C.A. Middelburg and M.A. Reniers

Introduction to Process Theory

March 31, 2004

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Capaciteitsgroep Informatica
Eindhoven

Contents

1. Transition systems

The notion of transition system can be considered to be the fundamental notion for the description of process behaviour. This chapter is meant to acquire a good insight into this notion and its relevance for the description of process behaviour. First of all, we explain informally what transition systems are and give some simple examples of their use in describing process behaviour (Section 1.1). After that, we define the notion of transition system in a mathematically precise way (Section 1.2). For a better understanding, we next investigate the connections between the notion of transition system and the familiar notions of program (Section 1.3) and automaton (Section 1.4). Then, we discuss several notions of equivalence on transition systems (Section 1.5). Those equivalences are useful because they allow us to abstract from details of transition systems that we often want to ignore. Finally, we look at a means of expressing properties of processes in terms of a logic called Hennessy-Milner logic (Section 1.6).

1.1 Informal explanation

Transition systems are often considered to be the same as automata. Both consist of states and labeled transitions between states. The main difference is that automata are primarily regarded as abstract machines to recognize certain languages and transition systems are primarily regarded as a means to describe the behaviour of interacting processes. In the case of transition systems, the intuition is that a transition is a state change caused by performing the action labeling the transition. A transition from a state s to a state s' labeled by an action a is usually written $s \xrightarrow{a} s'$. This can be read as "the system is capable of changing its state from s into s' by performing action a". Let us give an example to illustrate that it is quite natural to look at real-life computer-based systems as systems that change their state by performing actions.

Example 1.1.1 (Simple telephone system). We consider a simple telephone system. In this telephone system each telephone is provided with a process, called its basic call process, to establish and maintain connections with other telephones. Actions of this process include receiving an off-hook or on-hook signal from the telephone, receiving a dialed number from the telephone, sending a signal to start or to stop emitting a dial tone, ring tone or ring-back tone to the telephone, and receiving an alert signal from another telephone – indicating an incoming call. Suppose that a basic call process is in the idling state. In this state, it can change its state to the initial dialing state by receiving an off-hook signal from the telephone. Alternatively, it can change its state to the initial ringing state by receiving an alert signal from another telephone. In the initial dialing state, it can change its state to another dialing state by sending a signal to start emitting a dial tone to the telephone. In the initial ringing state, it can change its state to another ringing state by sending a signal to start emitting a ring tone to the telephone. And so forth.

Transition systems have been devised as a means to describe the behaviour of systems that have only discrete state changes. Despite this underlying purpose of transition systems, they can deal with continuous state changes as well. However, such use of transition systems will not be treated in this book. Instead, we focus on acquiring a good insight into the basic concepts of transition systems. That is not for pedagogical reasons alone. Systems that have only discrete state changes are still of utmost importance in the practice of developing computer-based systems and will remain so for a long time. Here are a couple of examples of the use of transition systems in describing the behaviour of systems with discrete state changes.

Example 1.1.2 (Bounded counter). We first consider a very simple system, viz. a bounded counter. A bounded counter can perform increments of its value by 1 till a certain value k is reached and can perform decrements of its value by 1 till the value 0 is reached. As states of a bounded counter, we have the natural numbers 0 to k. State i is the state in which the value of the counter is i. As actions, we have inc (increment) and dec (decrement). As transitions of a bounded counter, we have the following:

- for each state i that is less than k, a transition from state i to state $i+1$ labeled with the action inc, written $i \xrightarrow{\mathsf{inc}} i+1$;
- for each state i that is less than k, a transition from state $i+1$ to state i labeled with the action dec, written $i+1 \xrightarrow{\mathsf{dec}} i$.

If the number of states and transitions is small, a transition system can easily be represented graphically. The transition system describing the behaviour of the bounded counter is represented graphically in Figure 1.1 for the case where $k = 3$. In the graphical representation of transition systems, circles or ovals are used to denote the states of the transition system. In many cases, for easy reference, the identity of the state is written inside the circle or oval. A transition $s \xrightarrow{a} s'$ is represented by an arrow, labelled by the action name a, from the circle representing the state s to the circle representing the state s'. Notice that the bounded counter has a finite number of states and a finite number of transitions. Furthermore, the bounded counter will never become

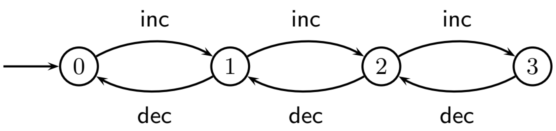


Fig. 1.1. Transition system for the bounded counter

inactive, i.e. it will never reach a state from which no transition is possible. Thus, the finiteness of the bounded counter does not keep the counter from making an infinite number of transitions. The bounded counter can easily

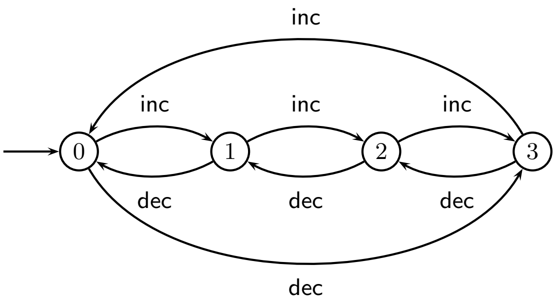


Fig. 1.2. Transition system for the modulo counter

be adapted to become a counter modulo $k + 1$, i.e. a counter whose value becomes 0 by performing an increment by 1 when its value is k and whose value becomes k by performing a decrement by 1 when its value is 0. We have the same states and actions as before and we have two additional transitions (see Figure 1.2):

- a transition from state k to state 0 labeled with the action inc, written $k \xrightarrow{\mathsf{inc}} 0$;
- a transition from state 0 to state k labeled with the action dec, written $0 \xrightarrow{\mathsf{dec}} k$.

Example 1.1.3 (Bounded buffer). We next consider another simple system, viz. a bounded buffer. A bounded buffer can add new data to the sequence of data that it keeps if the capacity l of the buffer is not exceeded, i.e. if the length of the sequence of data that it keeps is not greater than l. As long as it keeps data, it can remove the data that it keeps – in the order in which they were added. As states of a bounded buffer, we have the sequences of data of which the length is not greater than l. State σ is the state in which the sequence of data σ is kept in the buffer. As actions, we have $\mathsf{add}(d)$ (add d) and $\mathsf{rem}(d)$ (remove d) for each datum d. As transitions of a bounded buffer, we have the following:

- for each datum d and each state σ that has a length less than l, a transition from state σ to state $d\,\sigma$ labeled with the action $\mathsf{add}(d)$, written $\sigma \xrightarrow{\mathsf{add}(d)} d\,\sigma$;
- for each datum d and each state $\sigma\,d$, a transition from state $\sigma\,d$ to state σ labeled with the action $\mathsf{rem}(d)$, written $\sigma\,d \xrightarrow{\mathsf{rem}(d)} \sigma$.

The transition system describing the behaviour of the bounded buffer is represented graphically in Figure 1.3 for the case where $l = 2$ and the only data involved are the natural numbers 0 and 1. Although it has a finite capacity,

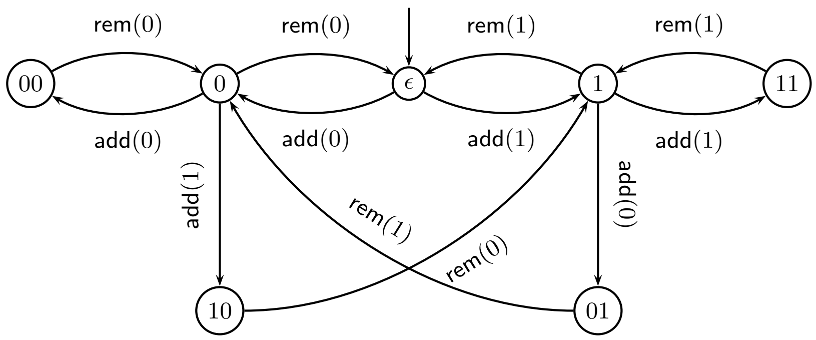


Fig. 1.3. Transition system for the bounded buffer

the bounded buffer will have an infinite number of states and an infinite number of transitions in the case where the number of data involved is infinite.

The bounded buffer can easily be adapted to become unreliable, e.g. to get into an error state by adding a datum when it is full. We have one additional state, say err, no additional actions, and the following additional transitions:

- for each datum d and each state σ that has a length equal to l, a transition from state σ to state err labeled with the action $\mathsf{add}(d)$, written $\sigma \xrightarrow{\mathsf{add}(d)} \mathsf{err}$.

Notice that this unreliable bounded buffer will become inactive when it reaches the state err. Thus, the additional feature of this buffer may keep it from making an infinite number of transitions.

It is usual to designate one of the states of a transition system as its initial state. At the start-up of a system, i.e. before it has performed any action, the system is considered to be in its initial state. The expected initial states of the bounded counter from Example 1.1.2 and the bounded buffer from Example 1.1.3 are 0 and ϵ (the empty sequence), respectively. The initial state of a transition system is indicated by an unconnected incoming arrow.

Although bounded counters and buffers arise frequently as basic components in computer-based systems, they are not regarded as typical examples of real-life computer-based systems. In the following example, we consider a simplified version of a small real-life computer-based system, viz. a calculator.

Example 1.1.4 (Calculator). We consider a calculator that can perform simple arithmetical operations on integers. It can only perform addition, subtraction, multiplication and division on integers between a certain values, say min and max. As states of the calculator, we have pairs (i, o), where $min \leq i \leq max$ or $i = *$ and $o \in \{\mathsf{add}, \mathsf{sub}, \mathsf{mul}, \mathsf{div}, \mathsf{eq}, \mathsf{clr}, *\}$. State (i, o) is roughly the state in which the result of the preceding calculations is i and the operator that must be applied next is o. If $o = *$, the operator that must be applied next is not available; and if in addition $i = *$, the result of the preceding calculations is not available either. As initial state, we have the pair $(*, *)$. As actions, we have $\mathsf{rd}(i)$ (read operand i) and $\mathsf{wr}(i)$ (write result i), both for $min \leq i \leq max$, and $\mathsf{rd}(o)$ (read operator o), for $o \in \{\mathsf{add}, \mathsf{sub}, \mathsf{mul}, \mathsf{div}, \mathsf{eq}, \mathsf{clr}\}$. As transitions of the calculator, we have the following:

- for each i with $min \leq i \leq max$:
  - a transition $(*, *) \xrightarrow{\mathsf{rd}(i)} (i, *)$,
  - a transition $(i, \mathsf{clr}) \xrightarrow{\mathsf{wr}(0)} (*, *)$,
  - a transition $(i, \mathsf{eq}) \xrightarrow{\mathsf{wr}(i)} (i, *)$;
- for each i with $min \leq i \leq max$ and $o \in \{\mathsf{add}, \mathsf{sub}, \mathsf{mul}, \mathsf{div}, \mathsf{eq}, \mathsf{clr}\}$:
  - a transition $(i, *) \xrightarrow{\mathsf{rd}(o)} (i, o)$;
- for each i with $min \leq i \leq max$ and j with $min \leq j \leq max$:
  - a transition $(i, \mathsf{add}) \xrightarrow{\mathsf{rd}(j)} (i + j, *)$ if $min \leq i + j \leq max$,
  - a transition $(i, \mathsf{sub}) \xrightarrow{\mathsf{rd}(j)} (i - j, *)$ if $min \leq i - j \leq max$,
  - a transition $(i, \mathsf{mul}) \xrightarrow{\mathsf{rd}(j)} (i \cdot j, *)$ if $min \leq i \cdot j \leq max$,
  - a transition $(i, \mathsf{div}) \xrightarrow{\mathsf{rd}(j)} (i \div j, *)$ if $min \leq i \div j \leq max$ and $j \neq 0$.

The transition system describing the behaviour of the calculator is represented graphically in Figure 1.4 for the case where $min = 0$ and $max = 1$. Although the extremely small range of integers makes this case actually useless, it turns out to be difficult to represent the transition system graphically. The textual description given above is still intelligible. However, it is questionable whether this would be the case for a more realistic calculator.

Examples like Example 1.1.4 indicate that in the case of real-life systems we probably need a way to describe process behaviour more concisely than by directly giving a transition system. This is one of the issues treated in the remaining chapters of this book.

Example 1.1.5 (Automatic teller machine). We consider the operation of a simple automatic teller machine (ATM) and we focus on the interaction between the user and the ATM. The user inserts his/her bank card ($\mathsf{insert\ card}$)

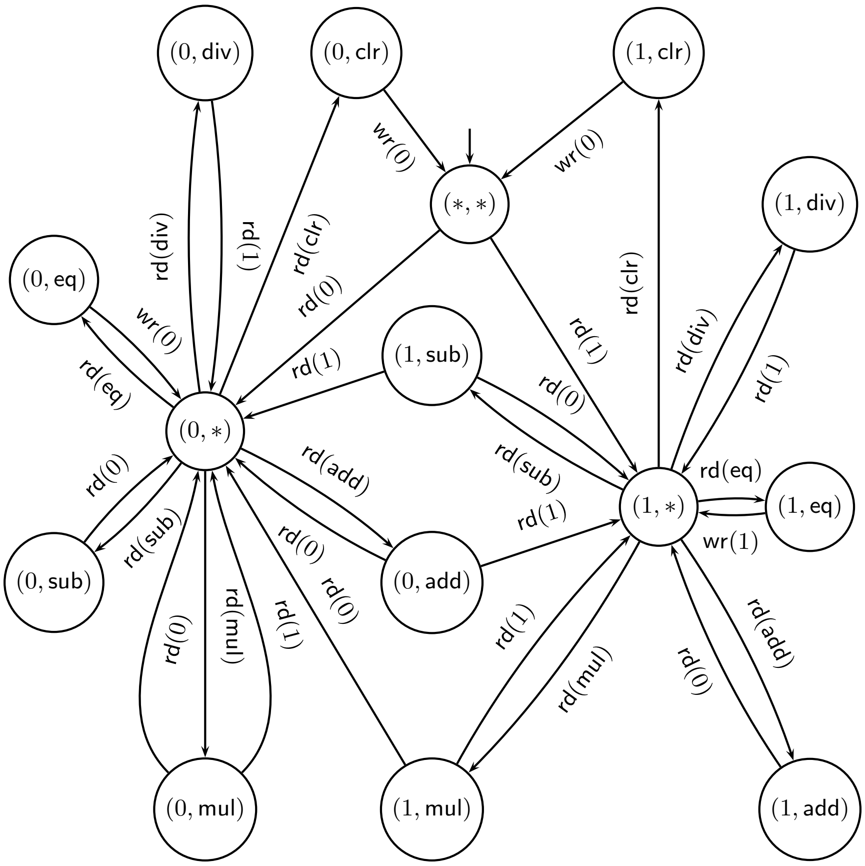


Fig. 1.4. Transition system for the calculator

and enters the Personnel Identification Number (enter PIN). The ATM signals to the user the outcome of some validation procedure (PIN valid or PIN invalid). In case the PIN is invalid, the ATM ejects the bank card (eject card) and returns to the idle state. In case, the PIN is validated, the user enters an amount of cash to be withdrawn (enter amount). After consulting with the bank the ATM signals whether this cash amount is balanced by the account of the user (accept amount) or not accepted (cancel). In the latter case the bank card is ejected. In case of acceptance, the ATM offers the cash (withdrawn) and ejects the bank card. The transition system for this ATM is given in Figure 1.5. Observe that the names of the states are not shown in this figure.

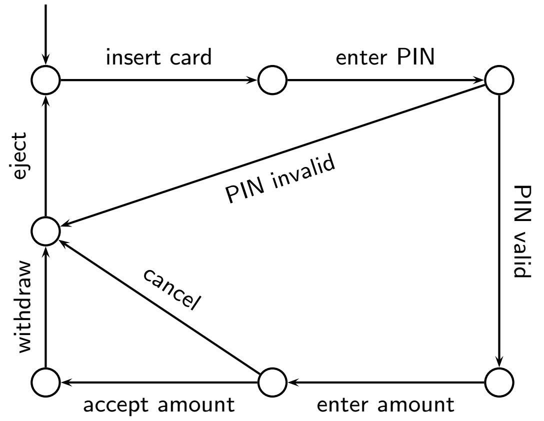


Fig. 1.5. Transition system for the user interface of an ATM

In describing the ATM in Example 1.1.5 we have chosen not to consider the procedure of checking the PIN and the procedure of checking whether the amount of cash is balanced by the bank account in detail. The reason to do so is that in this example our interest is with the user of the ATM. For the user of the ATM only the outcome of such procedures is of interest.

Exercise 1.1.1 (Unreliable bounded buffer). Give a transition system for the unreliable bounded buffer as discussed in Example 1.1.3.

Exercise 1.1.2 (One-place buffer). Give a transition system for a one-place buffer that can store values from the set $D = \{0, 1, 2\}$. The actions are $\mathsf{put}(d)$ for putting a datum $d \in D$ into the buffer and $\mathsf{take}(d)$ for taking a datum $d \in D$ from the buffer.

Exercise 1.1.3 (ATM with stop button). Adapt the transition system for the simple ATM from Example 1.1.5 in such a way that the user can push a stop button (stop) resulting in the abortion of the transaction. Of course the bank card (if already inserted) should in all cases be returned to the user. Pay attention to the situation that the transaction is confirmed by the ATM (probably this means that the money is already subtracted from the account of the user).

Exercise 1.1.4 (ATM with PIN-retry). Adapt the transition system for the simple ATM from Example 1.1.5 in such a way that upon invalidation of the PIN, the user can try again to enter a valid PIN. It should be the case that after entering three invalid PINs the bank card is not ejected anymore.

1.2 Formal definition

With the previous section, we have prepared the way for the formal definition of the notion of transition system.

Definition 1.2.1. A *transition system* T is a quintuple $(S, A, \rightarrow, \downarrow, s_0)$ where

- S is a set of *states*;
- A is a set of *actions*;
- $\rightarrow \subseteq S \times A \times S$ is a set of *transitions*;
- $\downarrow \subseteq S$ is a set of *successfully terminating states*;
- $s_0 \in S$ is the *initial state*.

If S and A are finite, T is called a *finite* transition system. We write $s \xrightarrow{a} s'$ and $s\downarrow$ instead of $(s, a, s') \in \rightarrow$ and $s \in\downarrow$, respectively. We write $\mathrm{act}(T)$ for A, i.e. the set of actions of T. The set $\twoheadrightarrow \subseteq S \times A^* \times S$ of *generalized transitions* of T is the smallest subset of $S \times A^* \times S$ satisfying:

- $s \overset{\epsilon}{\twoheadrightarrow} s$ for each $s \in S$;
- if $s \xrightarrow{a} s'$, then $s \overset{a}{\twoheadrightarrow} s'$;
- if $s \overset{\sigma}{\twoheadrightarrow} s'$ and $s' \overset{\sigma'}{\twoheadrightarrow} s''$, then $s \overset{\sigma\,\sigma'}{\twoheadrightarrow} s''$.

The notations A^*, the set of all sequences over A, and $\sigma\,\sigma'$, the concatenation of sequences, are defined formally in Appendix A.3.

A state $s \in S$ is called a *reachable* state of T if there is a $\sigma \in A^*$ such that $s_0 \overset{\sigma}{\twoheadrightarrow} s$. The set of all reachable states of a transition system T is denoted $\mathrm{reach}(T)$. A state $s \in S$ is called a *terminal* state of T if not $s\downarrow$ and there are no $a \in A$ and $s' \in S$ such that $s \xrightarrow{a} s'$. A state $s \in S$ is called a *deadlock* state of T if s is a reachable state of T and a terminal state of T.

We will now return to some of the transition systems introduced informally in the previous section. By convention, the set of successfully terminating states is considered to be empty, unless stated otherwise.

Example 1.2.1 (Bounded counter). We look again at the bounded counter from Example 1.1.2. Formally, the behaviour of a bounded counter with bound k is described by the transition system $(S, A, \rightarrow, s_0)$ where

$$\begin{aligned}
S &= \{i \in \mathbb{N} \mid i \leq k\},\\
A &= \{\mathsf{inc}, \mathsf{dec}\},\\
\rightarrow &= \{(i, \mathsf{inc}, i+1) \mid i \in \mathbb{N}, i < k\} \cup \{(i+1, \mathsf{dec}, i) \mid i \in \mathbb{N}, i < k\},\\
s_0 &= 0.
\end{aligned}$$

All states of this finite transition system are reachable. It does not have terminal states, and hence also no deadlock states.

