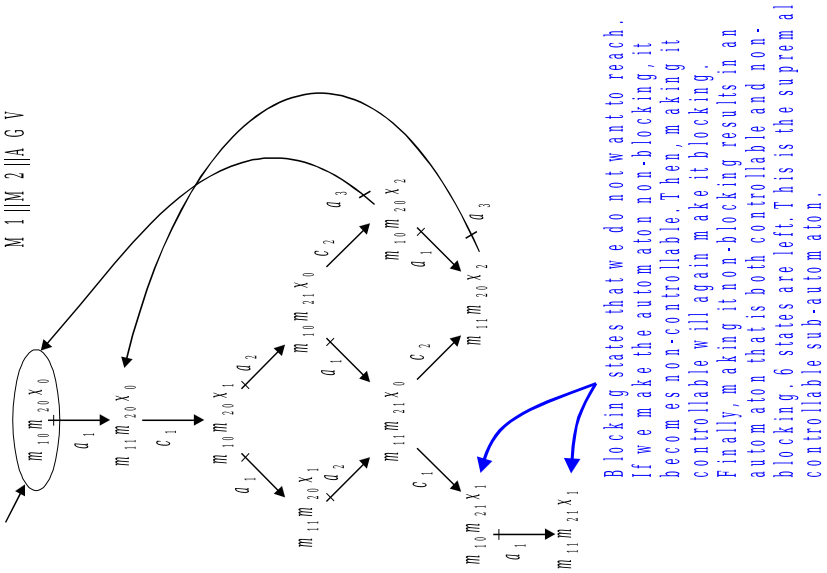


The selfloops control the wolf when it is on the same shore. It may appear strange that the man actually controls uncontrollable events, but remember that both the man and the wolf are parts of the plant. The controller, in this case, is designed to control the movements of the man.

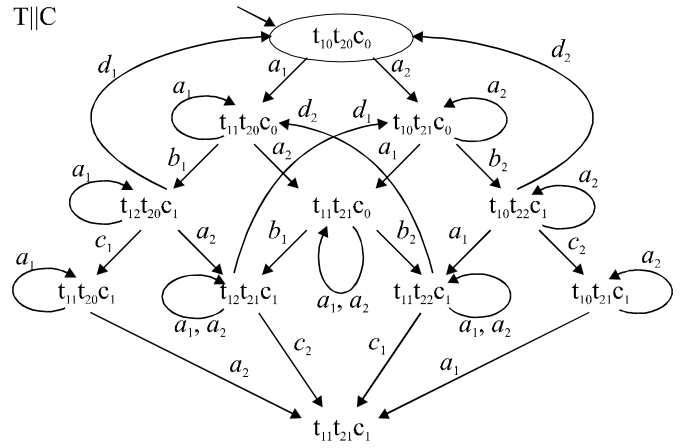
Now, this setting actually gives us a much shorter way to get across the river. The man simply rows the goat across, returns for the cabbage, rows it across and then waits for the wolf to swim across. The synthesised supervisor is shown below.