Example 1.2.2 (Unreliable bounded buffer). We also look again at the unreliable bounded buffer from Example 1.1.3. Formally, the behaviour of the unreliable bounded buffer with capacity l is described by the transition system $(S, A, \rightarrow, s_0)$ where

$$
\begin{aligned}
S &= \{\sigma \in D^* \mid |\sigma| \leq l\} \cup \{\mathsf{err}\},\\
A &= \{\mathsf{add}(d) \mid d \in D\} \cup \{\mathsf{rem}(d) \mid d \in D\},\\
\rightarrow &= \{(\sigma, \mathsf{add}(d), d\,\sigma) \mid \sigma \in D^*, |\sigma| < l\}\\
&\quad \cup \{(\sigma, \mathsf{add}(d), \mathsf{err}) \mid \sigma \in D^*, |\sigma| = l\}\\
&\quad \cup \{(\sigma\, d, \mathsf{rem}(d), \sigma) \mid \sigma \in D^*, |\sigma| < l\},\\
s_0 &= \epsilon.
\end{aligned}
$$

The notation $|\sigma|$, the length of the sequence σ, is defined formally in Appendix A.3. All states of this transition system are reachable. It has one terminal state, viz. err. This state is also a deadlock state.

Henceforth, we will only occasionally introduce transition systems in this formal style. After the informal explanation and formal definition of the notion of transition system, we are now in the position to relate it to the notions of program and automaton in the next two sections.

Exercise 1.2.1 (Traffic light). Give a transition system for a traffic light. The actions are the colours of the traffic light: red, yellow, and green. Initially, the traffic light is red.

Exercise 1.2.2 (Road crossing). Give a transition system for a one-directional road crossing with two traffic lights as presented in Figure 1.6. The actions involved are red_1, yellow_1, green_1, red_2, yellow_2, and green_2. Make

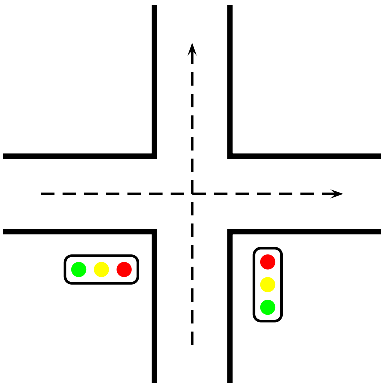


Fig. 1.6. Two traffic lights on a one-directional road crossing

sure that collisions cannot occur, assuming that drivers respect the traffic lights. Initially, both traffic lights are red.

Exercise 1.2.3 (Elevator system). Give a transition system for an elevator system. The elevator system serves 5 floors, numbered 0-4. The elevator starts at floor 0. The actions of the elevator system are up and down representing that the elevator goes one floor up and one floor down respectively.

Exercise 1.2.4 (Stopwatch). Consider the following stopwatch. The stopwatch has two buttons and one display. Initially, the display is empty. As soon as the start button is pushed, the stopwatch starts counting time (in seconds) from zero up. Pushing the stop button results in stopping the counting of seconds. After stopping the counting, on the display the amount of time that has elapsed is displayed.

Use the following actions: start for pushing the start button, stop for pushing the stop button, $\mathsf{display}(s)$ for displaying the elapsed time s (in seconds), and tick for modelling the passage of a second.

Decide for yourself how the stopwatch should react on pushing the start button while counting.

Exercise 1.2.5 (Binary adder). A binary adder takes to two bit strings representing numbers and produces their sum as output. For simplicity, let us assume that we are dealing only with positive integers and that we use an inverse binary representation. A serial adder processes two such numbers $x = a_0a_1 \ldots a_n$ and $b_0b_1 \ldots b_m$ bit by bit, starting at the left hand.

Give a transition system for a binary adder. Assume that two bits are read through the actions $\mathsf{read}(b_1, b_2)$ for $b_1, b_2 = 0, 1$, and that the bits of the sum are produced through the actions $\mathsf{write}(b)$ for $b = 0, 1$.

1.3 Programs and transition systems

For a better understanding of the notion of transition system, we now look into its connections with the familiar notion of program.

The behaviour of a program upon execution can be regarded as a transition system. In doing so, we can abstract from how the actions performed by a program are processed by a machine, and hence from how the values assigned to the program variables are maintained. In that case, we focus on the flow of control. The states of the transition system only serve as the control points of the program and its actions are merely requests to perform actions such as assignments, tests, etc. What we have in view here will be called the behaviour of a program upon *abstract* execution to distinguish it clearly from the behaviour of a program upon execution on a machine, which applies to the processing by a machine of the actions performed by the program. Here is an example of the use of transition systems in describing the behaviour of programs upon abstract execution.

Example 1.3.1 (Factorial). We consider the following PASCAL program to calculate factorials[1]:

```
PROGRAM factorial(input,output);
```

[1] For any natural number $n \in \mathbb{N}$, the factorial of n, denoted $n!$, is defined inductively by $0! = 1$ and $(n + 1)! = (n + 1) \times n!$.

```
VAR i,n,f: 0..maxint;
BEGIN
   read(n);
   i := 0; f := 1;
   WHILE i < n DO
      BEGIN i := i + 1; f := f * i END;
   write(f)
END
```

The behaviour of this program upon abstract execution can be described by a transition system as follows. As states of the factorial program, we have the natural numbers 0 to 7, with 0 as initial state. For easy reference we haven inserted the numbers representing the states also in the PASCAL program as comments.

```
PROGRAM factorial(input,output);
VAR i,n,f: 0..maxint;
BEGIN {0}
   read(n); {1}
   i := 0; {2} f := 1; {3}
   WHILE i < n DO
      BEGIN {4} i := i + 1; {5} f := f * i END; {6}
   write(f) {7}
END
```

The states can be viewed as the values of a "program counter". As actions, we have an action corresponding to each atomic statement[2] of the program as well as each test of the program and its opposite. As transitions, we have the following:

$$
\begin{aligned}
&0 \xrightarrow{\mathsf{read(n)}} 1,\ 1 \xrightarrow{\mathsf{i:=0}} 2,\ 2 \xrightarrow{\mathsf{f:=1}} 3,\\
&3 \xrightarrow{\mathsf{i<n}} 4,\ 4 \xrightarrow{\mathsf{i:=i+1}} 5,\ 5 \xrightarrow{\mathsf{f:=f*i}} 3,\\
&3 \xrightarrow{\mathsf{NOT\,i<n}} 6,\ 6 \xrightarrow{\mathsf{write(f)}} 7.
\end{aligned}
$$

Once the program has performed the statement `write(f)`, it cannot perform any more actions. As the program has reached its end successfully, the state associated, i.e., state 7, is designated as a final state. Hence $7\downarrow$. The transition system for the factorial program is represented graphically in Figure 1.7.

Here is another example.

Example 1.3.2 (Greatest Common Divisor). We consider the following PASCAL program to calculate greatest common divisors[3]:

[2] An atomic statement is a statement that is not to be divided into "smaller" statements.

[3] The greatest common divisor of two positive natural numbers m and n is the greatest natural number k such that k is a divisor of both m and n.

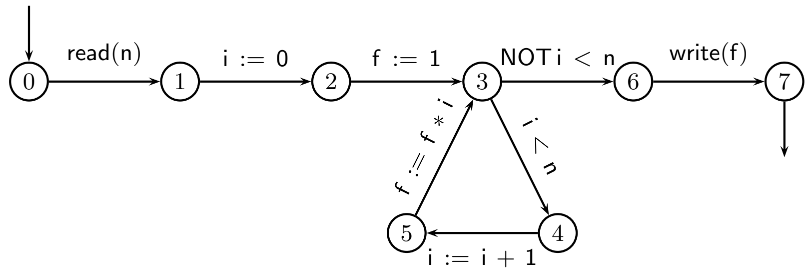


Fig. 1.7. Transition system for the factorial program

```
PROGRAM gcd(input,output);
VAR m,n: 1..maxint;
BEGIN
   read(m); read(n);
   REPEAT
      WHILE m > n DO m := m - n;
      WHILE n > m DO n := n - m
   UNTIL m = n;
   write(m)
END
```

The behaviour of this program upon abstract execution can be described by a transition system as follows. As states of the greatest common divisor program, we have the natural numbers 0 to 8, with 0 as initial state. As actions, we have an action corresponding to each atomic statement of the program as well as each test of the program and its opposite. As transitions, we have the following:

$$\begin{aligned}
&0 \xrightarrow{\mathsf{read(m)}} 1,\ 1 \xrightarrow{\mathsf{read(n)}} 2,\\
&2 \xrightarrow{\mathsf{m > n}} 3,\ 3 \xrightarrow{\mathsf{m := m - n}} 2,\\
&2 \xrightarrow{\mathsf{NOT\, m > n}} 4,\ 4 \xrightarrow{\mathsf{n > m}} 5,\ 5 \xrightarrow{\mathsf{n := n - m}} 4,\ 4 \xrightarrow{\mathsf{NOT\, n > m}} 6,\ 6 \xrightarrow{\mathsf{NOT\, m = n}} 2,\\
&6 \xrightarrow{\mathsf{m = n}} 7,\ 7 \xrightarrow{\mathsf{write(m)}} 8.
\end{aligned}$$

The final state of this program is state 8, hence $8\downarrow$. The transition system for the greatest common divisor program is represented graphically in Figure 1.8.

Notice that the transition systems described in Examples 1.3.1 and 1.3.2 have a single successfully terminating state.

A transition system derived from a program in the way described and illustrated above is reminiscent of a flowchart. However, the underlying idea is that the transition system describes the behaviour of the program upon execution in such a way that it can act concurrently and interact with a machine that processes the actions performed by the program. If it does so,

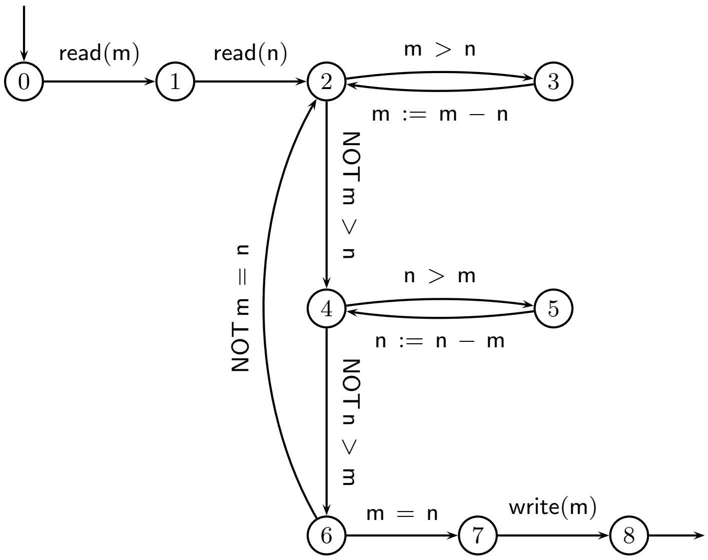


Fig. 1.8. Transition system for the greatest common divisor program

the combined behaviour can be regarded as the behaviour of the program upon execution on a machine. Interaction between processes is one of the issues treated in the remaining chapters of this book. We can also directly give a transition system describing the behaviour of the program upon execution on a machine. In that case, we have to take into account that an assignment changes the value of a program variable, the values of the program variables determine whether a test succeeds, etc. This is illustrated in the following couple of examples, which are concerned with the same programs as the previous two examples.

Example 1.3.3 (Factorial). We consider again the program from Example 1.3.1. The intended behaviour of this program upon execution on a machine can be described by a transition system as follows. As states of the program, we have pairs (l, s), where $l \in \mathbb{N}$ with $0 \leq l \leq 7$ and $s = (i, n, f)$ with $i, n, f \in \{i \in \mathbb{N} \mid i \leq \mathit{maxint}\} \cup \{*\}$. These states can be viewed as follows: l is the value of the program counter and $s = (i, n, f)$ is the storage that keeps the values of the program variables i, n, and f in that order. The special value $*$ is used to indicate that no value has yet been assigned to a program variable. The initial state is $(0, (*, *, *))$. As actions, we have again an action corresponding to each atomic statement of the program as well as each test of the program and its opposite. As transitions, we have the following:

- for each n:
  - a transition $(0, (*, *, *)) \xrightarrow{\mathsf{read(n)}} (1, (*, n, *))$,
  - a transition $(1, (*, n, *)) \xrightarrow{\mathsf{i:=0}} (2, (0, n, *))$,

- a transition $(2,(0,n,*)) \xrightarrow{\mathsf{f:=1}} (3,(0,n,1))$;
- for each i, n, f such that $i < n$ and $f = i\,!$:
  - a transition $(3,(i,n,f)) \xrightarrow{\mathsf{i<n}} (4,(i,n,f))$,
  - a transition $(4,(i,n,f)) \xrightarrow{\mathsf{i:=i+1}} (5,(i+1,n,f))$,
  - a transition $(5,(i+1,n,f)) \xrightarrow{\mathsf{f:=f*i}} (3,(i+1,n,f\cdot(i+1)))$;
- for each i, n, f such that $i = n$ and $f = i\,!$:
  - a transition $(3,(i,n,f)) \xrightarrow{\mathsf{NOT\,i<n}} (6,(i,n,f))$,
  - a transition $(6,(i,n,f)) \xrightarrow{\mathsf{write(f)}} (7,(i,n,f))$.

There are some noticeable differences between this transition system and the transition system from Example 1.3.1. The two relevant intuitions are as follows. In the same state, reading different numbers does not cause the same state change. In the same state, a test and its opposite do not succeed both.

Not all states are reachable. For example, states $(l,(i,n,f))$ with $i \neq *$ and $n \neq *$ for which $i > n$ holds are not reachable. We did not bother to restrict the transition system to the reachable states: we will see later that the resulting transition system would describe essentially the same behaviour.

Any state where the program counter equals 7 can be regarded a final state regardless the value of the program variables. However, only final states that additionally satisfy that $i = n$ and $f = i!$ are reachable. Thus we have the following final states:

- for each i, n, f such that $i = n$ and $f = i!$:
  - $(7,(i,n,f))\downarrow$.

The transition system (restricted to the reachable states) for the factorial program is represented graphically in Figure 1.9 for the case where $maxint = 2$.

Example 1.3.4 (Greatest common divisor). We also consider again the program from Example 1.3.2. The intended behaviour of this program upon execution on a machine can be described by a transition system as follows. As states of the program, we have pairs (l,s), where $l \in \mathbb{N}$ with $0 \leq l \leq 8$ and $s = (m,n)$ with $m, n \in \{i \in \mathbb{N} \mid 1 \leq i \leq maxint\} \cup \{*\}$. These states are like in Example 1.3.3. The initial state is $(0,(*,*))$. As actions, we have again an action corresponding to each atomic statement of the program as well as each test of the program and its opposite. As transitions, we have the following:

- for each m:
  - a transition $(0,(*,*)) \xrightarrow{\mathsf{read(m)}} (1,(m,*))$;
- for each m, n:
  - a transition $(1,(m,*)) \xrightarrow{\mathsf{read(n)}} (2,(m,n))$;
- for each m, n such that $m > n$:
  - a transition $(2,(m,n)) \xrightarrow{\mathsf{m>n}} (3,(m,n))$,

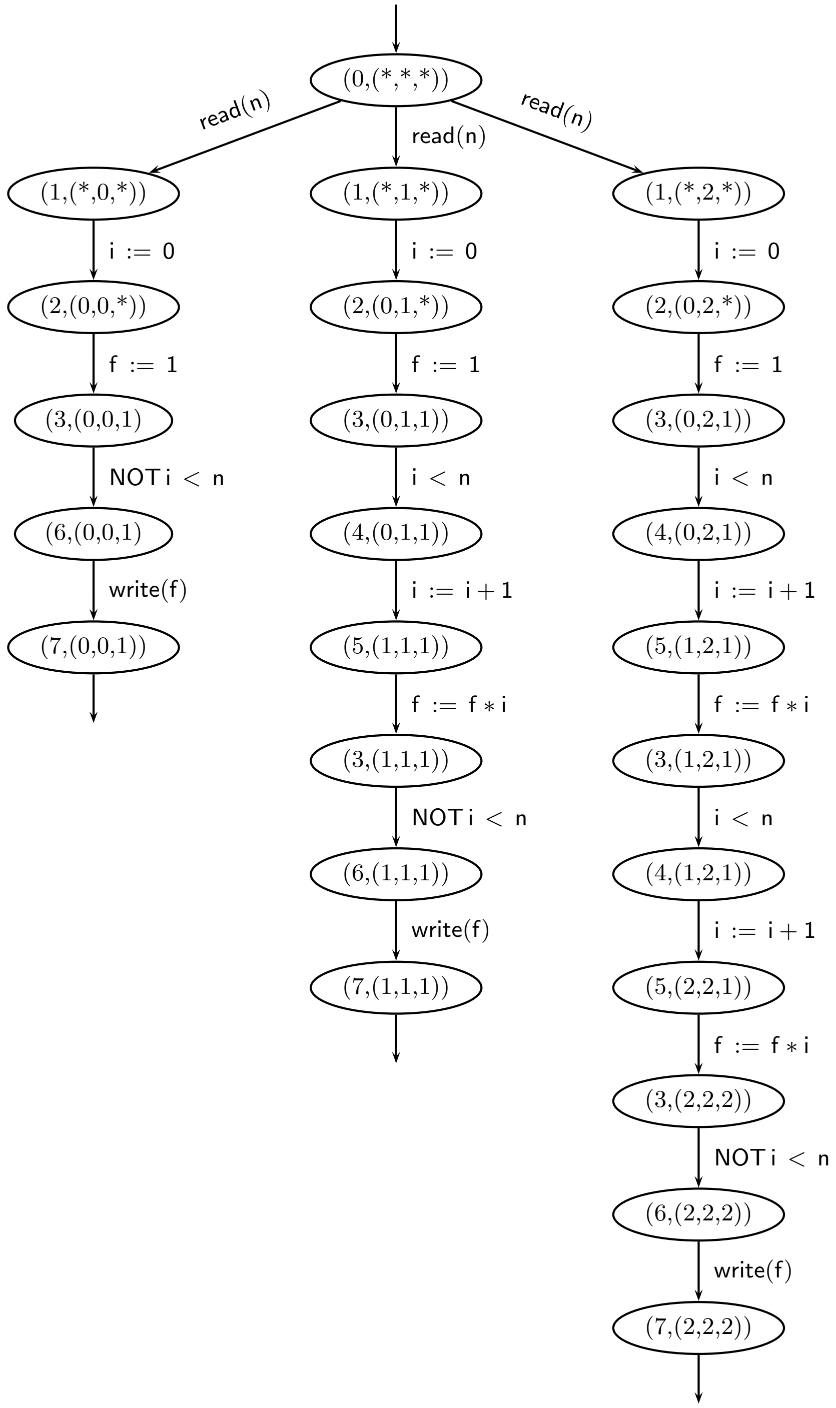


Fig. 1.9. Another transition system for the factorial program

  – a transition $(3,(m,n)) \xrightarrow{\mathsf{m := m - n}} (2,(m-n,n))$;
- for each m, n such that $m \leq n$:
  – a transition $(2,(m,n)) \xrightarrow{\mathsf{NOT\ m > n}} (4,(m,n))$;
- for each m, n such that $m < n$:
  – a transition $(4,(m,n)) \xrightarrow{\mathsf{n > m}} (5,(m,n))$,
  – a transition $(5,(m,n)) \xrightarrow{\mathsf{n := n - m}} (4,(m,n-m))$;
- for each m, n such that $m \geq n$:
  – a transition $(4,(m,n)) \xrightarrow{\mathsf{NOT\ n > m}} (6,(m,n))$;
- for each m, n such that $m \neq n$:
  – a transition $(6,(m,n)) \xrightarrow{\mathsf{NOT\ m = n}} (2,(m,n))$;
- for each m, n such that $m = n$:
  – a transition $(6,(m,n)) \xrightarrow{\mathsf{m = n}} (7,(m,n))$,
  – a transition $(7,(m,n)) \xrightarrow{\mathsf{write(m)}} (8,(m,n))$.

The final states of this transition system are those where the value of the program counter equals 8 and additionally the values of the program variables m and n are equal:

- for each m, n such that $m = n$:
  – $(8,(m,n)) \downarrow$.

The differences between this transition system and the transition system given in Example 1.3.2 are of the same kind as between the transition systems given for factorial program. Like in Example 1.3.3, not all states are reachable. The transition system for the greatest common divisor program is represented graphically in Figure 1.10 for the case where $maxint = 2$.