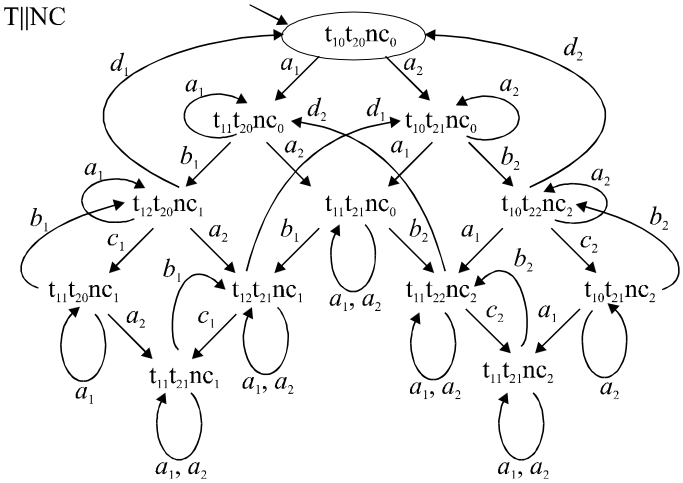


Solution to exercise 7.2



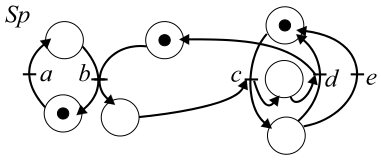
Solution to exercise 7.3



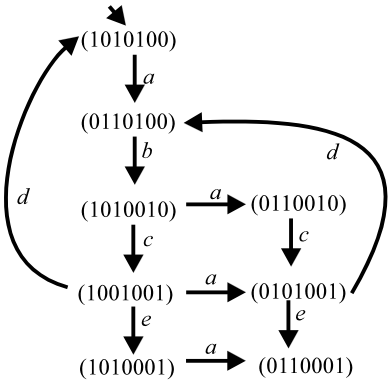


As we can see $T \parallel C$ is blocking while $T \parallel NC$ is non-blocking. Thus, $T \parallel C$ is not a very useful supervisor (how many blocking states are there in $T \parallel C$?).

Solution to exercise 7.4 The plant is $P = R \parallel M$, and the total specification is $Sp = P \parallel Spec$ with the initial state marked and no forbidden states.



The corresponding state-machine (the reachability graph) is shown below.

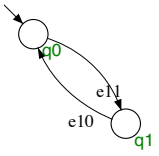


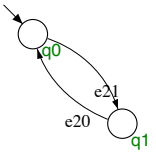
Observe that after an a event then P can always execute the uncontrollable event b, but in Sp there are states reached by a from which b cannot be executed. These states must

not be reached and must thus be removed to give a useful supervisor. This results in a state-machine that is controllable but also blocking. Thus, additional states must be removed. Removing this single state does not affect controllability so that the minimally restrictive supervisor is the result.

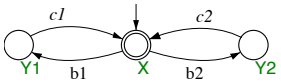
Solution to exercise 7.5 The following automata solve the problems.

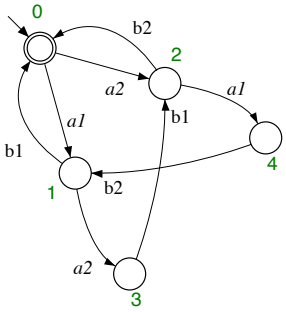
a)

$M_1 =$ 

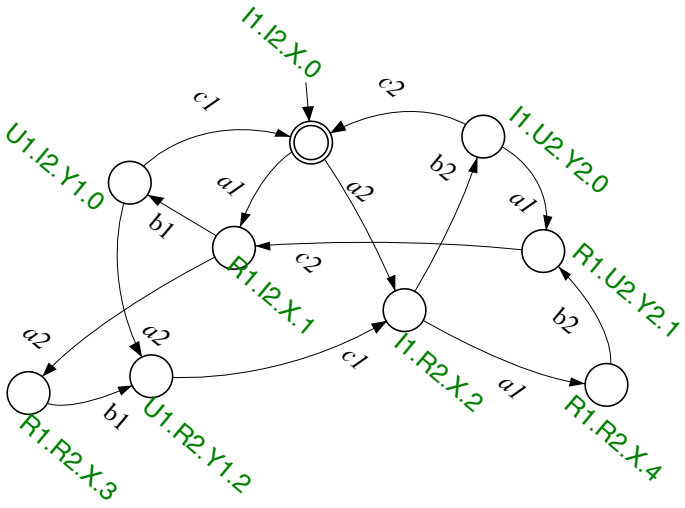
$M_2 =$ 

b)

$K_i =$ 

$K_{ii} =$ 

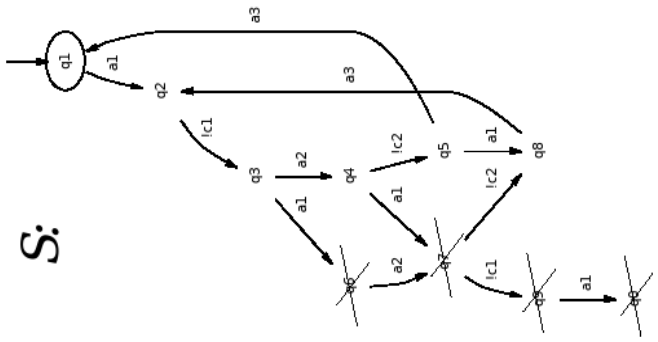
c) The supervisor is simply synthesised by synchronising the automata.



Solution to exercise 7.6

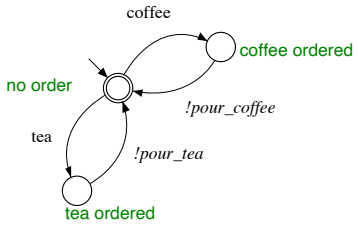
a) The synchronisation of P and Sp result in an automaton that looks exactly like P, except that only the initial state is marked. Of course, this means that the specification could just as well have been expressed as a set of marked (and forbidden) states in the process. This, by definition, means that the specification is static.

b) Since P is blocking, some states must be forbidden, more specifically $q0$ and $q9$. This, in turn, makes $q7$ uncontrollable, since $c1$ is uncontrollable. That is, also $q7$ must be forbidden which makes $q6$ blocking and therefore forbidden. Then we're done.

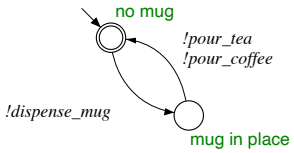


Solution to exercise 7.7

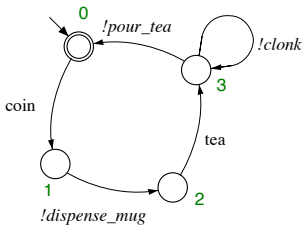
a)



b)



c)

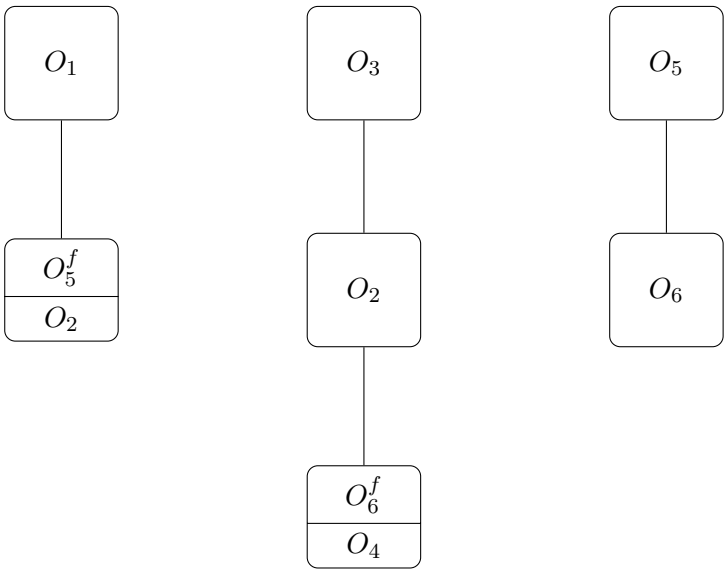


The event *coffee* is in the alphabet of the supervisor (disabled in all states). Unfortunately, it is not possible to allow the users to order coffee as it may cause them the inconvenience of being served tea which would violate the specification.

Chapter 8

Extended Models

Exercise 8.1 (Sequence of Operations)**.** A number of non repeated operations need to be coordinated. Generally, an operation O_k starts when the event s_k occurs and it is completed when the event c_k is fired. Six operations $O_1, ...Q_6$ are going to be executed. In principle all operations can be executed concurrently, except for a couple of restrictions between the operations that are shown in the figure below. The operation sequences are executed from top to bottom, but additional preconditions are included above on some of the operation names (O_k^f means final state for operation O_k). This implies that for instance O_2 in the left sequence must wait until both O_1 and O_5 have been completed. Furthermore, the middle sequence specifies that O_2 also must wait until O_3 has been completed.



a) Formulate a model by EFAs that specify a coordination between the different operations such that the given operation restrictions are satisfied. Include one EFA model O_i for each operation, where appropriate guards are introduced to represent the given restrictions. The total behavior is specified by the synchronized system $O_1 \parallel O_2 \parallel ... \parallel O_6$.

b) Formulate an alternative modular model, composed of a number of automata synchronized by common events, that specify the same behavior as in a).

c) Formulate a Petri net that specify the same behavior as in a) and b).

d) Show that the final state, where all operations have been completed, is reachable, by giving one string of events that is included in the language generated by the given models.