For a given programming language, the behaviour of its programs upon execution on a machine is called its operational semantics. It is usually described in a style known as structural operational semantics. This means that the behaviour of a compound language construct is described in terms of the behaviour of its constituents. The transition systems from the previous two examples were not formally based on a given (structural) operational semantics.

1.4 Automata and transition systems

For a better understanding of the notion of transition system, we looked in the previous section into its connections with the familiar notion of program. For the same reason, we now look into its connections with the familiar notion of automaton.

Automata can be regarded as a specialized kind of transition systems. In this section, we restrict ourselves to the kind of automata known as *nondeterministic finite accepters*. We also do not allow transitions labelled with

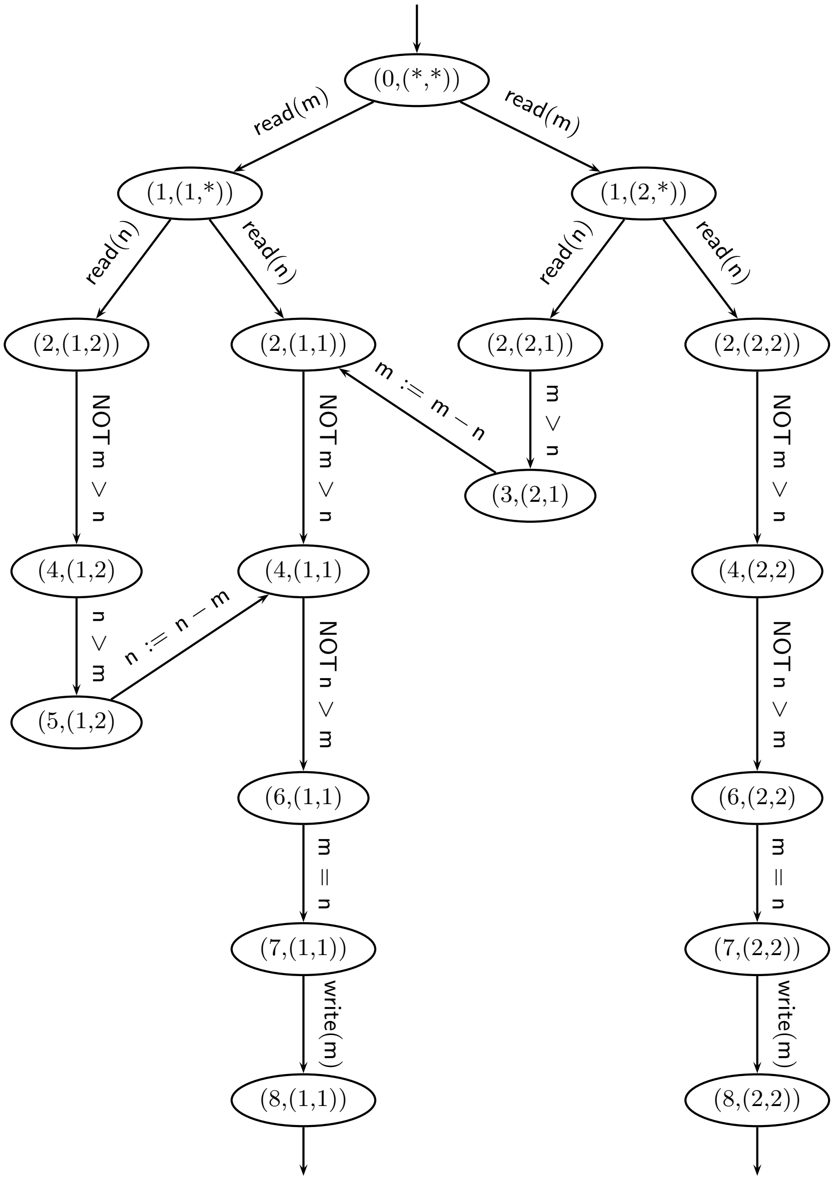


Fig. 1.10. Another transition system for the greatest common divisor program

λ (representing a transition without reading a symbol)[4]. They are illustrative for almost any kind of automata. If no confusion can arise, we will call them simply *automata*. The difference between automata and transition systems is mainly a matter of intended use. As mentioned in Section 1.1, transition systems are primarily regarded as a means to describe the behaviour of processes and automata are primarily regarded as abstract machines to recognize certain languages. Because of the different intended use, final states are indispensable in the case of automata: reaching a final state means that a complete sentence has been recognized. We do not give the standard definition of the notion of automaton. Our definition underlines the resemblance to transition systems mentioned above.

Definition 1.4.1. An *automaton* M is a quintuple $(S, A, \rightarrow, s_0, F)$ where

- S is a finite set of *internal states*;
- A is a finite set of *symbols*, called the *input alphabet*;
- $\rightarrow \subseteq S \times A \times S$ is a set of *transitions*;
- $s_0 \in S$ is the *initial state*;
- $F \subseteq S$ is a set of *final states*.

The set $\twoheadrightarrow \subseteq S \times A^* \times S$ of *generalized transitions* of M is defined exactly as for transition systems. The *language* accepted by M, written $\mathcal{L}(M)$, is the set $\{\sigma \in A^* \mid s_0 \xrightarrow{\sigma}\!\!\!\!\rightarrow s \text{ for some } s \in F\}$.

An important difference between automata and transition systems is that in automata both the set of internal states and the set of symbols are finite. In the standard definition of the notion of automaton, we have a *transition function* $\delta : S \times A \rightarrow \mathcal{P}(S)$ instead of a set $\rightarrow \subseteq S \times A \times S$ of transitions. If we take δ such that $s' \in \delta(s, a)$ if and only if $s \xrightarrow{a} s'$, then we get an automaton according to the standard definition.

If we regard symbols as actions of reading the symbols, automata are simply transition systems with designated final states. An automaton can be considered to accept certain sequences of symbols as follows. A transition of an automaton is regarded as a state change caused by reading a symbol. A sequence of symbols $a_1 \ldots a_n$ is accepted if a sequence of consecutive state changes from the initial state to one of the final states can be obtained by reading the symbols $a_1, \ldots, a_n$ in turn. This informal explanation can be made more precise as follows.

Let M be the automaton $(S, A, \rightarrow, s_0, F)$ and let A' be the set of actions $\{\mathsf{read}(a) \mid a \in A\}$. Now consider the transition system $T = (S, A', \rightarrow', F, s_0)$ where $s_1 \xrightarrow{\mathsf{read}(a)}{}' s_2$ iff $s_1 \xrightarrow{a} s_2$. The sentences of the language accepted by M are exactly the sequences of symbols that can be consecutively read by T till a state is reached that is contained in F.

[4] As any non-deterministic finite accepter with λ-transitions can be replaced by a non-deterministic finite accepter without such λ-transitions such that the two non-deterministic finite accepters accept the same language, this imposes no real limitations.

Let us look at a simple example of the use of automata in recognizing a language.

Example 1.4.1 (Sentences). We consider a very simple language. A sentence of the language consists of a noun clause followed by a verb followed by a noun clause. A noun clause consists of an article followed by a noun. A noun is either man or $\mathsf{machine}$. A verb is either $\mathsf{simulates}$ or mimics. An example sentence is $\mathsf{the\ man\ mimics\ a\ machine}$. This language is accepted by the following automaton. As internal states of the automaton, we have pairs (p, i), where $p \in \{\mathsf{left}, \mathsf{right}\}$ and $i \in \mathbb{N}$ with $0 \leq i \leq 2$. The choice of states is not really relevant. We could have taken the natural numbers 0 to 5 equally well, but the choice made here allows for a short presentation of the automaton. The initial state is $(\mathsf{left}, 0)$ and the only final state is $(\mathsf{right}, 2)$. The input alphabet consists of a, the, man, $\mathsf{machine}$, $\mathsf{simulates}$ and mimics. As transitions, we have the following:

- for $p = \mathsf{left}, \mathsf{right}$:
  - a transition $(p, 0) \xrightarrow{\mathsf{a}} (p, 1)$,
  - a transition $(p, 0) \xrightarrow{\mathsf{the}} (p, 1)$,
  - a transition $(p, 1) \xrightarrow{\mathsf{man}} (p, 2)$,
  - a transition $(p, 1) \xrightarrow{\mathsf{machine}} (p, 2)$;
- a transition $(\mathsf{left}, 2) \xrightarrow{\mathsf{simulates}} (\mathsf{right}, 0)$;
- a transition $(\mathsf{left}, 2) \xrightarrow{\mathsf{mimics}} (\mathsf{right}, 0)$.

The automaton for our very simple language is represented graphically in Figure 1.11. It is obvious that this automaton accepts the same sequences

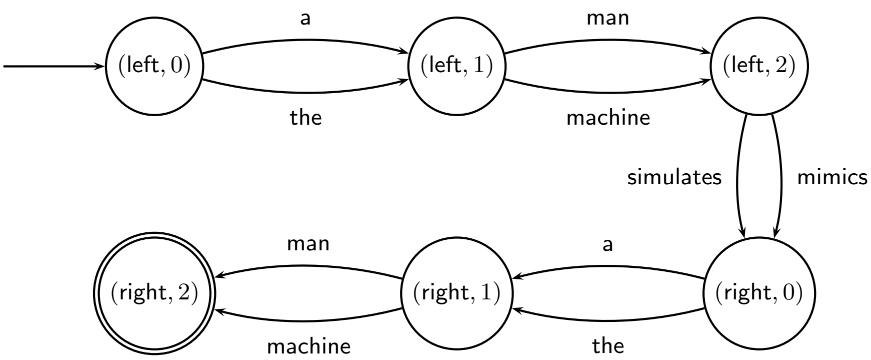


Fig. 1.11. Automaton accepting a very simple language

of symbols as the finite transition system obtained from this automaton by replacing the symbols a, the, man, $\mathsf{machine}$, $\mathsf{simulates}$ and mimics by actions of reading these symbols.

Conversely, we can also view any finite transition system as an automaton by regarding its actions as symbols and its successfully terminating states as final states. This is interesting because the sequences of actions it can consecutively perform are an important aspect of the behaviour of a process. We will get back to that later in Section 1.5. Here is an example that illustrates the potential usefulness of focussing on the sequences of actions that a system can consecutively perform.

Example 1.4.2 (Bounded counter). We consider the bounded counter with bound k from Example 1.1.2. The behaviour of this bounded counter is described by a transition system in Example 1.2.1. The sequences of actions that can be performed are exactly the sequences w that satisfy the following condition:

- for all prefixes[5] v of w, $0 \leq n_{\mathsf{inc}}(v) - n_{\mathsf{dec}}(v) \leq k$;

where $n_a(u)$ stands for the number of occurrences of action a in sequence u. This description of the sequences of actions that can be performed may be regarded as the specification of the intended system.

If we designate all reachable states of the transition system as final states, the transition system can be viewed as an automaton recognizing the language on the alphabet $\{\mathsf{inc}, \mathsf{dec}\}$ that consists of the sequences $w \in \{\mathsf{inc}, \mathsf{dec}\}^*$ satisfying the conditions just mentioned. When viewing the transition system as an automaton, the point is that inc and dec are considered to be symbols to be read instead of actions to be performed.

The following is known from automata theory. The languages that can be accepted by an automaton as defined here, i.e. a non-deterministic finite accepter without λ-transitions, are exactly the regular languages. Intuitively, a regular language has a structure simple enough that a limited memory is sufficient to accept all its sentences. Many actual languages are not regular. Broader language categories include the context-free languages and the context-sensitive languages. They can be accepted by automata of more powerful kinds: non-deterministic pushdown accepters for context-free languages and linear bounded accepters for context-sensitive languages. Those kinds of automata are in turn closely related to restricted kinds of infinite transition systems.

1.5 Equivalences on processes

In this section, we look at a couple of notions that are taken up to abstract from those details of transition systems that are often supposed to be irrelevant.

[5] See Appendix A.3 for a definition of prefixes.

Example 1.5.1. Suppose that we want to implement a program that prints the natural numbers from 1 upto N (for some $N \geq 1$). There are many programs that solve this problem. For example the programs

```
PROGRAM numbers(input,output);
VAR i: 0..maxint;
CONST n = N
BEGIN
   FOR i := 1 TO n DO write(i) OD
END
```

and

```
PROGRAM numbers(input,output);
VAR j: 0..maxint;
CONST n = N
BEGIN
   FOR j := n DOWNTO 1 DO write(n-j+1) OD
END
```

both result in a list of consecutive natural numbers starting from 1 and ending with N being printed on the screen. The transition systems that could be used to express the execution of these programs are $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ where

$$\begin{aligned}
S &= \{0, 1, \ldots, N\},\\
A &= \{0, 1, \ldots, N\},\\
\rightarrow &= \{(i, i+1, i+1) \mid 0 \leq i < N\},\\
\downarrow &= \{N\},\\
s_0 &= 0
\end{aligned}$$

and

$$\begin{aligned}
S' &= \{0, 1, \ldots, N\},\\
A' &= \{0, 1, \ldots, N\},\\
\rightarrow' &= \{(j, N-j+1, j-1) \mid 0 < j \leq N\},\\
\downarrow' &= \{0\},\\
s'_0 &= N
\end{aligned}$$

respectively. Obviously, these transition systems are not the same as $\rightarrow \neq \rightarrow'$, $\downarrow \neq \downarrow'$, and $s_0 \neq s'_0$. Nevertheless, if we are interested in the results of the programs on screen, we can observe no difference between the two programs.

Definition 1.5.1 (Isomorphy). Two transition systems $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ are *isomorphic*, notation $T \cong T'$, if and only if there exists a bijective[6] function $\hbar : \mathrm{reach}(T) \rightarrow \mathrm{reach}(T')$ such that

[6] See Appendix A.2 for a definition of bijective functions.

1. $\hbar(s_0) = s'_0$;
2. whenever $\hbar(s_1) = s'_1$ and $\hbar(s_2) = s'_2$, then $s_1 \xrightarrow{a} s_2$ if and only if $s'_1 \xrightarrow{a}' s'_2$;
3. whenever $\hbar(s) = s'$, then $s \downarrow$ if and only if $s' \downarrow'$.

Example 1.5.2. For the previous example the function $\hbar$ from the states S (which are precisely the reachable states) of T to the (reachable) states S' of T' defined by $\hbar(s) = N - s$ is a bijective function that proves $T \cong T'$. The transition systems and the function $\hbar$ proving the isomorphy between them are illustrated in Figure 1.12 for $N = 4$.

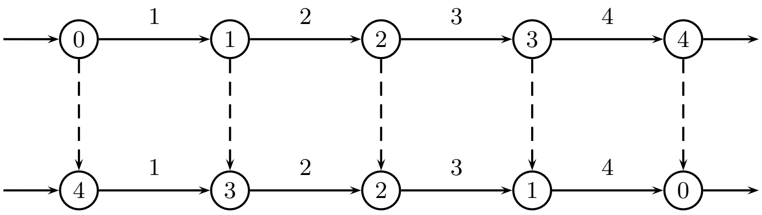
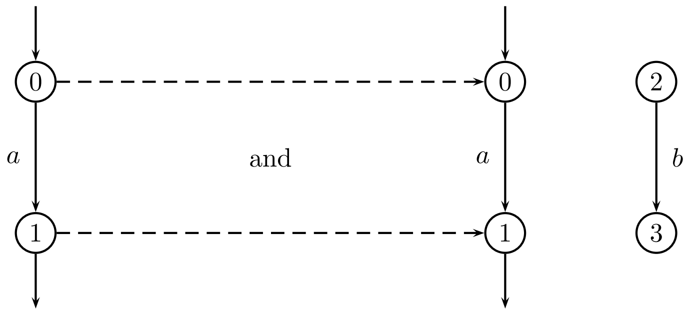


Fig. 1.12. Isomorphic transition systems

Property 1.5.1. Isomorphy is an equivalence relation, i.e., a reflexive, symmetric, and transitive relation.

Example 1.5.3. Consider the transition systems given below. As one can easily see, the second transition system has states that cannot be reached from the initial state by any sequence of actions. Nevertheless, these transition systems are considered to be isomorphic.



The function $\hbar$ that proves this isomorphy is given by $\hbar(0) = 0$ and $\hbar(1) = 1$.

In the sequel we will mostly neglect the unreachable states of transition systems as well as the transitions starting from those. This can be considered an operation on transition systems, called reduction.

Definition 1.5.2. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. Then the *reduction* of T, written $\mathrm{red}(T)$, is the transition system $(S', A', \rightarrow', \downarrow', s_0)$ where

- $S' = \mathrm{reach}(T)$;
- $A' = \{a \in A \mid \text{for some } s, s' \in S' \colon s \xrightarrow{a} s'\}$;
- $\rightarrow' = \rightarrow \cap (S' \times A' \times S')$;
- $\downarrow' = \downarrow \cap S'$.

Any transition system is isomorphic to its reduction, which is a connected transition system.

Property 1.5.2. Let T be a transition system. Then the following holds:

$$T \cong \mathrm{red}(T).$$

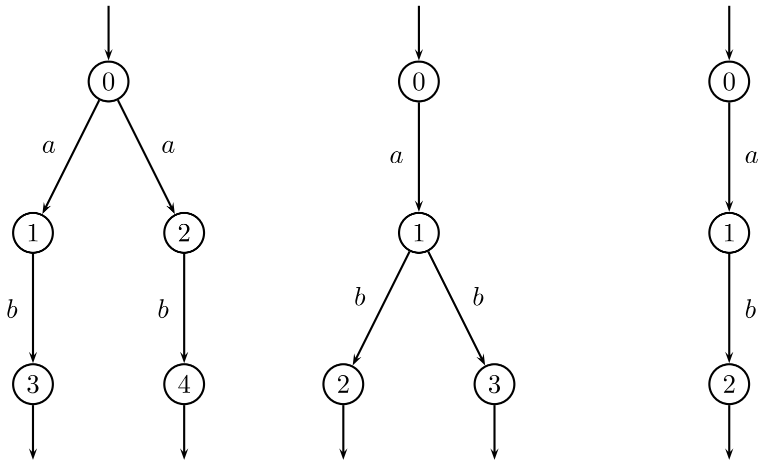
Proof. The function $\hbar : \mathrm{reach}(T) \rightarrow \mathrm{reach}(T')$ defined by $\hbar(s) = s$ for all $s \in \mathrm{reach}(T)$ is a bijective function from the reachable states of T to the reachable states of $\mathrm{red}(T)$ (which are actually the same as the reachable states of T). It also satisfies the conditions from the definition of isomorphy (Definition 1.5.1).

Usually, transition systems show details that are not considered to be relevant to the behaviour of processes. There are, for example, applications of transition systems where only the sequences of actions that can be performed consecutively starting from the initial state and ending in a successfully terminating state, called the terminating traces of the transition system, matter. This is, for example, the case when transition systems are used for the same purposes as automata, i.e., to accept languages.

Definition 1.5.3 (Language equivalence). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. A *terminating* trace of T is a sequence $\sigma \in A^*$ such that $s_0 \xrightarrow{\sigma}\!\!\!\!\rightarrow s$ and $s\downarrow$ for some $s \in S$. We write $\mathrm{lang}(T)$ for the set of all terminating traces of T. Two transition systems T and T' are *language equivalent*, written $T \equiv_{\mathrm{l}} T'$, if $\mathrm{lang}(T) = \mathrm{lang}(T')$.

Obviously the terminology used here is based on viewing a transition system as an automaton by regarding its actions as symbols and its successfully terminating states as final states, cf. Section 1.4.

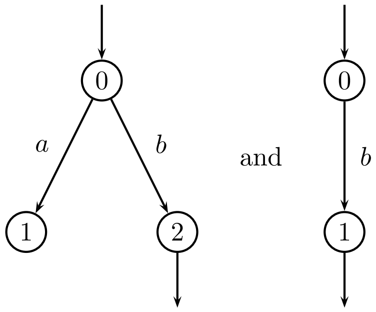
Example 1.5.4. The following transition systems are language equivalent as for all of them the set of terminating traces only consists of the trace ab.



Observe that these transition systems all differ with respect to isomorphy.

The following example illustrates that language equivalence abstracts from the parts of a transition system that do not lead to a successfully terminating state.

Example 1.5.5. The transition systems



called T_1 and T_2, respectively, are language equivalent ($T_1 \equiv_{\mathrm{l}} T_2$) as for both transition systems the set of terminating traces only consists of the trace b: $\mathrm{lang}(T_1) = \mathrm{lang}(T_2) = \{b\}$. Thus the sequence of actions a is neglected in the comparison of the transition systems.