Exercise 8.2 (Synchronizing EFAs)**.** Consider in the Lecture Notes on p.122 the EFAs and corresponding automata models in Fig. 8.2 a) and b), respectively.

a) Generate an automaton for the synchronized EFAs and the corresponding automata.

b) Compare the results and discuss similarities and differences.

Exercise 8.3 (Markov Chain of a Machine)**.** A machine has one operating state q_1 and one broken (idle) state q_2. Assume that the conditional transition probability to transit from operating to broken state is α and the corresponding conditional transition probability from broken to working state is β.

a) Draw a state transition diagram for this discrete-time Markov chain.

b) Calculate the state probability after k discrete time periods, i.e. $p(t_k)$ when the initial state is q_1 (the machine is working). Assume that $\alpha = 0.1$ and $\beta = 0.4$.

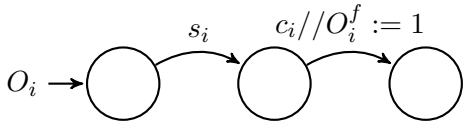
c) Calculate the stationary state probability for arbitrary α and β. Compare with the solution in b) when $k \to \infty$. What happens when $\alpha = \beta$? Explain why!

Exercise 8.4 (Queuing Systems)**.** Derive (8.15) and (8.16) in the Lecture Notes.

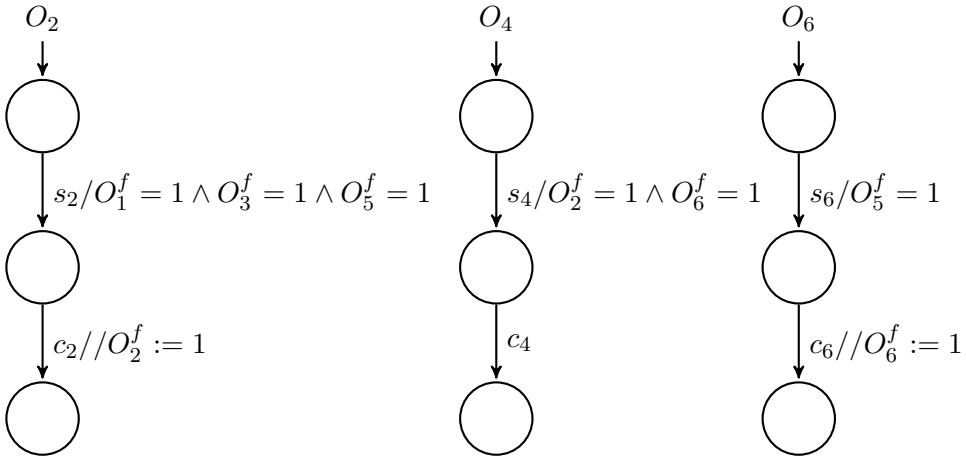
Solutions

Solution to exercise 8.1

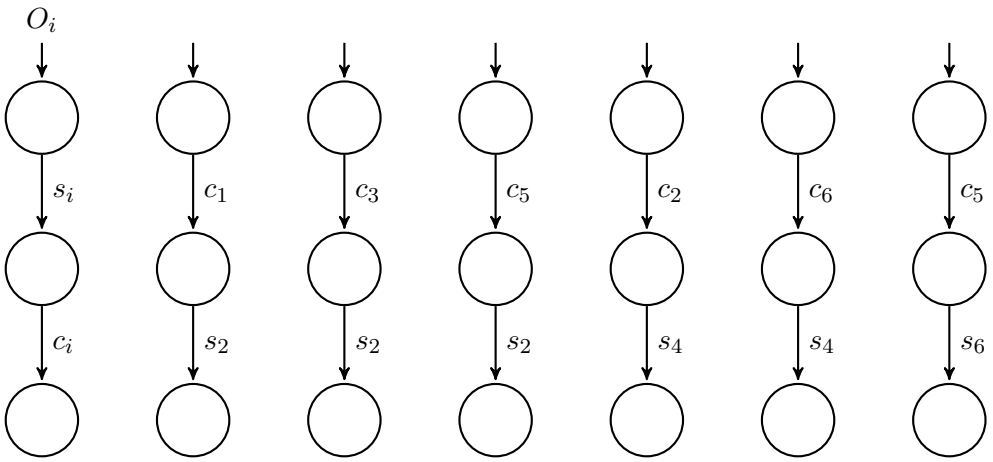
a) The EFA model for $i = 1, 3, 5$ is:



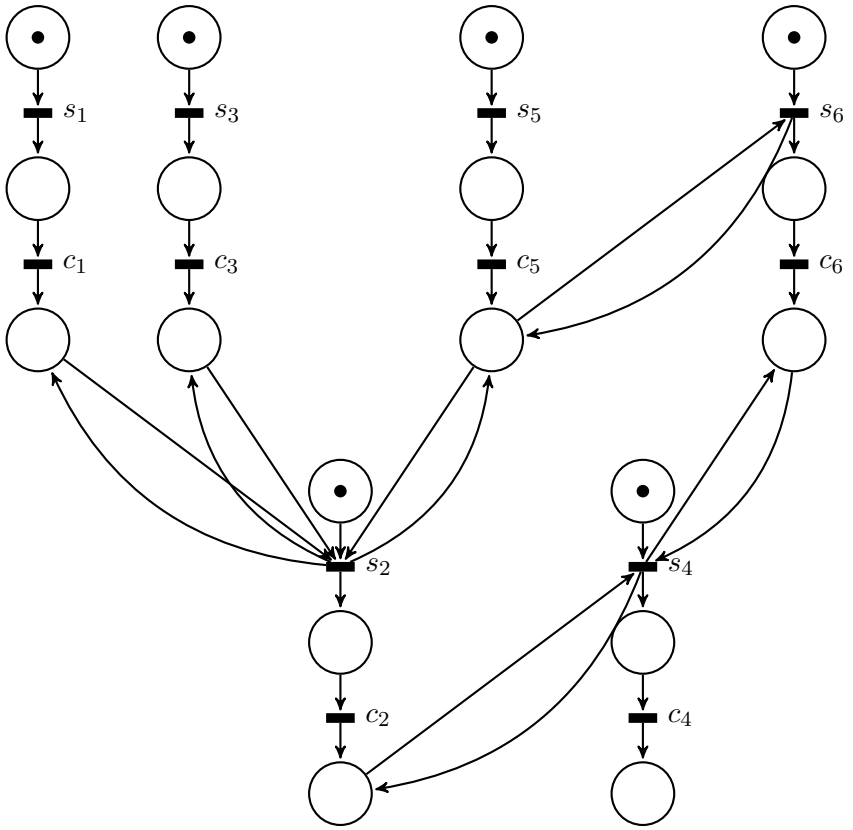
And for the remaining:



b) We formulate a model for each operation $O_i, i = 1, ..., 6$ and six models for specifying the sequence of operations:



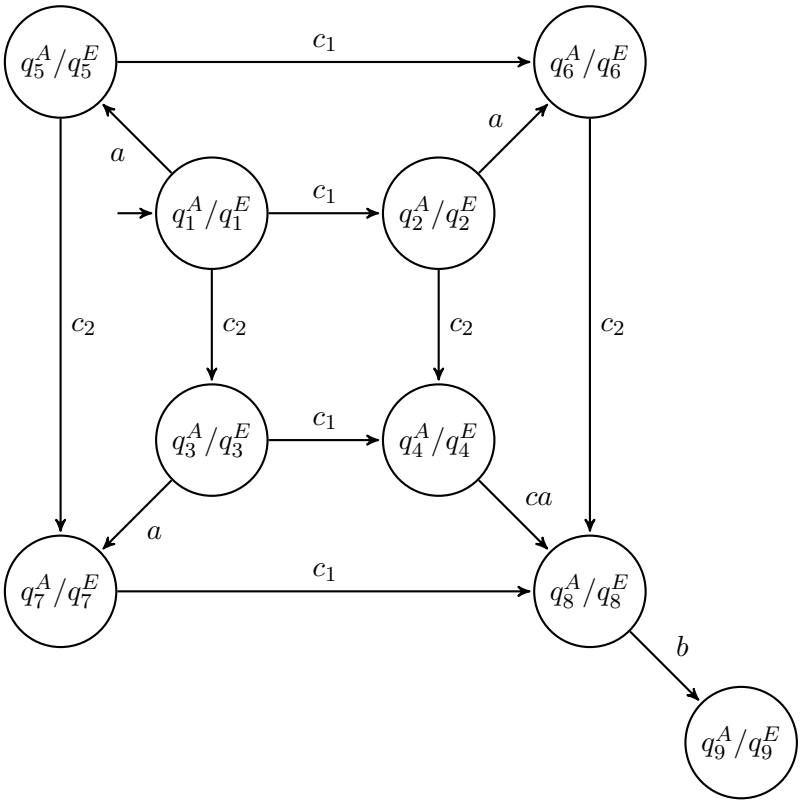
c) The corresponding Petri net is:



d) For instance $s_1c_1s_3c_3s_5c_5s_6c_6s_2c_2s_4c_4$.

Solution to exercise 8.2

a) The synchronized systems are:
$A = A_1 \parallel A_2 \parallel A_3 \parallel A_4$
$E = E_1 \parallel E_2 \parallel E_3 \parallel E_4$

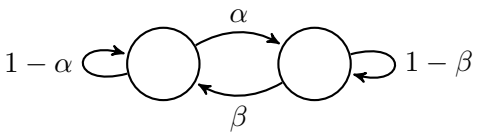


$$
\begin{array}{ll}
q_1^A = \langle q_1^{A_1}, q_1^{A_2}, q_1^{A_3}, q_1^{A_4} \rangle & q_1^E = \langle q_1^{E_1}, q_1^{E_2}, q_1^{E_3}, q_1^{E_4}, \overline{\sigma}_1, \overline{\sigma}_2, \overline{\sigma}_3 \rangle \\
q_2^A = \langle q_2^{A_1}, q_1^{A_2}, q_1^{A_3}, q_1^{A_4} \rangle & q_2^E = \langle q_2^{E_1}, q_1^{E_2}, q_1^{E_3}, q_1^{E_4}, \sigma_1, \overline{\sigma}_2, \overline{\sigma}_3 \rangle \\
\vdots & \\
q_8^A = \langle q_2^{A_1}, q_2^{A_2}, q_2^{A_3}, q_1^{A_4} \rangle & q_8^E = \langle q_2^{E_1}, q_2^{E_2}, q_2^{E_3}, q_1^{E_4}, \sigma_1, \sigma_2, \overline{\sigma}_3 \rangle \\
q_9^A = \langle q_2^{A_1}, q_2^{A_2}, q_2^{A_3}, q_2^{A_4} \rangle & q_9^E = \langle q_2^{E_1}, q_2^{E_2}, q_2^{E_3}, q_2^{E_4}, \sigma_1, \sigma_2, \sigma_3 \rangle
\end{array}
$$

b) The reachable states for the synchronized systems are the same. However, the EFAs also include explicit variables, which means that the number of locations in E_1 and E_2 are only 2, but 3 states in the corresponding automata A_1 and A_2 that model the change of σ_1 and σ_2.

Solution to exercise 8.3

a) The transition diagram:



$$P = \begin{bmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{bmatrix}$$

b) With $\alpha = 0.1$ and $\beta = 0.4$:

$$
\begin{aligned}
\begin{bmatrix} p_1(t_{k+1}) & p_2(t_{k+1}) \end{bmatrix} &= \begin{bmatrix} p_1(t_k) & p_2(t_k) \end{bmatrix} \begin{bmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{bmatrix} \\
&= \begin{bmatrix} 0.9p_1(t_k) + 0.4p_2(t_k) & 0.1p_1(t_k) + 0.6p_2(t_k) \end{bmatrix}
\end{aligned}
$$