Property 1.5.3. Language equivalence is an equivalence relation.

Transition systems that are isomorphic are identical except for the identity of their states. As trace equivalence also abstracts from the identity of the states, any two isomorphic transition systems are necessarily also language equivalent. This is expressed by the following property.

Property 1.5.4. Any two isomorphic transition systems are also language equivalent: if $T \cong T'$, then $T \equiv_{\mathrm{l}} T'$.

Notice that in all cases where a transition system is used to accept a language, as described in Section 1.4, only the terminating traces are relevant. In fact, language equivalence of automata and language equivalence of the transition systems representing these automata are identical.

Property 1.5.5. Let $T = (S, A, \to, s_0, F)$ and $T' = (S', A', \to, s'_0, F')$ be automata. Then, $\mathcal{L}(T) = \mathcal{L}(T')$ if and only if $T_1 \equiv_l T'_1$, where $T_1 = (S, A, \to, F, s_0)$ and $T'_1 = (S', A', \to', F', s'_0)$.

A prominent aspect of language equivalence is that only the sequences of actions are considered that end in a successfully terminating state. Very often, processes are not supposed to terminate in any state. There are applications of transition systems where all the sequences of actions that can be performed consecutively starting from the initial state of a transition system, called the traces of the transition system, matter. Here is a simple example of a case where only the traces matter.

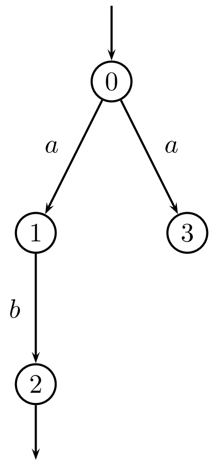
Example 1.5.6. We consider again the the bounded counter with bound k from Example 1.1.2. Its traces are exactly the traces w for which the condition $0 \leq n_{\mathsf{inc}}(w) - n_{\mathsf{dec}}(w) \leq k$ holds[7]. This description of its traces expresses all we expect from the bounded counter: we regard any transition system that has those traces as a bounded counter. For this reason, only the traces are relevant in this case.

In all those cases where only the traces of the transition system matter, it is useful to ignore all other details. This is done by identifying transition systems that not only have the same set of terminating traces, but additionally have the same set of traces. Such transition systems are called trace equivalent. Here is a precise definition.

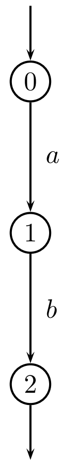
Definition 1.5.4 (Trace equivalence). Let $T = (S, A, \to, \downarrow, s_0)$ be a transition system. A *trace* of T is a sequence $\sigma \in A^*$ such that $s_0 \xrightarrow{\sigma}\!\!\!\!\rightarrow s$ for some $s \in S$. We write traces(T) for the set of all traces of T. Then two transition systems T and T' are *trace equivalent*, written $T \equiv_{\mathrm{tr}} T'$, if traces$(T)$ = traces(T') and lang(T) = lang(T').

Example 1.5.7. Consider the transition systems

[7] Recall from Exercise 1.4.2 that $n_a(u)$ stands for the number of occurrences of action a in sequence u.



and



called T_1 and T_2, respectively. These transition systems have the same terminating traces: $\mathrm{lang}(T_1) = \mathrm{lang}(T_2) = \{ab\}$. Also, they have the same traces: $\mathrm{traces}(T_1) = \mathrm{traces}(T_2) = \{\epsilon, a, ab\}$. Hence, these transition systems are considered trace equivalent: $T_1 \equiv_{\mathrm{tr}} T_2$.

Example 1.5.8. The transition systems T_1 and T_2 from Example 1.5.5 are not trace equivalent ($T_1 \not\equiv_{\mathrm{tr}} T_2$). The transition system T_1 has a trace a, whereas the transition system T_2 does not have such a trace. More precisely: $\mathrm{traces}(T_1) = \{\epsilon, a, b\}$ and $\mathrm{traces}(T_2) = \{\epsilon, b\}$.

Property 1.5.6. Trace equivalence is an equivalence relation.

From the previous discussions and the above example, it should be clear that trace equivalence identifies less transition systems than language equivalence.

Property 1.5.7. Any two trace equivalent transition systems are also language equivalent: if $T \equiv_{\mathrm{tr}} T'$, then $T \equiv_{\mathrm{l}} T'$.

As can be seen from the above definition and examples trace equivalence abstracts from the precise identity of the states of the transition systems and from the moments of branching. Isomorphy only abstracts from the identity of the states and is therefore a stronger equivalence (i.e., it identifies less transition systems). These observations lead to the following property.

Property 1.5.8. Any two isomorphic transition systems are also trace equivalent: if $T \cong T'$, then $T \equiv_{\mathrm{tr}} T'$.

We will see below that there are also cases where not only the traces of the transition system matter. In those cases, trace equivalence is obviously not the right equivalence to make use of.

There exist different viewpoints on what should be considered relevant to the behaviour of processes. The equivalence known as bisimulation equivalence is based on the idea that not only the traces of equivalent transition systems should coincide, but also the stages at which the choices of different possibilities occur. Therefore, bisimulation equivalence is said to preserve the branching structure of transition systems. Here is an example of a case where apparently not only the traces matter, but also the stages at which the choices of different possibilities occur.

Example 1.5.9. We consider a split connection between nodes in a network. A split connection has one input port and two output ports. A datum that has been consumed at the input port can be delivered at either one of the output ports. That is, the choice of the output ports is resolved after the datum has been consumed. The behaviour of a split connection with input port k and output ports l and m can be described as follows. We assume a set of data D. As states of the split connection, we have $*$ and the data $d \in D$, with $*$ as initial state. As actions, we have $\mathsf{s}_i(d)$ (send d at port i) and $\mathsf{r}_i(d)$ (receive d at port i) for $i = k, l, m$ and $d \in D$. As transitions, we have the following:

- for each $d \in D$, a transition $* \xrightarrow{\mathsf{r}_k(d)} d$;
- for each $i \in \{l, m\}$ and $d \in D$, transitions $d \xrightarrow{\mathsf{s}_i(d)} *$.

The transition system for the split connection is represented graphically in Figure 1.13 for the case where $D = \{0, 1\}$. Next we consider a transition

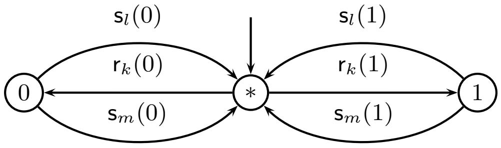


Fig. 1.13. Transition system for the split connection

system that is trace equivalent to the one just presented. As states, we have the pairs (i, d) for $i = k, l, m$ and $d \in D \cup \{*\}$, with $(k, *)$ as initial state. As actions, we still have $\mathsf{s}_i(d)$ and $\mathsf{r}_i(d)$ for $i = k, l, m$ and $d \in D$. As transitions, we have the following:

- for each $i \in \{l, m\}$ and $d \in D$: $(k, *) \xrightarrow{\mathsf{r}_k(d)} (i, d)$, $(i, d) \xrightarrow{\mathsf{s}_i(d)} (k, *)$.

This transition system is represented graphically in Figure 1.14 for the case where $D = \{0, 1\}$. This transition system does not describe the intended behaviour of the split connection correctly. A datum that has been consumed cannot be delivered at either of the output ports because the choice of the output ports is resolved at the instant that the datum is consumed. So, we

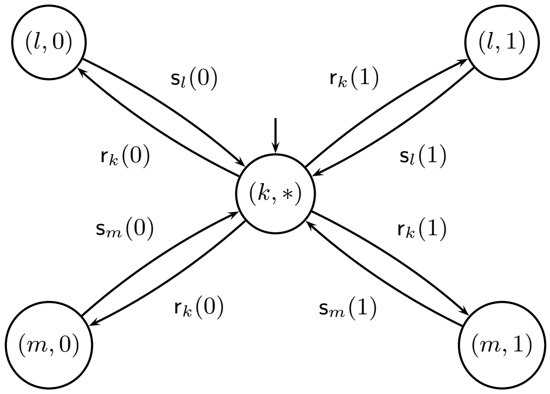


Fig. 1.14. Transition system for the split-like connection

do not want to identify this transition system with the previous one. They are not identified by bisimulation equivalence.

What is exactly meant by "the stages at which the choices of different possibilities occur" in our intuitive explanation of bisimulation equivalence becomes clear in the following informal definition. Two transition systems T and T' are bisimulation equivalent if their states can be related such that:

- the initial states are related;
- if states s_1 and s_1' are related and in T a transition with label a is possible from s_1 to some s_2, then in T' a transition with label a is possible from s_1' to some s_2' such that s_2 and s_2' are related;
- likewise, with the role of T and T' reversed;
- if states s and s' are related and s is a successfully terminating state in T, then s' is a successfully terminating state in T';
- likewise, with the role of T and T' reversed.

This means that, starting from any pair of related states, T can simulate T' and that conversely T' can simulate T.

Bisimulation equivalence can also be characterized as follows: it identifies transition systems if they cannot be distinguished by any conceivable experiment with an experimenter that is only able to detect which actions are performed at any stage and whether the system has terminated successfully. The kind of identifications made by bisimulation equivalence is illustrated with the following example.

Example 1.5.10. We consider a merge connection between nodes in a network. A merge connection has two input ports and one output port. Each datum that has been consumed at one of the input ports is delivered at the output port. The behaviour of a merge connection with input ports k and l and output port m can be described as follows. We assume a set of data D. As states, we have the pairs (i, d) for $i = k, l, m$ and $d \in D \cup \{*\}$, with $(m, *)$ as initial state. As actions, we have again $\mathsf{s}_i(d)$ and $\mathsf{r}_i(d)$ for $i = k, l, m$ and $d \in D$. As transitions, we have the following:

- for each $i \in \{k, l\}$ and $d \in D$: $(m, *) \xrightarrow{\mathsf{r}_i(d)} (i, d)$, $(i, d) \xrightarrow{\mathsf{s}_m(d)} (m, *)$.

This transition system for the merge connection is represented graphically in Figure 1.15 for the case where $D = \{0, 1\}$. Next we consider the following

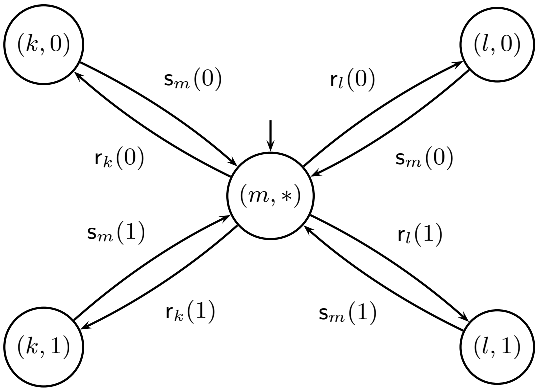


Fig. 1.15. Transition system for the merge connection

transition system. As states, we have $*$ and the data $d \in D$, with $*$ as initial state. As actions, we still have $\mathsf{s}_i(d)$ and $\mathsf{r}_i(d)$ for $i = k, l, m$ and $d \in D$. As transitions, we have the following:

- for each $i \in \{k, l\}$ and $d \in D$, a transition $* \xrightarrow{\mathsf{r}_i(d)} d$;
- for each $d \in D$, a transition $d \xrightarrow{\mathsf{s}_m(d)} *$.

This transition system is represented graphically in Figure 1.16 for the case where $D = \{0, 1\}$. This transition system describes the intended behaviour of the merge connection correctly as well. Is this transition system identified with the previous one by bisimulation equivalence? Yes, it is: relate state $(m, *)$ to state $*$ and, for each $i \in \{k, l\}$ and $d \in D$, state (i, d) to state d.

Let us now give the formal definition of bisimulation equivalence.

Definition 1.5.5. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ be transition systems such that $A = A'$. Then a *bisimulation* B between T and T' is a binary relation $B \subseteq S \times S'$ such that the following conditions hold:

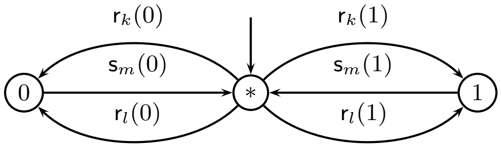


Fig. 1.16. Another transition system for the merge connection

1. $B(s_0, s'_0)$;
2. whenever $B(s_1, s'_1)$ and $s_1 \xrightarrow{a} s_2$, then there is a state s'_2 such that $s'_1 \xrightarrow{a}{}' s'_2$ and $B(s_2, s'_2)$;
3. whenever $B(s_1, s'_1)$ and $s'_1 \xrightarrow{a}{}' s'_2$, then there is a state s_2 such that $s_1 \xrightarrow{a} s_2$ and $B(s_2, s'_2)$;
4. whenever $B(s, s')$ and $s\downarrow$, then $s' \downarrow'$;
5. whenever $B(s, s')$ and $s' \downarrow'$, then $s\downarrow$.

The two transition systems T and T' are *bisimulation equivalent* (also called *bisimilar*), written $T \leftrightarroweq T'$, if there exists a bisimulation B between T and T'. A bisimulation between T and T is called an *autobisimulation* on T.

Example 1.5.11. A bisimulation relation between graphical representations of transition systems is usually indicated by connecting all the states that are related by the bisimulation relation by means of a dashed line. For the bisimulation equivalent transition systems of the merge connection, this visualization is shown in Figure 1.17.

Property 1.5.9. Bisimulation equivalence is an equivalence relation.

Restriction to relations B between the reachable states of T and the reachable states of T' does not change the notion of bisimulation equivalence.

Bisimulation equivalence identifies more transition systems than isomorphy, but less than trace equivalence. In Example 1.5.9, we have seen two transition systems that are trace equivalent, but not bisimulation equivalent. The two transition systems depicted in Figure 1.18 are not isomorphic. They are, however, bisimulation equivalent.

Property 1.5.10. Any two isomorphic transition systems are also bisimulation equivalent: if $T \cong T'$, then $T \leftrightarroweq T'$.

Property 1.5.11. Any two bisimilar transition systems are also trace equivalent: if $T \leftrightarroweq T'$, then $T \equiv_{\mathrm{tr}} T'$.

Let us return to the experimenter that is only able to detect which actions are performed at any stage. If performing the same experiment on a system more than once leads to the same outcome for all his (or her) experiments, the

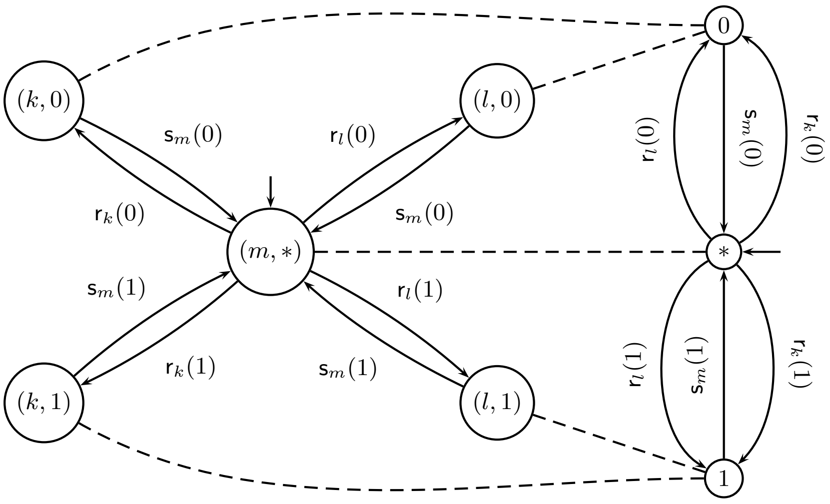


Fig. 1.17. A bisimulation relation for the merge connection transition systems

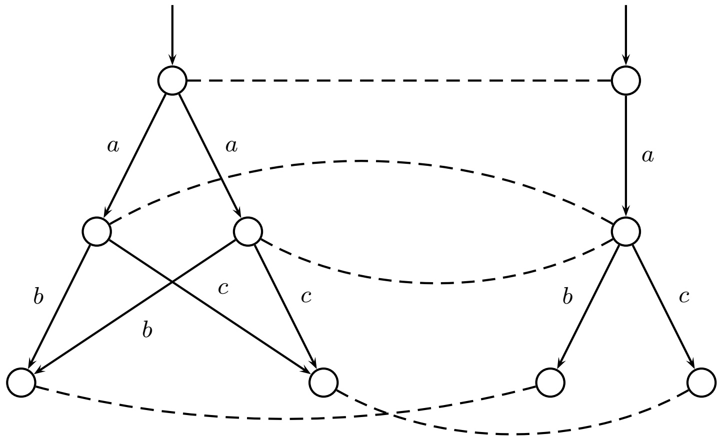


Fig. 1.18. Two transition systems that are not isomorphic, but are bisimulation equivalent

system behaves predictably. Such a system is called determinate. This is an important notion in the design of a system. In many cases, we have to arrive at a determinate system from components of which some are not determinate. This is, for example, the case with the simple data communication protocol treated in the next chapter. Here is the precise definition of determinacy.

Definition 1.5.6 (Determinacy). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. Then T is *determinate* if the following condition holds:

whenever $s_0 \overset{\sigma}{\twoheadrightarrow} s$ and $s_0 \overset{\sigma}{\twoheadrightarrow} s'$, then there is an autobisimulation B on T such that $B(s, s')$.

For determinate transition systems trace equivalence and bisimulation equivalence coincide.

Property 1.5.12. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s_0')$ be transition systems such that $A = A'$. Then the following holds:

if T and T' are determinate, then $T \underline{\leftrightarrow} T'$ if and only if $T \equiv_{\text{tr}} T'$.

A direct consequence of the above property is that for determinate transition systems, the equivalences discussed in this section, namely isomorphy, trace equivalence, and bisimulation equivalence coincide.

Property 1.5.13. For determinate transition systems T and T' the following holds: $T \cong T'$ if and only if $T \equiv_{\text{tr}} T'$ if and only if $T \underline{\leftrightarrow} T'$.

The notion of determinism of a transition system is closely related to the notion of determinacy of a transition system.

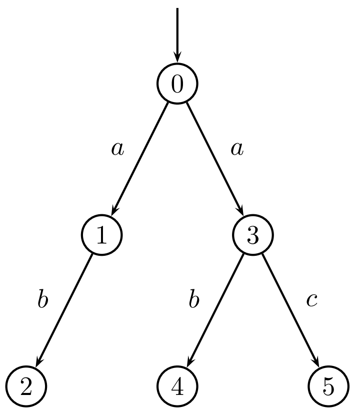
Definition 1.5.7 (Determinism). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. Then T is *deterministic* if the following condition holds:

whenever $s_0 \overset{\sigma}{\twoheadrightarrow} s$ and $s_0 \overset{\sigma}{\twoheadrightarrow} s'$, then $s = s'$.

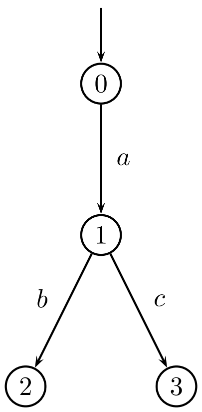
It is easy to see that all deterministic transition systems are determinate, but not all determinate transition systems are deterministic. One could say that a determinate transition system is deterministic up to bisimulation.

In this section and the previous one, we have shortly introduced the use of equivalences for abstraction from details of transition systems that we want to ignore. This plays a prominent part in techniques for the analysis of process behaviour. We will come back to trace and bisimulation equivalence later.

Exercise 1.5.1. Determine whether the transition systems

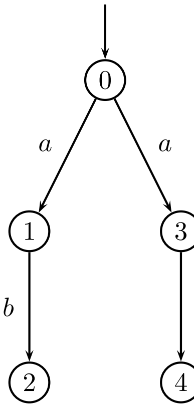


and

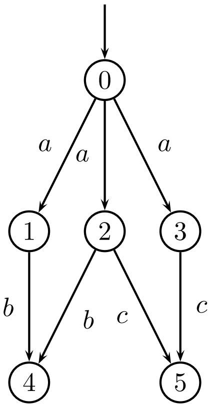


are language equivalent. Determine also whether they are trace equivalent, bisimulation equivalent, and isomorphic.

Exercise 1.5.2. Determine whether the transition systems



and



are language equivalent. Determine also whether they are trace equivalent, bisimulation equivalent, and isomorphic.