And with

$$p_2(t) = 1 - p_1(t)$$

this leads to

$$\begin{aligned} \Longrightarrow \qquad p_1(t_{k+1}) &= 0.9p_1(t_k) + 0.4 - 0.4p_1(t_k) \\ &= 0.5p_1(t_k) + 0.4 \end{aligned}$$

Setting $p_1(0) = 1$ we get

$$\begin{aligned} p_1(0) &= 1 \\ p_1(t_1) &= 0.5p_1(0) + 0.4 \\ p_1(t_2) &= 0.5^2 p_1(0) + 0.5 \cdot 0.4 + 0.4 \\ p_1(t_3) &= 0.5^3 p_1(0) + (0.5^2 + 0.5 + 1) \cdot 0.4 \\ p_1(t_k) &= 0.5^k p_1(0) + \sum_{i=0}^{k-1} 0.5^i \cdot 0.4 \\ \lim_{k\to\infty} p_1(t_k) &= \sum_{i=0}^{\infty} 0.5^i \cdot 0.4 \\ &= \frac{0.4}{1 - 0.5} = 0.8 \end{aligned}$$

and

$$p_2(t_k) = 1 - p_1(t_k) = 0.2$$

c) Once again, we set $p_2 = 1 - p_1$, which leads to

$$\begin{aligned} \begin{bmatrix} p_1 & 1 - p_1 \end{bmatrix} &= \begin{bmatrix} p_1 & 1 - p_1 \end{bmatrix} \begin{bmatrix} 1 - \alpha & \alpha \\ \beta & 1 - \beta \end{bmatrix} \\ &= \begin{bmatrix} (1 - \alpha)p_1 + \beta(1 - p_1) & \alpha p_1 + (1 - \beta)(1 - p_1) \end{bmatrix} \end{aligned}$$

for the stationary state. Now, we get

$$\begin{aligned} p_1 &= (1 - \alpha)p_1 + \beta - \beta p_1 \\ \Leftrightarrow (\alpha + \beta)p_1 &= \beta \\ \Leftrightarrow p_1 &= \frac{\beta}{(\alpha + \beta)} \end{aligned}$$

and

$$p_2 = 1 - p_1 = 1 - \frac{\beta}{(\alpha + \beta)} = \frac{\alpha}{(\alpha + \beta)}$$

To compare to b), we set $\alpha = 0.1$ and $\beta = 0.4$, which also results in $p_1 = 0.8$ and $p_2 = 0.2$.

If $\alpha = \beta$

$$\Longrightarrow \; p_1 = \frac{\alpha}{2\alpha} = 0.5 = p_2$$

The same probability to move to the other state means that the stationary state probability will be the same.

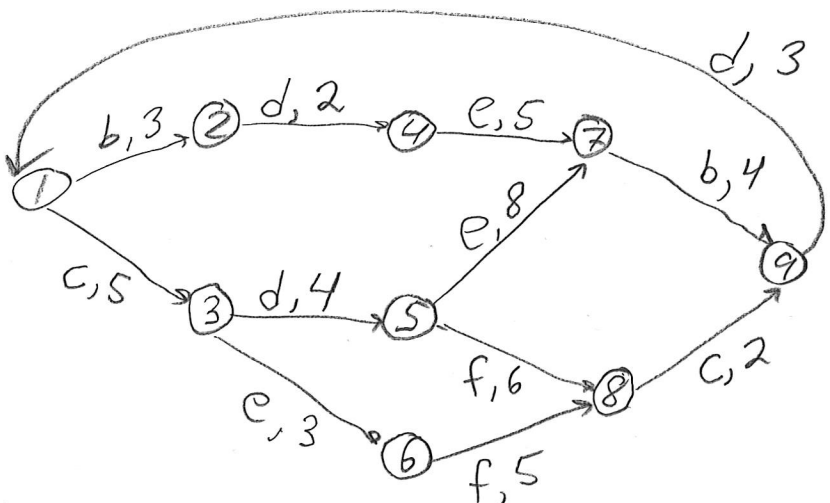
Bibliography

[Fab04] Martin Fabian. Discrete event systems – lecture notes. Technical report, Chalmers University of Technology, 2004.