Exercise 1.5.3. Draw the transition systems $T = (S, A, \rightarrow, \downarrow, s_0)$ where

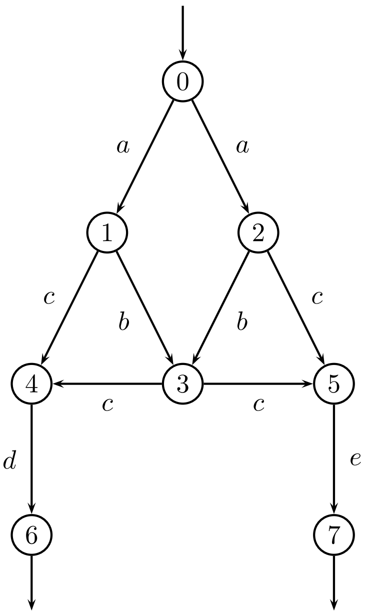
$$\begin{array}{ll} S & = \{0, 1, \ldots, 7\}, \\ A & = \{a, b, c\}, \\ \rightarrow & = \{(0, a, 1), (1, b, 2), (2, c, 3), (0, a, 4), (4, b, 5), (5, c, 6), (5, b, 7)\}, \\ \downarrow & = \emptyset, \\ s_0 & = 0. \end{array}$$

and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ where

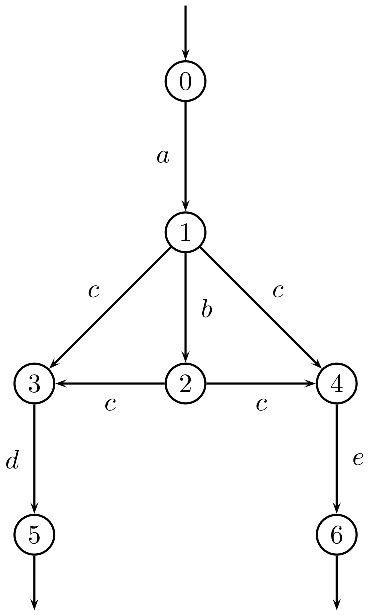
$$\begin{array}{ll} S' & = \{0, 1, \ldots, 4\}, \\ A' & = \{a, b, c\}, \\ \rightarrow' & = \{(0, a, 1), (1, b, 2), (2, c, 3), (1, b, 4)\}, \\ \downarrow' & = \emptyset, \\ s'_0 & = 0. \end{array}$$

Determine whether these transition systems are trace equivalent ($T \equiv_{\mathrm{tr}} T'$) and/or bisimulation equivalent ($T \leftrightarrows T'$).

Exercise 1.5.4. Determine whether the transition systems

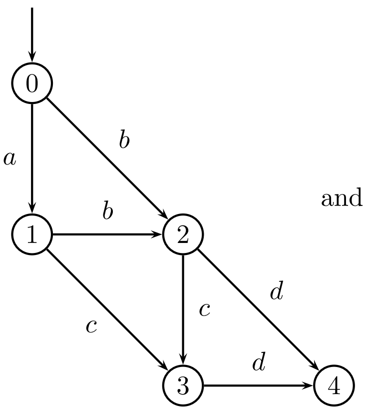


and

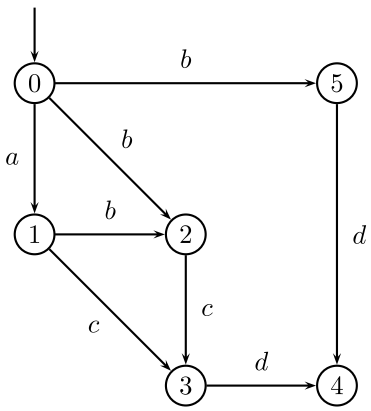


are language equivalent, trace equivalent, and/or bisimulation equivalent.

Exercise 1.5.5. Determine whether the transition systems



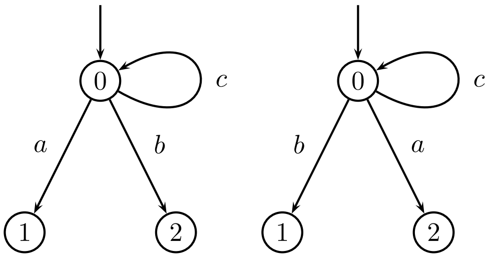
and

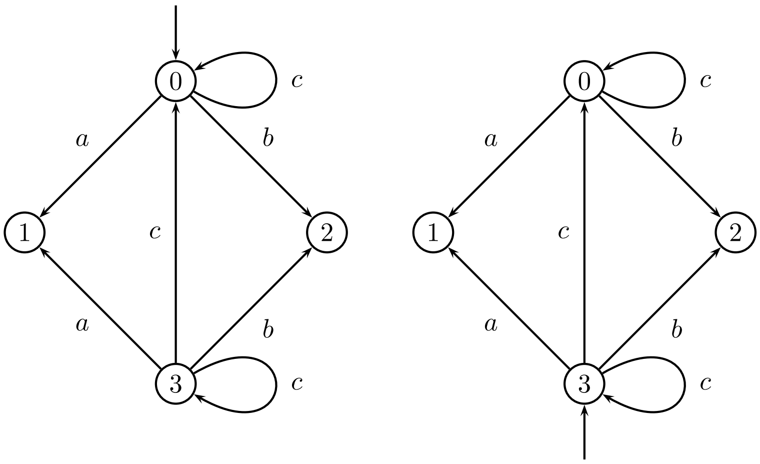


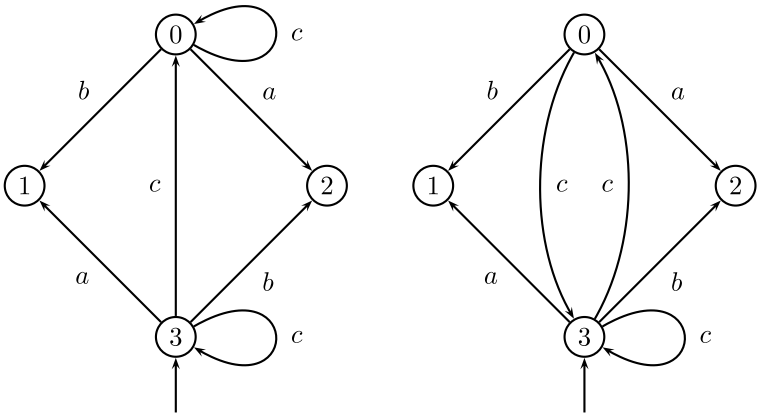
are language equivalent, trace equivalent, and/or bisimulation equivalent.

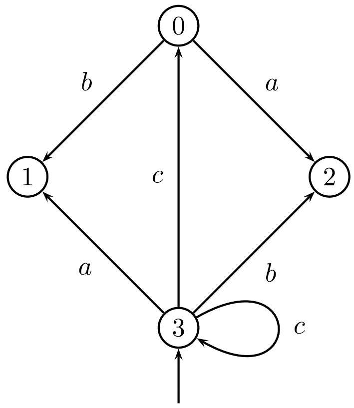
Exercise 1.5.6. Give a determinate transition system that is not deterministic.

Exercise 1.5.7. Consider the transition systems given below. Give, for each of them, the reachable states and the deadlock states. Find, for each pair of transition systems, the strongest equivalence (from: isomorphism, bisimulation equivalence, trace equivalence, and language equivalence) between them.


0
c
a
b
1
2
0
c
b
a
1
2


0
c
a
b
1
c
2
a
b
3
c
0
c
a
b
1
c
2
a
b
3
c


0
c
b
a
1
c
2
a
b
3
c
0
b
a
1
c
c
2
a
b
3
c

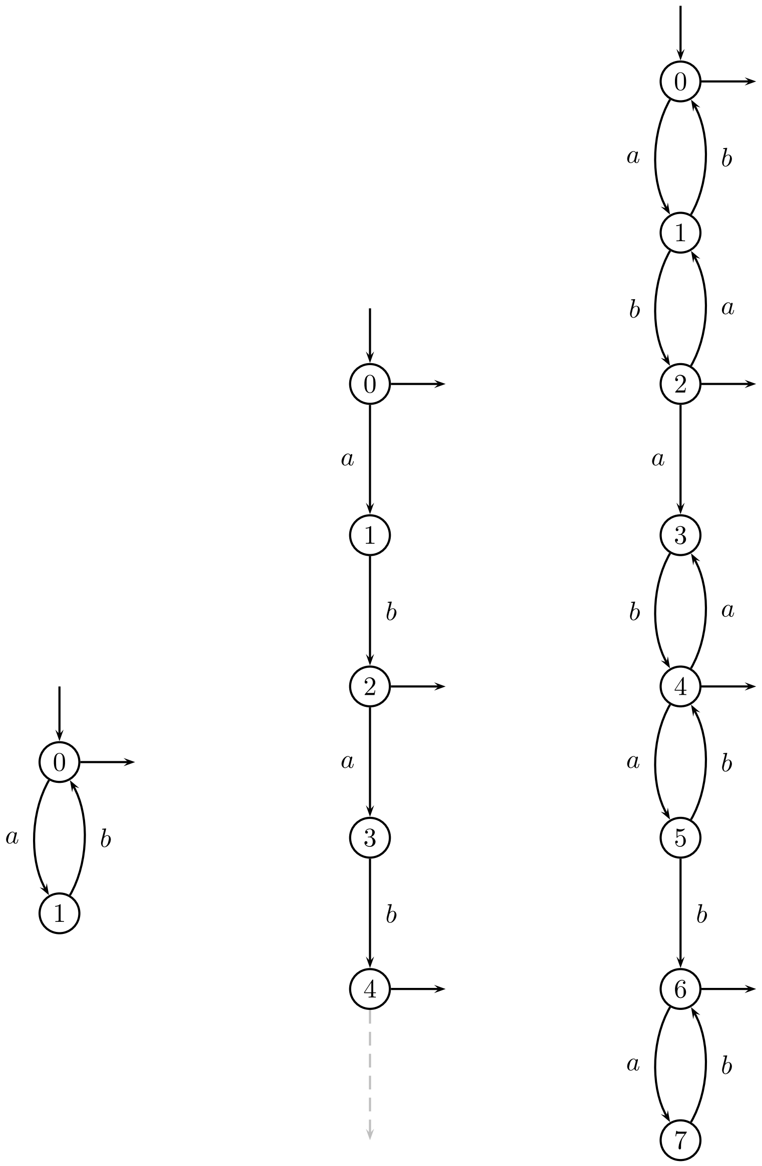


Exercise 1.5.8. Consider the transition systems given below.

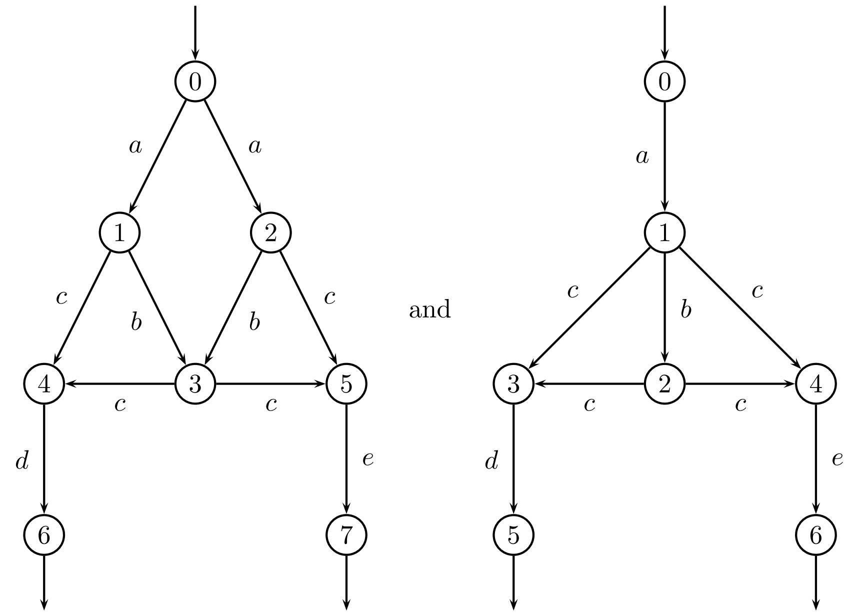
1. Determine for each pair of transition systems whether these are bisimulation equivalent. If so, give a bisimulation between them; if not, give a formula in Hennessy-Milner logic that discriminates between them.
2. Give, for each of these transition systems, the maximal autobisimulation[8]. Determine whether these maximal autobisimulations are equivalence relations.
3. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a given transition system and let B be a maximal autobisimulation on T. Then we define the B-reduction of T, notation $T_{/B}$, to be the transition system $T_{/B} = (S, A', \rightarrow', \downarrow', s'_0)$ where
   - S' is the set of equivalence classes of S under the equivalence B: $S' = S_{/B} = \{[s]_{/B} \mid s \in S\}$ where the equivalence class of $s \in S$ with respect to B, notation $[s]_{/B}$, is defined as $[s]_{/B} = \{s' \in S \mid (s, s') \in B\}$.
   - $A' = A$
   - $\rightarrow' = \{([s]_{/B}, a, [s']_{/B}) \mid s \xrightarrow{a} s'\}$
   - $\downarrow' = \{[s]_{/B} \mid s \downarrow\}$
   - $s'_0 = [s_0]_{/B}$

   Compute for each of the given transition systems, their reduction with respect to the maximal autobisimulation.
4. Prove the following proposition: for any transition system T and any maximal autobisimulation B on T, $T \leftrightarrows T_{/B}$.

[8] The maximal autobisimulation of a transition system T is the largest autobisimulation. Actually, it is the union of all autobisimulations on T.



Exercise 1.5.9. Consider the following transition systems. These are not bisimulation equivalent.



Add transitions to one of these transition systems in such a way that they become bisimulation equivalent. Give the bisimulation relation that proves this.

Exercise 1.5.10 (Unbounded counter). Let the transition system $T = (S, A, \rightarrow, \downarrow, s_0)$ be given by

$$S = \mathbb{N}$$
$$A = \{\mathsf{inc}, \mathsf{dec}\}$$
$$\rightarrow = \{(n, \mathsf{inc}, n+1), (n+1, \mathsf{dec}, n) \mid n \in \mathbb{N}\}$$
$$\downarrow = \emptyset$$
$$s_0 = 0$$

Is there a finite transition system T' such that $T \leftrightarrows T'$? If so, give such a finite transition system; if not, explain why not.

1.6 Hennessy-Milner logic

So far, we have discussed equality of transition systems. Though, in many cases, we are not so much interested in equality, but more in whether or not, a model of a real-life system satisfies some properties. Such properties are often expressed as a logical formula.

A simple, though not very expressive, logic that can be used for this purpose is Hennessy-Milner logic.

Definition 1.6.1. The formulas of Hennessy-Milner logic over a set of actions A are the following:

1. *true* and *false* are formulas,
2. $\checkmark$ is a formula,
3. for all formulas ϕ: $\neg\phi$ is a formula,
4. for all formulas ϕ and ψ: $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \implies \psi$, and $\phi \Leftrightarrow \psi$ are formulas,
5. for any set I and formulas ϕ_i (for $i \in I$): $\bigwedge_{i \in I} \phi_i$ and $\bigvee_{i \in I} \phi_i$ are formulas,
6. for all formulas ϕ and sets of actions $K \subseteq A$: $\langle K \rangle \phi$ and $[K]\phi$ are formulas.

The collection of all formulas of Hennessy-Milner logic over A is denoted $\mathcal{HM}(A)$.

As a notational convention, for a finite set $K = \{a_1, \cdots, a_n\}$, we write $[a_1, \cdots, a_n]\phi$ and $\langle a_1, \cdots, a_n \rangle \phi$, respectively, instead of $[\{a_1, \cdots, a_n\}]\phi$ and $\langle \{a_1, \cdots, a_n\} \rangle \phi$.

Now that we have introduced a means to describe properties of processes, it is time to consider the validity of such a property given a process described as a transition system. The formulas *true* and *false* are satisfied by all states and no state respectively. The formula $\checkmark$ is satisfied if and only if the state is a successful termination state. The interpretation of the standard logical connectives is precisely as expected. The formula $[K]\phi$ is satisfied in a state s if each state that can be reached from the state s by performing an action from K satisfies the formula ϕ. Similarly, $\langle K \rangle \phi$ is satisfied if there is some state reached by an action from K that satisfies ϕ.

Definition 1.6.2 (Satisfaction). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. The satisfaction relation $\models_T \subseteq S \times \mathcal{HM}(A)$ is defined inductively as follows: for $s \in S$

1. $s \models_T$ *true* for all $s \in S$;
2. $s \models_T$ *false* for no $s \in S$;
3. $s \models_T \checkmark$ if and only if $s \downarrow$;
4. $s \models_T \neg\phi$ if and only if not $s \models_T \phi$;
5. $s \models_T \phi \wedge \psi$ if and only if $s \models_T \phi$ and $s \models_T \psi$;
6. $s \models_T \phi \vee \psi$ if and only if $s \models_T \phi$ or $s \models_T \psi$;
7. $s \models_T \phi \implies \psi$ if and only if not $s \models_T \phi$, or $s \models_T \psi$;
8. $s \models_T \phi \Leftrightarrow \psi$ if and only if $s \models_T \phi$ if and only if $s \models_T \psi$;
9. $s \models_T \bigwedge_{i \in I} \phi_i$ if and only if $s \models \phi_i$ for all $i \in I$;
10. $s \models_T \bigvee_{i \in I} \phi_i$ if and only if $s \models \phi_i$ for some $i \in I$;
11. $s \models_T \langle K \rangle \phi$ if and only if $s' \models_T \phi$ for some $a \in K$ and $s' \in S$ such that $s \xrightarrow{a} s'$;
12. $s \models_T [K]\phi$ if and only if $s' \models_T \phi$ for all $a \in K$ and $s' \in S$ such that $s \xrightarrow{a} s'$.

A transition system $T = (S, A, \rightarrow, \downarrow, s_0)$ is said to satisfy a formula ϕ if and only if $s_0 \models_T \phi$. This is denoted by $\models_T \phi$.

Given a transition system T and a state s, we can express that an a-action can be performed from this state as $\langle a\rangle \mathit{true}$. Similarly, expressing that a certain action a cannot be performed, is achieved through the formula $[a]\mathit{false}$.

Example 1.6.1 (Bounded counter). Consider the bounded counter from Example 1.1.2 with bound $k = 3$. The formula $[\mathsf{inc}]\mathit{true}$ expresses that initially, the bounded counter can execute the action inc and that in each state resulting from executing an inc-action, the formula *true* must hold. In this case, there is one possible inc-transition from state 0, i.e. the initial state. As the formula *true* holds in each state, obviously it also holds in each state reached by performing the inc-action. Hence, the bounded counter satisfies the property that initially an increment action can be performed: $\models_T [\mathsf{inc}]\mathit{true}$.

Initially, it is impossible for the bounded counter to perform a decrement action. That this property is captured by the formula $[\mathsf{dec}]\mathit{false}$ can be seen as follows. If there would be a dec-transition from state 0, leading to a state s, then the formula *false* must be satisfied in this state s. But, by definition, the formula *false* is not satisfied by each state. Thus the formula $[\mathsf{dec}]\mathit{false}$ excludes the possibility of a dec-action to be performed.

If we consider the same two formulas but now with respect to the modulo counter from the same example, we observe that both formulas are satisfied by the modulo counter. Hence, one could say that the bounded counter and the modulo counter are different processes since they have different properties. We return to this subject shortly.

Example 1.6.2 (Traces and terminating traces). In the previous section we have seen that one way of looking at processes is through the traces of such a process. The formulas of Hennessy-Milner logic are capable of expressing the property that a certain sequence of actions $a_1 \cdots a_n$ is among the traces of a process through the formula $\langle a_1\rangle\langle a_2\rangle\cdots\langle a_n\rangle \mathit{true}$. Similarly, the formula $\langle a_1\rangle\langle a_2\rangle\cdots\langle a_n\rangle\checkmark$ states that the process has a terminating trace $a_1 \cdots a_n$.

Example 1.6.3 (Necessity of action execution). Very often it is desirable to not only express the possibility of certain actions occurring but also the necessity of their execution. Suppose we want to express that a certain action a has to take place. This can be captured by expressing that action a can be performed and by expressing that all actions except for a cannot be performed and that there can be no successful termination. Hence, the formula

$$\langle a\rangle \mathit{true} \wedge [A \setminus \{a\}]\mathit{false} \wedge \neg\checkmark$$

expresses that the action a must happen next.

This formula can easily be generalized to express that an action from a certain set $K \subseteq A$ must occur next:

$$\langle K\rangle \mathit{true} \wedge [A \setminus K]\mathit{false} \wedge \neg\checkmark .$$

Example 1.6.4 (Deadlock). For a given transition system T and a state s from T, the formula $[A]\mathit{false} \wedge \neg\checkmark$ and the formula $\neg(\langle A\rangle \mathit{true} \vee \checkmark)$ both express that this state represents a deadlock. Hence, absence of deadlock is specificied by means of $\neg([A]\mathit{false} \wedge \neg\checkmark)$ or $\langle A\rangle \mathit{true} \vee \checkmark$.

Example 1.6.5. Consider the transition systems given in Figure 1.19. Formally

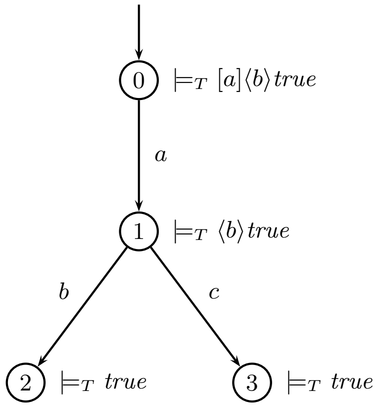


Fig. 1.19. Example of transition systems with different properties

they can be represented by transition systems $T = (S, A, \rightarrow, s_0)$ and $T' = (S', A', \rightarrow', s'_0)$ where

$$\begin{aligned} S &= \{0, 1, 2, 3\}, \\ A &= \{a, b, c\}, \\ \rightarrow &= \{(0, a, 1), (1, b, 2), (1, c, 3)\}, \\ s_0 &= 0 \end{aligned}$$

and

$$\begin{aligned} S' &= \{0, 1, 2, 3, 4\}, \\ A' &= \{a, b, c\}, \\ \rightarrow' &= \{(0, a, 1), (0, a, 2), (1, b, 3), (2, c, 4)\}, \\ s'_0 &= 0. \end{aligned}$$

Observe that these transition systems are trace equivalent ($T_1 \equiv_{\mathrm{tr}} T_2$) and not bisimilar ($T_1 \not\leftrightarrow T_2$). Consider the formula $[a]\langle b\rangle \mathit{true}$, which, in words, states “after the execution of an a-transition, there is the possibility to execute a b-transition”. This property is satisfied by transition system T, but not by transition system T': $\models_T [a]\langle b\rangle \mathit{true}$ and $\not\models_{T'} [a]\langle b\rangle \mathit{true}$. In Figure 1.19 the (relvant) formulas that are satisfied by the different states are listed.

In the previous section, processes represented by transition systems have been compared to each other by considering the transitions that can or cannot be performed. A different way of comparing transition systems is through the properties that they satisfy. It is very natural to consider two transition systems equivalent if they satisfy the same properties. The following theorem states that two transition systems are bisimulation equivalent if and only if they satisfy the same properties expressed by means of Hennessy-Milner logic.

Theorem 1.6.1. *Let $T = (S, A, \rightarrow, s_0)$ and $T' = (S', A', \rightarrow', s_0')$ be transition systems. Then we have $T \leftrightarrows T'$ if and only if for all formulas $\phi \in \mathcal{HM}(A \cup A')$: $\models_T \phi$ if and only if $\models_{T'} \phi$.*

Exercise 1.6.1. Give a formula that discriminates the transition systems given in Exercise 1.5.1.

Exercise 1.6.2. Give a formula that discriminates the transition systems given in Exercise 1.5.2.

Exercise 1.6.3. Draw the transition systems $T = (S, A, \rightarrow, s_0)$ where

$$\begin{aligned}
S &= \{0, 1, \ldots, 8\}, \\
A &= \{a, b, \ldots, f\}, \\
\rightarrow &= \{(0, a, 1), (1, b, 2), (1, c, 3), (3, d, 4), \\
&\quad\;\; (0, a, 5), (5, c, 6), (6, e, 7), (5, f, 8)\}, \\
s_0 &= 0,
\end{aligned}$$

and $T' = (S', A', \rightarrow', s_0')$ where

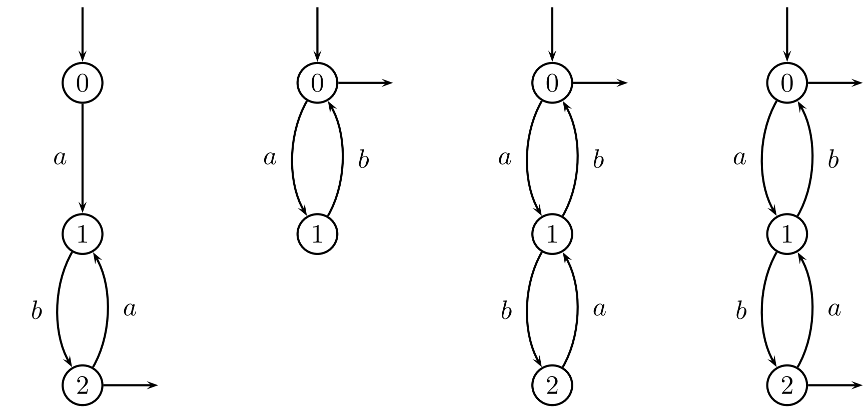
$$\begin{aligned}
S' &= \{0, 1, \ldots, 8\}, \\
A' &= \{a, b, \ldots, f\}, \\
\rightarrow' &= \{(0, a, 1), (1, b, 2), (1, c, 3), (3, e, 4), \\
&\quad\;\; (0, a, 5), (5, c, 6), (6, d, 7), (5, f, 8)\}, \\
s_0' &= 0.
\end{aligned}$$

Determine whether these transition systems are trace equivalent ($T \equiv_{\mathrm{tr}} T'$) and/or bisimulation equivalent ($T \leftrightarrows T'$). If they are not bisimulation equivalent give a formula ϕ that is satisfied by T, but not by T' and a formula ψ that is satisfied by T', but not by T.

Exercise 1.6.4. Give a formula that discriminates the transition systems given in Exercise 1.5.4.

Exercise 1.6.5. Give a formula that discriminates the transition systems given in Exercise 1.5.5.

Exercise 1.6.6. Consider the following transition systems.



Determine for each pair of transition systems whether these are bisimulation equivalent. If so, give a bisimulation between them; if not, give a formula in Hennessy-Milner logic that discriminates between them.

Exercise 1.6.7. Consider the transition systems T' and T'' where $T' = (S', A', \rightarrow', \downarrow', s'_0)$ is given by

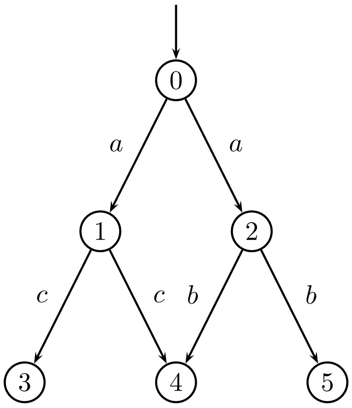
$$
\begin{aligned}
&S' = \mathbb{R}^{\geq 0}\\
&A' = \{\mathsf{half}\}\\
&\rightarrow' = \{(2x, \mathsf{half}, x) \mid x \in \mathbb{R}^{\geq 0}\}\\
&\downarrow' = \emptyset\\
&s'_0 = 1
\end{aligned}
$$

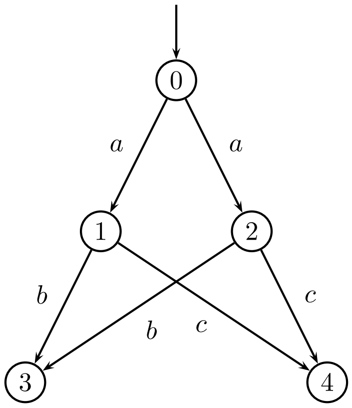
and $T'' = (S'', A'', \rightarrow'', \downarrow'', s''_0)$ is given by

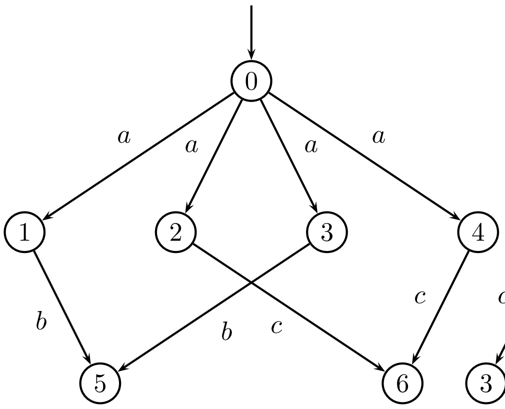
$$
\begin{aligned}
&S'' = \mathbb{R}^{\geq 0}\\
&A'' = \{\mathsf{half}\}\\
&\rightarrow'' = \{(2x, \mathsf{half}, x) \mid x \in \mathbb{R}^{>0}\}\\
&\downarrow'' = \emptyset\\
&s''_0 = 1.
\end{aligned}
$$

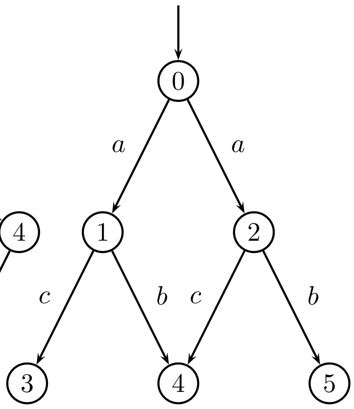
Determine whether these transition systems are bisimulation equivalent. If so, give a bisimulation between them; if not, give a formula in Hennessy-Milner logic that discriminates between them.

Exercise 1.6.8. Consider the transition systems given below. Determine for each pair of transition systems whether these are bisimulation equivalent. If so, give a bisimulation between them; if not, give a formula in Hennessy-Milner logic that discriminates between them.

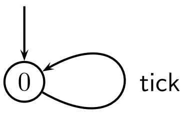








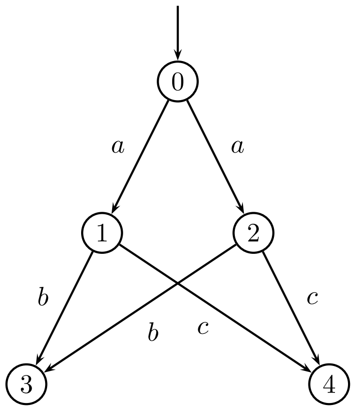
Exercise 1.6.9. Consider the following transition system.



Verify whether or not the following Hennessy-Milner formula are satisfied by this process:

1. $\langle \mathsf{tick} \rangle \mathit{true}$
2. $[\mathsf{tick}]\mathit{false}$
3. $[\mathsf{tick}](\langle \mathsf{tick} \rangle \mathit{true} \wedge [\mathsf{tock}]\mathit{false})$
4. $[\mathsf{tick}](\langle \mathsf{tock} \rangle \mathit{true} \vee [\mathsf{tick}]\mathit{false})$

Exercise 1.6.10. Consider the following transition system.



Verify whether or not the following Hennessy-Milner formula are satisfied by this process:

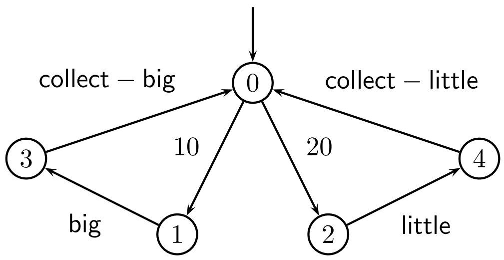
1. $[a]([a]\mathit{true} \vee [a]\mathit{false})$
2. $[a]([b]\checkmark \wedge [c]\mathit{true})$
3. $[a]([a]\checkmark \wedge [c]\mathit{true})$

Exercise 1.6.11. Consider the example of the bounded counter with bound 3 (Example 1.1.2). Give a formula in Hennessy-Milner logic that expresses that it is impossible to perform 4 consecutive increments of the counter.

Exercise 1.6.12. Consider the example of a split connection (Example 1.5.9).

1. Give a formula in Hennessy-Milner logic that expresses that any datum put into the split connection via its input port, will be transferred to one of the output ports next.
2. Give a formula in Hennessy-Milner logic that expresses that it is impossible to have two consecutive input actions.
3. Similarly as the previous question, only this case for two consecutive output actions.

Exercise 1.6.13. Consider the following simple vending machine. First, a coin (either 10 or 20 eurocents) has to be deposited. If a 10 eurocent coin has been deposited a button for requesting a small cup of coffee can be depressed. Alternatively, after depositing 20 eurocents, a big coffee can be ordered. The right amount of coffee is dispensed by the vending machine.



Give formula in Hennessy-Milner logic for the following properties and verify whether these are satisfied by the vending machine:

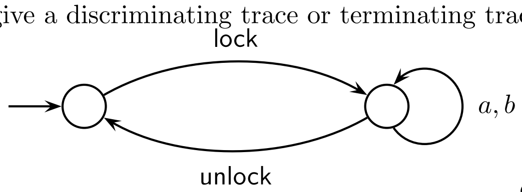
1. a button cannot be depressed (before money is deposited);
2. after 20 eurocent is deposited, the little button cannot be depressed whereas the big one can;
3. after a coin has been deposited no other coin may be deposited;
4. after a coin is deposited and a button is depressed, an item can be collected.

Exercise 1.6.14. Consider the following two transition systems.

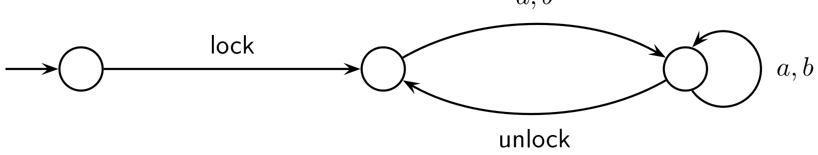
Is it possible to give a Hennessy-Milner formula that is satisfied by one of these transition systems and not by the other? If so, give such a formula; if not, prove this.

Exercise 1.6.15.
Determine for any two of the following transition systems whether they are trace equivalent, bisimulation equivalent, or neither of those. In case they are bisimulation equivalent, give a bisimulation relation that proves this. In case they are not bisimulation equivalent but are trace equivalent, give a formula in Hennessy-Milner logic that discriminates between them. In case they are not trace equivalent, give a discriminating trace or terminating trace.

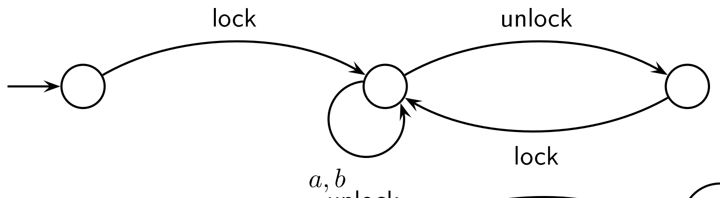
1.



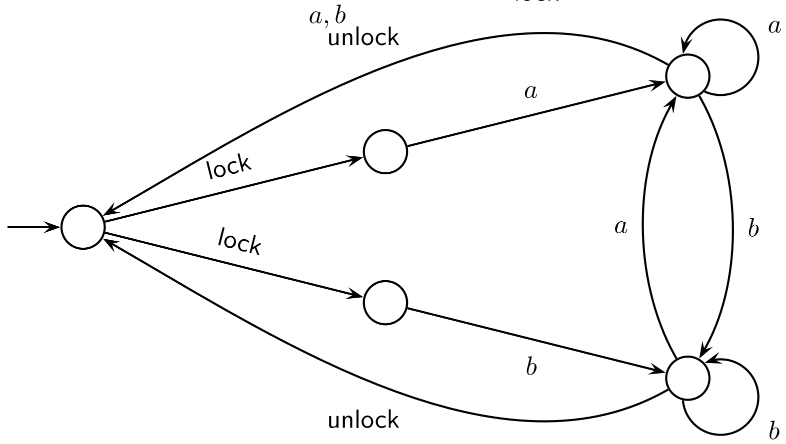
2.



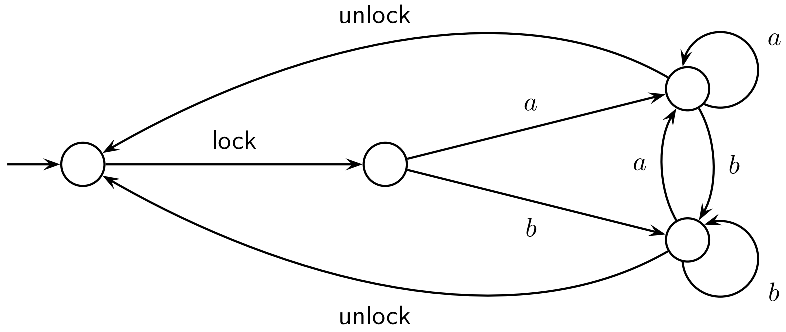
3.


lock
unlock
lock
a, b

4.


unlock
a
a
lock
lock
a
b
b
b
unlock

5.


unlock
a
a
lock
a
b
b
b
unlock

2. Concurrency and Interaction

Complex systems are generally composed of several components that act concurrently and interact with each other. This chapter deals with the issue of concurrency and interaction by introducing the notion of parallel composition of transition systems. First of all, we explain informally what parallel composition of transition systems is and give a simple example of its use in describing process behaviour (Section 2.1). After that, we define the notion of parallel composition of transition systems in a mathematically precise way (Section 2.2). For a better understanding, we next investigate the connections between the notion of parallel composition of transition systems and the more familiar notion of parallel execution of programs (Section 2.3). We also describe a typical example of a real-life system composed of components that act concurrently and interact with each other, viz. a simple data communication protocol, using parallel composition of transition systems (Section 2.4). Finally, we have another look at trace equivalence and bisimulation equivalence (Section 2.5). For the interested reader, we consider a different structure to describe interacting processes, called Petri nets (Section 2.6), and we relate the notion of parallel composition of transition systems with the notion of parallel composition of nets (Section 2.7).

2.1 Informal explanation

Sending a message to another component and receiving a message from another component are typical examples of the kinds of actions that are performed by a component of a system in order to interact with other components that act concurrently. Synchronous communication of a message between two components is a typical example of an interaction that takes place when a send action of one component and a matching receive action of the other component are performed synchronously. When two actions are performed synchronously, those actions cannot be observed seperately. Therefore, the intuition is that only one action is left when two actions are performed synchronously. For instance, when a send action and a matching receive action are performed synchronously, only a communication action can be observed. It does not have to be the case that any two actions can be performed synchronously. Usually, two actions can be performed synchronously only if they can establish an interaction. That is, for example, not the case for two send actions.

Now consider the use of transition systems in describing the behaviour of systems. In the case where a system is composed of components that act concurrently and interact which each other, we would like to reflect the composition in the description of the behaviour of the system. That is, we would like to use transition systems to describe the behaviour of the components and to be able to describe the behaviour of the whole system by expressing that its transition system is obtained from the transition systems describing the behaviour of the components by applying a certain operation to those transition systems. Parallel composition of transition systems as introduced in this chapter serves this purpose. The intuition is that the parallel composition of two transition systems T and T' can perform at each stage any action that T can perform next, any action that T' can perform next, and any action that results from synchronously performing an action that T can perform next and an action that T' can perform next. Parallel composition does not prevent actions that can be performed synchronously from being performed on their own. In order to prevent certain actions from being performed on their own, we introduce a seperate operation on transition systems, called encapsulation. The reason why parallel composition and encapsulation are not combined in a single operation will be explained later at the end of Section 2.2. Here is an example of the use of parallel composition and encapsulation in describing the behaviour of systems composed of components that act concurrently and interact which each other.

Example 2.1.1 (Bounded buffers). We consider the system composed of two bounded buffers, buffer 1 and buffer 2 with capacities l_1 and l_2 respectively, where each datum removed from the data kept in buffer 1 is simultaneously added to the data kept in buffer 2. In this way, data from buffer 1 is transferred to buffer 2. We start from the bounded buffers from Example 1.1.3. In the case of buffer 1, we rename the actions $\mathsf{add}(d)$ and $\mathsf{rem}(d)$ into $\mathsf{add}_1(d)$ and $\mathsf{rem}_1(d)$, respectively. In the case of buffer 2, we rename the actions $\mathsf{add}(d)$ and $\mathsf{rem}(d)$ into $\mathsf{add}_2(d)$ and $\mathsf{rem}_2(d)$, respectively. In this way, we can distinguish between the action of adding a datum to the data kept in one buffer and the action of adding the same datum to the data kept in the other buffer, as well as between the action of removing a datum from the data kept in one buffer and the action of removing the same datum from the data kept in the other buffer.