[Giu92] Alessandro Giua. *Petri Nets as Discrete Event Models for Supervisory Control*. PhD thesis, 1992.

[Won99] W. Murray Wonham. Notes on control of discrete event systems. Technical report, University of Toronto, 1999.

Q-learning example



$$J^*(x) = \max_{a \in \Sigma(x)} [S(x,a) + \gamma J^*(\delta(x,a))]$$

$$J^*(1) = \max \{ 3 + \gamma J^*(2), 5 + \gamma J^*(3) \}$$

$$J^*(2) = 2 + \gamma J^*(4) = 2 + \gamma (5 + \gamma J^*(7)) =$$

$$= 2 + 5\gamma + \gamma^2 (4 + \gamma J^*(9)) =$$

$$= 2 + 5\gamma + 4\gamma^2 + \gamma^3 (3 + \gamma J^*(1)) =$$

$$= 2 + 5\gamma + 4\gamma^2 + 3\gamma^3 + \gamma^4 J^*(1)$$

$$J^*(7) = 4 + 3\gamma + \gamma^2 J^*(1)$$

$$J^*(8) = 2 + 3\gamma + \gamma^2 J^*(1)$$

$$J^*(6) = 5 + 2\gamma + 3\gamma^2 + \gamma^3 J^*(1)$$

$$J^*(5) = \max \{ 8 + \gamma J^*(7), 6 + \gamma J^*(8) \} =$$

$$= \max \{ 8 + 4\gamma + 3\gamma^2 + \gamma^3 J^*(1), 6 + 2\gamma + 3\gamma^2 + \gamma^3 J^*(1) \}$$

$$= 8 + 4\gamma + 3\gamma^2 + \gamma^3 J^*(1)$$

$$J^*(3) = \max \{ 4 + \gamma J^*(5), 3 + \gamma J^*(6) \} =$$

$$= \max \{ 4 + 8\gamma + 4\gamma^2 + 3\gamma^3 + \gamma^4 J^*(1), 3 + 5\gamma + 2\gamma^2 + 3\gamma^3 + \gamma^4 J^*(1) \}$$

$$= 4 + 8\gamma + 4\gamma^2 + 3\gamma^3 + \gamma^4 J^*(1)$$

$$= 4 + 8\gamma + 4\gamma^2 + 3\gamma^3 + \gamma^4 J^*(1)$$

$$J^*(1) = \max\{3 + 2\gamma + 5\gamma^2 + 4\gamma^3 + 3\gamma^4 + \gamma^5 J^*(1),$$
$$5 + 4\gamma + 8\gamma^2 + 4\gamma^3 + 3\gamma^4 + \gamma^5 J^*(1)\} =$$

$$= 5 + 4\gamma + 8\gamma^2 + 4\gamma^3 + 3\gamma^4 + \gamma^5 J^*(1)$$

$$J^*(1) = \frac{5 + 4\gamma + 8\gamma^2 + 4\gamma^3 + 3\gamma^4}{1 - \gamma^5}$$

$Q(x,a) = g(x,a) + \gamma J^*(\delta(x,a))$ $\quad \mu(x) = \arg\max_{a \in \Sigma(x)} Q(x,a)$

$Q(1,b) = 3 + \gamma J^*(2)$ $\quad Q(1,c) = 5 + \gamma J^*(3)$

$\mu(1) = \arg\max_{a \in \{b,c\}} Q(1,a) = c$

since $Q(1,c) = 5 + \gamma J^*(3) =$

$= 5 + 4\gamma + 8\gamma^2 + 4\gamma^3 + 3\gamma^4 + \gamma^5 J^*(1) >$

$Q(1,b) = 3 + 2\gamma + 5\gamma^2 + 4\gamma^3 + 3\gamma^4 + \gamma^5 J^*(1)$

Thus $Q(1,c) > Q(1,b)$ for any $\gamma \in [0,1]$

In the same way

$\mu(3) = \arg\max_{a \in \{d,e\}} Q(3,a) = d$ since $Q(3,d) > Q(3,e)$

$\mu(5) = \arg\max_{a \in \{e,f\}} Q(5,a) = e$ since $Q(5,e) > Q(5,f)$