The renamings yield the following. As states of bounded buffer i, $i = 1, 2$, with capacity l_i, we have the sequences of data of which the length is not greater than l_i. As initial state, we have the empty sequence. As actions, we have $\mathsf{add}_i(d)$ and $\mathsf{rem}_i(d)$ for each datum d. As transitions of bounded buffer i, we have the following:

- for each datum d and each state σ that has a length less than l_i, a transition $\sigma \xrightarrow{\mathsf{add}_i(d)} d\,\sigma$;

- for each datum d and each state $\sigma\, d$, a transition $\sigma\, d \xrightarrow{\mathsf{rem}_i(d)} \sigma$.

In Figure 2.1, the transition systems for the buffers are given for the case that both have capacity 1. In the case where, for each datum d, the actions

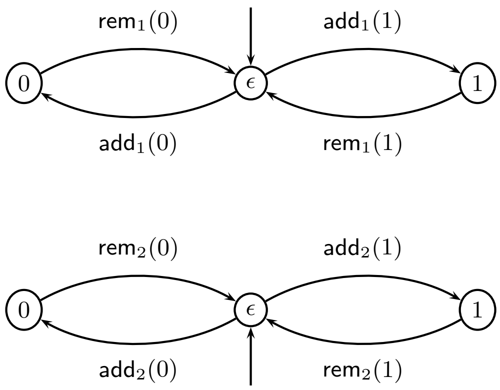


Fig. 2.1. Transition systems for the bounded buffers

$\mathsf{rem}_1(d)$ and $\mathsf{add}_2(d)$ can be performed synchronously, and $\mathsf{trf}(d)$ (transfer d) is the action left when these actions are performed synchronously, parallel composition of buffer 1 and buffer 2 results in the following transition system. As states, we have pairs (σ_1, σ_2) where σ_i $(i = 1, 2)$ is a sequence of data of which the length is not greater than l_i. State (σ_1, σ_2) is the state in which the sequence of data σ_i $(i = 1, 2)$ is kept in buffer i. As initial state, we have (ϵ, ϵ). As actions, we have $\mathsf{add}_i(d)$, $\mathsf{rem}_i(d)$ and $\mathsf{trf}(d)$ for each datum d and $i = 1, 2$. As transitions, we have the following:

- for each datum d and each state (σ_1, σ_2) with the length of σ_1 less than l_1, a transition $(\sigma_1, \sigma_2) \xrightarrow{\mathsf{add}_1(d)} (d\, \sigma_1, \sigma_2)$;
- for each datum d and each state (σ_1, σ_2) with the length of σ_2 less than l_2, a transition $(\sigma_1, \sigma_2) \xrightarrow{\mathsf{add}_2(d)} (\sigma_1, d\, \sigma_2)$;
- for each datum d and each state $(\sigma_1\, d, \sigma_2)$, a transition $(\sigma_1\, d, \sigma_2) \xrightarrow{\mathsf{rem}_1(d)} (\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1, \sigma_2\, d)$, a transition $(\sigma_1, \sigma_2\, d) \xrightarrow{\mathsf{rem}_2(d)} (\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1\, d, \sigma_2)$ with the length of σ_2 less than l_2, a transition $(\sigma_1\, d, \sigma_2) \xrightarrow{\mathsf{trf}(d)} (\sigma_1, d\, \sigma_2)$.

This transition system is represented graphically in Figure 2.2 for the case where $l_1 = l_2 = 1$ and the only data involved are the natural numbers 0 and 1. For each datum d, actions $\mathsf{rem}_1(d)$ and $\mathsf{add}_2(d)$ can still be performed on their own. Encapsulation with respect to these actions prevents them from

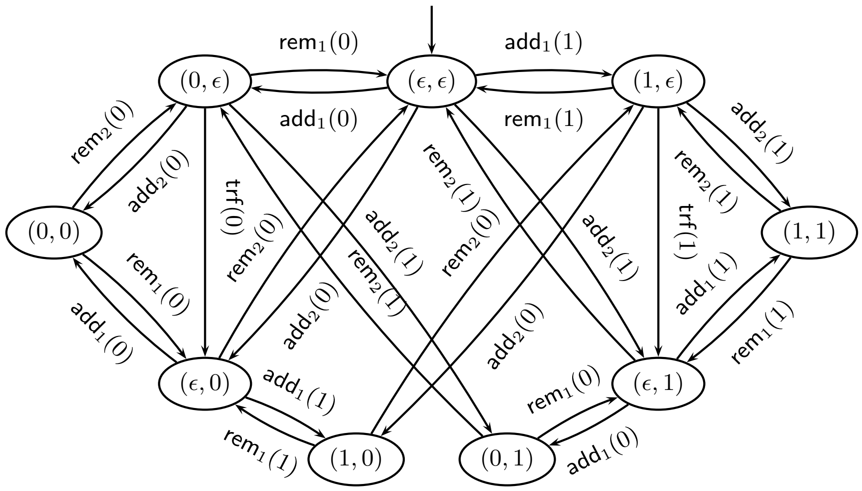


Fig. 2.2. Transition system for parallel composition of bounded buffers

being performed on their own, i.e. it results in the following transition system. We have the same states as before. As actions, we have $\mathsf{add}_1(d)$, $\mathsf{rem}_2(d)$ and $\mathsf{trf}(d)$ for each datum d. As transitions, we have the following:

- for each datum d and each state (σ_1, σ_2) with the length of σ_1 less than l_1, a transition $(\sigma_1, \sigma_2) \xrightarrow{\mathsf{add}_1(d)} (d\,\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1, \sigma_2\,d)$, a transition $(\sigma_1, \sigma_2\,d) \xrightarrow{\mathsf{rem}_2(d)} (\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1\,d, \sigma_2)$ with the length of σ_2 less than l_2, a transition $(\sigma_1\,d, \sigma_2) \xrightarrow{\mathsf{trf}(d)} (\sigma_1, d\,\sigma_2)$.

This transition system is represented graphically in Figure 2.3 for the case where $l_1 = l_2 = 1$ and the only data involved are the natural numbers 0 and 1. So encapsulation is needed to prevent that the actions $\mathsf{rem}_1(d)$ and $\mathsf{add}_2(d)$ do not lead to transfer of datum d from buffer 1 to buffer 2. The transition system obtained from the two bounded buffers by parallel composition and encapsulation would be bisimulation equivalent (see Section 1.5) to a bounded buffer with capacity $l_1 + l_2$ if we could abstract from the internal transfer actions $\mathsf{trf}(d)$. Abstraction from internal actions is one of the issues treated in the remaining chapters of this book.

Although systems composed of bounded buffers that act concurrently and interact with each other as described above actually arise in computer-based systems, they are not regarded as typical examples of real-life computer-based systems composed of components that act concurrently and interact with each other. Later, in Section 2.4, we give a fairly typical example, viz. a simple data communication protocol known as the ABP (Alternating Bit Protocol).

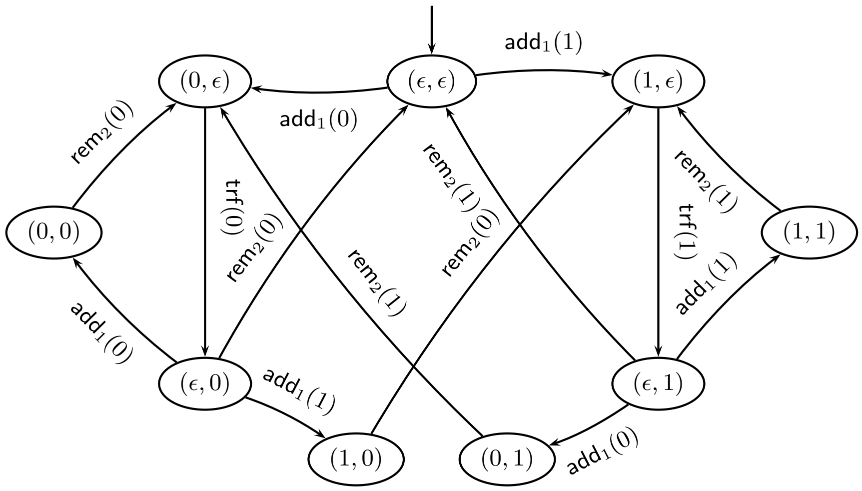


Fig. 2.3. Transition system for encapsulation of two parallel bounded buffers

2.2 Formal definitions

With the previous section, we have prepared the way for the formal definitions of the notions of parallel composition of transition systems and encapsulation of a transition system.

Whether two actions can be performed synchronously, and if so what action is left when they are performed synchronously, is mathematically represented by a communication function. Here is the definition of a communication function.

Definition 2.2.1. Let A be a set of actions. A *communication function* on A is a partial function $\gamma : A \times A \to A$ satisfying for $a, b, c \in A$:

- if $\gamma(a, b)$ is defined, then $\gamma(b, a)$ is defined and $\gamma(a, b) = \gamma(b, a)$;
- if $\gamma(a, b)$ and $\gamma(\gamma(a, b), c)$ are defined, then $\gamma(b, c)$ and $\gamma(a, \gamma(b, c))$ are defined and $\gamma(\gamma(a, b), c) = \gamma(a, \gamma(b, c))$.

The reason for the first condition is evident: there should be no difference between performing a and b synchronously and performing b and a synchronously. The reason for the second condition is essentially the same, but for the case where more than two actions can be performed synchronously. Let us give an example to illustrate that it is straightforward to define the communication function needed.

Example 2.2.1. We consider again the parallel composition of bounded buffers from Example 2.1.1. In that example, for each datum d, the actions $\mathsf{rem}_1(d)$ and $\mathsf{add}_2(d)$ can be performed synchronously, and $\mathsf{trf}(d)$ is the action left when these actions are performed synchronously. This is simply represented by the communication function γ defined such that $\gamma(\mathsf{rem}_1(d), \mathsf{add}_2(d)) = \gamma(\mathsf{add}_2(d), \mathsf{rem}_1(d)) = \mathsf{trf}(d)$ for each datum d, and it is undefined otherwise.

Let us now look at the formal definitions of parallel composition and encapsulation.

Definition 2.2.2. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ be transition systems. Let γ be a communication function on a set of actions that includes $A \cup A'$. The *parallel composition* of T and T' under γ, written $T \parallel_\gamma T'$, is the transition system $(S'', A'', \rightarrow'', \downarrow'', s''_0)$ where

- $S'' = S \times S'$;
- $A'' = A \cup A' \cup \{\gamma(a, a') \mid a \in A, a' \in A', \gamma(a, a') \text{ is defined}\}$;
- $\rightarrow''$ is the smallest subset of $S'' \times A'' \times S''$ such that:
  - if $s_1 \xrightarrow{a} s_2$ and $s' \in S'$, then $(s_1, s') \xrightarrow{a}'' (s_2, s')$;
  - if $s'_1 \xrightarrow{b}' s'_2$ and $s \in S$, then $(s, s'_1) \xrightarrow{b}'' (s, s'_2)$;
  - if $s_1 \xrightarrow{a} s_2$, $s'_1 \xrightarrow{b}' s'_2$ and $\gamma(a, b)$ is defined, then $(s_1, s'_1) \xrightarrow{\gamma(a,b)}'' (s_2, s'_2)$;
- $\downarrow'' = \downarrow \times \downarrow'$;
- $s''_0 = (s_0, s'_0)$.

In the above definition, the transition relation $\rightarrow''$ could have been given by means of

$$\begin{aligned}
\rightarrow'' = {} & \{((s_1, s'), a, (s_2, s')) \mid (s_1, a, s_2) \in \rightarrow, s' \in S'\} \\
\cup {} & \{((s, s'_1), b, (s, s'_2)) \mid (s'_1, b, s'_2) \in \rightarrow', s \in S\} \\
\cup {} & \{((s_1, s'_1), \gamma(a, b), (s_2, s'_2)) \mid (s_1, a, s_2) \in \rightarrow, \\
& \qquad (s'_1, b, s'_2) \in \rightarrow', \\
& \qquad \gamma(a, b) \text{ defined } \}
\end{aligned}$$

as well.

In describing systems that are composed in parallel of more than two subsystems, we use the convention of association to the left for parallel composition to reduce the number of parentheses, e.g. we write $T_1 \parallel_\gamma T_2 \parallel_{\gamma'} T_3$ for $(T_1 \parallel_\gamma T_2) \parallel_{\gamma'} T_3$. Justification for this convention will be provided later.

Example 2.2.2. The transition systems for the two buffers with capacity 1 are given by $T = (S, A, \rightarrow, \downarrow, s_0)$ where

$$\begin{aligned}
S & = \{\epsilon, 0, 1\}, \\
A & = \{\mathsf{add}_1(d), \mathsf{rem}_1(d) \mid d \in D\}, \\
\rightarrow & = \{(\epsilon, \mathsf{add}_1(d), d), (d, \mathsf{rem}_1(d), \epsilon) \mid d \in D\}, \\
\downarrow & = \emptyset, \\
s_0 & = \epsilon,
\end{aligned}$$

and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ where

$$\begin{aligned}
S' & = \{\epsilon, 0, 1\}, \\
A' & = \{\mathsf{add}_2(d), \mathsf{rem}_2(d) \mid d \in D\}, \\
\rightarrow' & = \{(\epsilon, \mathsf{add}_2(d), d), (d, \mathsf{rem}_2(d), \epsilon) \mid d \in D\}, \\
\downarrow' & = \emptyset, \\
s'_0 & = \epsilon,
\end{aligned}$$

respectively. We compute the parallel composition of T and T' under γ, the communication function from Example 2.2.1. The set of states of $T \parallel_\gamma T'$ is obtained by constructing the Cartesian product of the sets of states of T and T':

$$S'' = S \times S' = \{(\epsilon, \epsilon), (0, \epsilon), (1, \epsilon), (\epsilon, 0), (0, 0), (1, 0), (\epsilon, 1), (0, 1), (1, 1)\}.$$

The set of actions is the union of the sets of actions of T and T' and additionally any actions that result from communication between the transition systems:

$$\begin{aligned} A'' &= A \cup A' \cup \{\gamma(a, a') \mid a \in A, a' \in A', \gamma(a, a') \text{ defined }\} \\ &= \{\mathsf{add}_1(d), \mathsf{rem}_1(d), \mathsf{add}_2(d), \mathsf{rem}_2(d), \mathsf{trf}(d) \mid d \in D\}. \end{aligned}$$

The transitions are obtained by copying the transitions of transition system T and those of transition system T'. These are given by

$$\begin{aligned} &\{((s_1, s'), a, (s_2, s')) \mid (s_1, a, s_2) \in \rightarrow, s' \in S'\} \\ = &\{((\epsilon, s'), \mathsf{add}_1(d), (d, s')), ((d, s'), \mathsf{rem}_1(d), (\epsilon, s')) \mid d \in D, s' \in S'\} \end{aligned}$$

and

$$\begin{aligned} &\{((s, s_1'), a, (s, s_2')) \mid (s_1', a, s_2') \in \rightarrow', s \in S\} \\ = &\{((s, \epsilon), \mathsf{add}_2(d), (s, d)), ((s, d), \mathsf{rem}_2(d), (s, \epsilon)) \mid d \in D, s \in S\} \end{aligned}$$

respectively. To this the transitions that result from communication have to be added. The only possible communication is between the actions $\mathsf{rem}_1(d)$ and $\mathsf{add}_2(d)$ (for each $d \in D$). The only state (in T) from which $\mathsf{rem}_1(d)$ is possible is the state d and the only state (in T') from which $\mathsf{add}_2(d)$ is possible is the state ϵ. Hence, the following transitions result from communication:

$$\begin{aligned} &\{((s_1, s_1'), \gamma(a, a'), (s_2, s_2')) \mid (s_1, a, s_2) \in \rightarrow, (s_1', a', s_2') \in \rightarrow', \\ &\qquad\qquad\qquad\qquad\qquad\qquad \gamma(a, a') \text{ defined}\} \\ = &\{((d, \epsilon), \mathsf{trf}(d), (\epsilon, d)) \mid d \in D\}. \end{aligned}$$

So, we have obtained

$$\begin{aligned} \rightarrow'' = &\{((\epsilon, s'), \mathsf{add}_1(d), (d, s')), ((d, s'), \mathsf{rem}_1(d), (\epsilon, s')) \mid d \in D, s' \in S'\} \\ \cup &\{((s, \epsilon), \mathsf{add}_2(d), (s, d)), ((s, d), \mathsf{rem}_2(d), (s, \epsilon)) \mid d \in D, s \in S\} \\ \cup &\{((d, \epsilon), \mathsf{trf}(d), (\epsilon, d)) \mid d \in D\}. \end{aligned}$$

As T and T' do not have successfully terminating states, also their parallel composition does not have successfully terminating states:

$$\downarrow'' = \downarrow \times \downarrow' = \emptyset \times \emptyset = \emptyset.$$

The initial state is the pair of initial states of T and T':

$$s_0'' = (s_0, s_0').$$

The reader can easily verify that the transition system depicted in Figure 2.2 corresponds to the transition system T'' as defined here.

If parallel composition is applied to transition systems T and T' where all states are reachable, then the states of their parallel composition will result in a transition system where all states are reachable as well.

Property 2.2.1. Let T and T' be transition systems such that all states of T and T' are reachable. Then, for arbitrary communication function γ, we have $\mathrm{reach}(T \parallel_\gamma T') = \mathrm{reach}(T) \times \mathrm{reach}(T')$.

Similarly, if parallel composition is applied to transition systems T and T' where all actions are used in reaching states, then the parallel composition of such transition systems will result in a transition system where all actions are used in reaching states.

Property 2.2.2. Let T and T' be transition systems such that $\mathrm{act}(T) = \mathrm{act}(\mathrm{red}(T))$ and $\mathrm{act}(T') = \mathrm{act}(\mathrm{red}(T'))$. Then, for arbitrary communication function γ, we have $\mathrm{act}(T \parallel_\gamma T') = \mathrm{act}(\mathrm{red}(T)) \cup \mathrm{act}(\mathrm{red}(T')) \cup \{\gamma(a, a') \mid a \in \mathrm{act}(T), a' \in \mathrm{act}(T'), \gamma(a, a') \text{ defined}\}$.

This means that by applying parallel composition no states and actions become unreachable. This is not the case for all compositions on transition systems that we encounter. For example, in encapsulating certain actions it is possible that some states and actions that used to be reachable are not so anymore. Such operations can be defined in two equally reasonable ways.

- A definition such that the resulting transition system is reduced.
- A definition such that the resulting transition system is not necessarily reduced. It may contain unreachable states and actions.

From a readability point of view, the second type of definition is easier. In this book, we always give definitions of compositions of the second type.

Definition 2.2.3 (Encapsulation). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. Let H be a set of actions. The *encapsulation* of T with respect to H, written $\partial_H(T)$, is the transition system $(S', A', \rightarrow', \downarrow', s'_0)$ where

- $S' = S$;
- $A' = A$;
- $\rightarrow' = \rightarrow \cap (S \times (A \setminus H) \times S)$;
- $\downarrow' = \downarrow$;
- $s'_0 = s_0$.

Example 2.2.3. The encapsulation of the two parallel bounded buffers is obtained from the transition system T'' from Example 2.2.2 as follows. The states and actions of $\partial_H(T'')$ with $H = \{\mathsf{rem}_1(d), \mathsf{add}_2(d) \mid d \in D\}$ are simply the states and actions of T'', respectively. The transitions of $\partial_H(T'')$ are obtained by removing all transitions with an action from H from the transitions of T''. The successfully terminating states and the initial state remain the same as well. The resulting transition system is depicted in Figure 2.3.

In many applications, $\gamma(\gamma(a,b),c)$ is undefined for all $a,b,c \in A$. That case is called *handshaking communication*. We introduce some standardized terminology and notation for handshaking communication. Transition systems send, receive and communicate data at *ports*. If a port is used for communication between two transition systems, it is called *internal*. Otherwise, it is called *external*. We write:

$\mathsf{s}_i(d)$ for the action of sending datum d at port i;
$\mathsf{r}_i(d)$ for the action of receiving datum d at port i;
$\mathsf{c}_i(d)$ for the action of communicating datum d at port i.

Assuming a set of data D, the communication function is defined such that

$$\gamma(\mathsf{s}_i(d), \mathsf{r}_i(d)) = \gamma(\mathsf{r}_i(d), \mathsf{s}_i(d)) = \mathsf{c}_i(d)$$

for all $d \in D$, and it is undefined otherwise.

It is important to remember that handshaking communication is just one kind of communication. It is not required that $\gamma(\gamma(a,b),c)$ is undefined for all $a,b,c \in A$. Here is an example of another kind of communication.

Example 2.2.4. We consider a kind of communication in which three transition systems participate. A communication of this kind takes place by synchronously performing one send action and two matching receive actions. Using a notation which is reminiscent of the standardized notation for handshaking communication, this ternary kind of communication can be represented by a communication function as follows. Assuming a set of data D, the communication function is defined such that

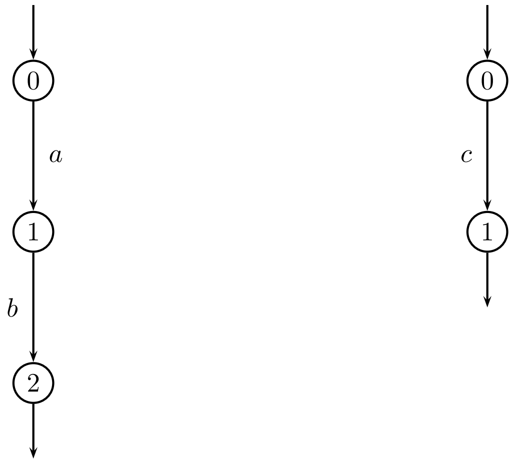
$$\begin{aligned}
&\gamma(\mathsf{r}_i(d), \mathsf{r}_i(d)) = \mathsf{rr}_i(d)\ , \\
&\gamma(\mathsf{s}_i(d), \mathsf{r}_i(d)) = \gamma(\mathsf{r}_i(d), \mathsf{s}_i(d)) = \mathsf{sr}_i(d)\ , \\
&\gamma(\mathsf{s}_i(d), \mathsf{rr}_i(d)) = \gamma(\mathsf{rr}_i(d), \mathsf{s}_i(d)) = \mathsf{c}_i(d)\ , \\
&\gamma(\mathsf{sr}_i(d), \mathsf{r}_i(d)) = \gamma(\mathsf{r}_i(d), \mathsf{sr}_i(d)) = \mathsf{c}_i(d)\ ,
\end{aligned}$$

for all $d \in D$, and it is undefined otherwise. The actions $\mathsf{sr}_i(d)$ and $\mathsf{rr}_i(d)$ represent the possible partial communications.

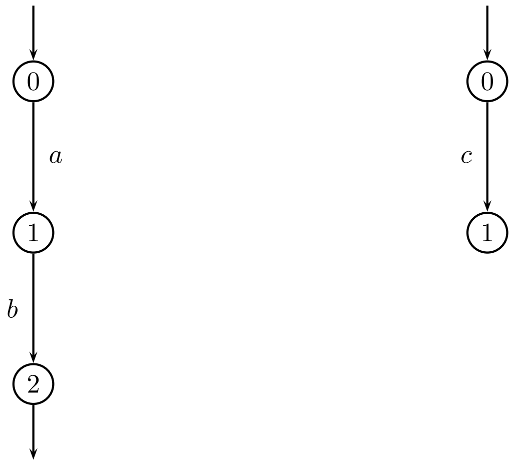
An important thing to note about the kind of communication treated in the preceding example is the following. If parallel composition and encapsulation were combined in a single operation that prevents actions that can be performed synchronously from being performed on their own, this kind of communication would be excluded.

Exercise 2.2.1. Let γ be a communication function such that $\gamma(a,b)$ is undefined for all actions a and b. Compute the parallel composition of the following pairs of transition systems.

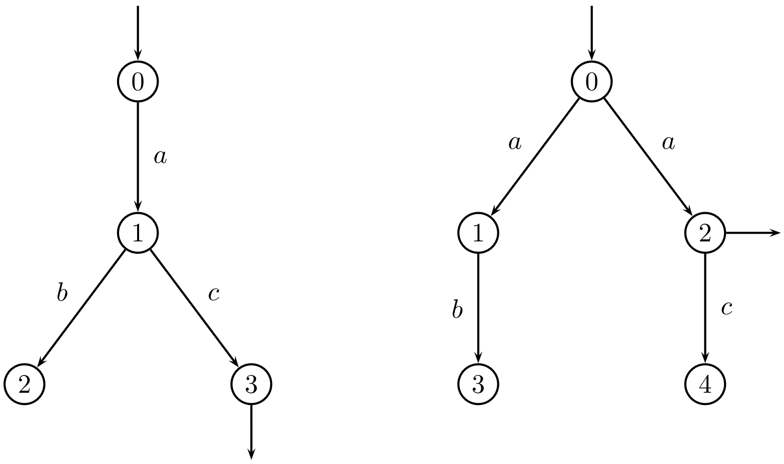
1.



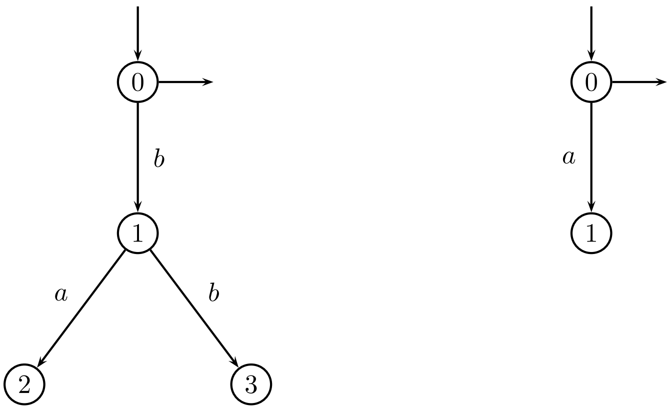
2.



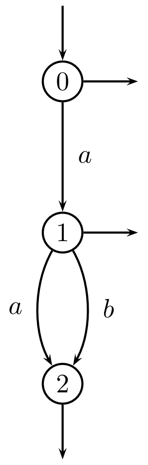
3.

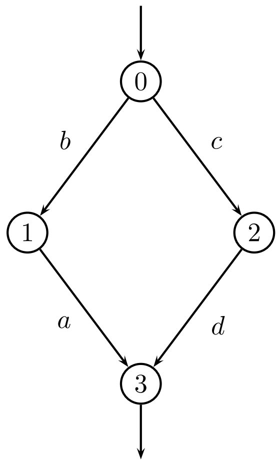


Exercise 2.2.2. Let γ be a communication function such that $\gamma(a, b) = c$ and γ is undefined otherwise. Compute the parallel composition of the following transition systems.

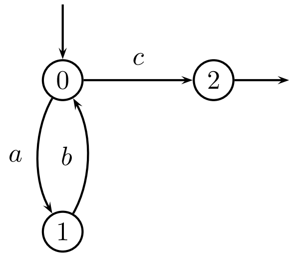
1. 

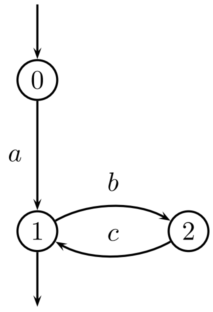
2.





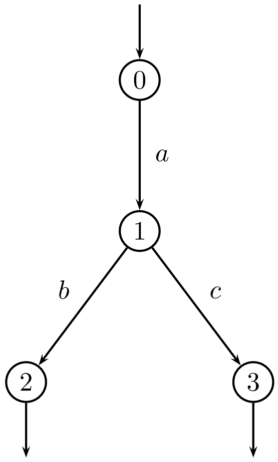
3.

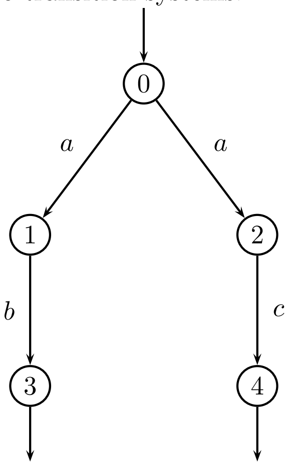




Exercise 2.2.3. Compute, for each of the transition systems from the previous two exercises, their encapsulation with respect to the action b. Do the same, for the transition systems that result from the parallel compositions.

Exercise 2.2.4. Consider the following two transition systems.





These transition systems are language equivalent and trace equivalent.

1. Compute, for each of these, the encapsulation with respect to action c.
2. Are the resulting transition systems language equivalent?
3. Are the resulting transition systems trace equivalent?

Exercise 2.2.5. Suppose that we want to describe a communication protocol for the exchange of data from a set D from a *sender* to a *receiver*. The sender repeats the following behaviour. It receives the datum to be communicated, say d, from the environment through an action $\mathsf{r}_1(d)$. Upon receipt of such a datum, the sender sends the datum to the receiver by means of the action $\mathsf{s}_2(d)$. The sender then waits for the receipt of an acknowledgement $\mathsf{r}_2(\mathsf{ack})$ from the receiver of the datum.

We consider three different receiver processes.

- Receiver process 1: upon receipt of a datum ($\mathsf{r}_2(d)$), the receiver first sends the datum to its environment ($\mathsf{s}_3(d)$) and then sends an acknowledgement to the sender ($\mathsf{s}_2(\mathsf{ack})$. This behaviour is repeated indefinitely.
- Receiver process 2: upon receipt of a datum ($\mathsf{r}_2(d)$), the receiver first sends an acknowledgement to the sender ($\mathsf{s}_2(\mathsf{ack})$) and then sends the datum to its environment ($\mathsf{s}_3(d)$). This behaviour is repeated indefinitely.
- Receiver process 3: upon receipt of a datum ($\mathsf{r}_2(d)$), the receiver sends an acknowledgement to the sender ($\mathsf{s}_2(\mathsf{ack})$ and sends the datum to its environment ($\mathsf{s}_3(d)$). The order in which these two actions take place is unspecified and hence non-deterministic. This behaviour is repeated indefinitely.

1. Give a transition system for the sender process and the three versions of the receiver process. Also draw these transition systems for the case that $D = \{0, 1\}$.
2. For each combination of the sender process and a receiver process compute their parallel composition where the communication function is given by $\gamma(\mathsf{s}_2(\mathsf{ack}), \mathsf{r}_2(\mathsf{ack})) = \mathsf{c}_2(\mathsf{ack})$ and $\gamma(\mathsf{s}_2(d), \mathsf{r}_2(d)) = \mathsf{c}_2(d)$ for each $d \in D$ and γ is undefined otherwise. Draw the resulting transition systems for the case that $D = \{0, 1\}$.
3. For each of the above parallel compositions compute the encapsulation with respect to the set of actions $H = \{\mathsf{s}_2(d), \mathsf{r}_2(d) \mid d \in D \cup \{\mathsf{ack}\}\}$. Draw the resulting transition systems for the case that $D = \{0, 1\}$.
4. Explain the differences in the transition systems that result from parallel composition and encapsulation.

Exercise 2.2.6. Suppose that we are given a communication function γ. Prove that from the conditions that are satisfied by γ it follows that: if $\gamma(a, b)$ and $\gamma(\gamma(a, b), c)$ are defined, then $\gamma(a, c)$ and $\gamma(\gamma(a, c), b)$ are defined and $\gamma(\gamma(a, b), c) = \gamma(\gamma(a, c), b)$.

Exercise 2.2.7. We define a transition system T to have deadlock, notation *deadlock*(T), if and only if it has a deadlock state. Prove the following statements, or give a counterexample:

1. if $T \cong T'$ then *deadlock*(T) if and only if *deadlock*(T');
2. if $T \equiv_{\mathrm{l}} T'$ then *deadlock*(T) if and only if *deadlock*(T');
3. if $T \equiv_{\mathrm{tr}} T'$ then *deadlock*(T) if and only if *deadlock*(T');
4. if $T \leftrightarroweq T'$ then *deadlock*(T) if and only if *deadlock*(T').

Exercise 2.2.8. Prove the following properties for arbitrary transition systems T and arbitrary sets of actions H_1 and H_2:

1. $\partial_{\emptyset}(T) = T$;
2. $\partial_{H_1}(\partial_{H_2}(T)) = \partial_{H_1 \cup H_2}(T)$.

2.3 Programs, machines and parallel composition

For about twenty years, there are programming languages in which it can be expressed that a number of (sequential) subprograms must be executed in parallel. What exactly does that mean? Can it be described in a straightforward way by means of transition systems using parallel composition? It turns out that the answers to these questions do not only depend on whether one abstracts from the processing of actions by a machine, but also on the way in which the programming language used supports interaction between subprograms executed in parallel. Roughly speaking, the basic ways of interaction are:

- by synchronous communication, i.e. communication where the sending subprogram must wait till each receiving subprogram (usually one) is ready to participate in the communication;
- by asynchronous communication, i.e. communication where the sending subprogram does not have to wait till each receiving subprogram (usually one) is ready to participate in the communication;
- via shared variables, i.e. program variables to which more than one subprogram has access.

Some programming languages support a combination of these basic ways. An important thing to note is that, in virtually all programming languages that support synchronous or asynchronous communication, the data communicated may depend on the values of program variables.

In this section, we will look at the questions posed above in more detail. We do so primarily to acquire a better understanding of the notion of parallel composition of transition systems. In line with Section 1.3, we like to abstract initially from how the actions performed by subprograms are processed by a machine. That is, we like to focus initially on the flow of control or abstract execution.

Let T_{P_1} and T_{P_2} be transition systems describing the behaviour of two subprograms P_1 and P_2 upon abstract execution. If the programming language does not support synchronous communication, then the behaviour of P_1 and P_2 upon parallel abstract execution can be described by

$$T_{P_1} \parallel_\gamma T_{P_2} \ ,$$

where γ is undefined for any two actions.

In order to illustrate by an example how this works, we have to choose a programming language first. Our choice is a simple extension of PASCAL introcuced by Ben-Ari back in 1982. The extension concerned simply permits to write statements of the form `COBEGIN P1; ...; Pn COEND`, where `P1`, ..., `Pn` are procedures defined in the program body to express that those procedures must be executed in parallel. Moreover, assignments and tests are indivisible and nothing else is indivisible. That is all. The extension does not support communication in a direct way. Interaction is only possible via shared variables. Let us now turn to the promised example.

Example 2.3.1 (Peterson's protocol). We consider a program implementing a simple mutual exclusion protocol. A mutual exclusion protocol concerns the exclusive access by components of a system to a shared resource while using that shared resource. As the saying is, a component is in its critical section while it is using the shared resource. We consider Peterson's protocol for guaranteeing that at most one component of a system is in its critical section. The protocol assumes that there are three shared variables `c0`, `c1` and `t`, with initial value `false`, `false` and `0`, respectively, and that all assignments and tests concerning these variables are indivisible.

The idea behind the protocol is as follows. The components have sequence numbers `0` and `1`. The value of `t` is the sequence number of the component that last started an attempt to enter its critical section. That the value of `c0` is `false` signifies that component `0` is not in its critical section; and that the value of `c1` is `false` signifies that component `1` is not in its critical section. If component `0` intends to enter its critical section it must assign the value `true` to `c0` before it checks the value of `c1`, to prevent situations in which the value of both variables is `true`. Analogously for component `1`. This may lead to situations in which the value of both `c0` and `c1` is `true`. In order to prevent deadlock in that case, each component checks whether the other one last started an attempt to enter its critical section, and the one of which the check succeeds actually enters its critical section.

In the program that we will give below, we have taken the most simple critical sections for which the mutual exclusion problem is not trivial: a sequence of two indivisible statements. Here is the program.

```
PROGRAM peterson;
VAR
   c0, c1: boolean;
   t: 0..1;

PROCEDURE p0;
BEGIN
   WHILE true DO
```

```
        BEGIN
           c0 := true;
           t := 0;
           REPEAT UNTIL c1 = false OR t = 1;
           enter0;  {enter critical section}
           leave0;  {leave critical section}
           c0 := false;
        END
END

PROCEDURE p1;
BEGIN
   WHILE true DO
      BEGIN
         c1 := true;
         t := 1;
         REPEAT UNTIL c0 = false OR t = 0;
         enter1;  {enter critical section}
         leave1;  {leave critical section}
         c1 := false;
      END
END

.
.
.

BEGIN
   c0 := false;
   c1 := false;
   t := 0;
   COBEGIN p0; p1 COEND
END
```

Actually, `enter0`, `leave0`, `enter1` and `leave1` are no real statements. They stand for arbitrary indivisible statements that use the shared resource.

The behaviour of the procedures `p0` and `p1` upon abstract execution can be described by transition systems in the same way as in Examples 1.3.1 and 1.3.2. As states, we have in either case the natural numbers 0 to 7, with 0 as initial state. As actions, we have in either case an action corresponding to each atomic statement of the procedure as well as each test of the procedure and its opposite. As transitions, we have the following in the case of `p0`:

$$
\begin{aligned}
&0 \xrightarrow{\mathsf{true}} 1,\ 1 \xrightarrow{\mathsf{c0 := true}} 2,\ 2 \xrightarrow{\mathsf{t := 0}} 3,\\
&3 \xrightarrow{\mathsf{NOT\ (c1 = false\ OR\ t = 1)}} 3,\ 3 \xrightarrow{\mathsf{c1 = false\ OR\ t = 1}} 4,\\
&4 \xrightarrow{\mathsf{enter0}} 5,\ 5 \xrightarrow{\mathsf{leave0}} 6,\ 6 \xrightarrow{\mathsf{c0 := false}} 0,\\
&0 \xrightarrow{\mathsf{NOT\ true}} 7;
\end{aligned}
$$

and the following in the case of `p1`:

$$
\begin{aligned}
&0 \xrightarrow{\mathsf{true}} 1,\ 1 \xrightarrow{\mathsf{c1 := true}} 2,\ 2 \xrightarrow{\mathsf{t := 1}} 3,\\
&3 \xrightarrow{\mathsf{NOT\ (c0 = false\ OR\ t = 0)}} 3,\ 3 \xrightarrow{\mathsf{c0 = false\ OR\ t = 0}} 4,\\
&4 \xrightarrow{\mathsf{enter1}} 5,\ 5 \xrightarrow{\mathsf{leave1}} 6,\ 6 \xrightarrow{\mathsf{c1 := false}} 0,\\
&0 \xrightarrow{\mathsf{NOT\ true}} 7
\end{aligned}
$$

Here, enter0, leave0, enter1 and leave1 are no real actions. They stand for the actions corresponding to the statements that `enter0`, `leave0`, `enter1` and `leave1` stand for.

The transition systems for the procedures `p0` and `p1` are represented graphically in Figure 2.4. We call these transition systems $T_{\mathtt{p0}}$ and $T_{\mathtt{p1}}$, respectively. The behaviour of the procedures `p0` and `p1` upon parallel abstract execution can be described as follows:

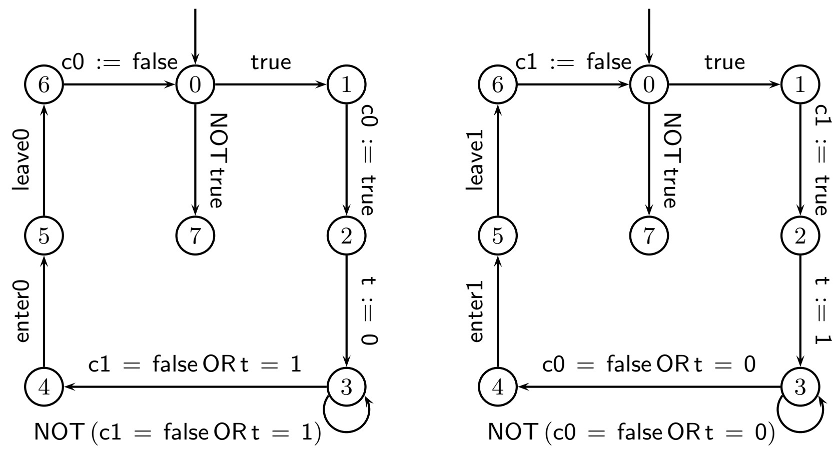


Fig. 2.4. Transition systems for Peterson's protocol

$$T_{\mathtt{p0}} \parallel_\gamma T_{\mathtt{p1}}$$

where the communication function γ is undefined for any two actions.

Notice that the preceding example is based on the idea that the parallel abstract execution of two subprograms can be reduced to arbitrary interleaving only, i.e. to performing again and again an action that one or the other of the two can perform next. This is obviously problematic in the presence of synchronous communication: simultaneously performing actions is not taken into account. However, if one abstracts from the processing of actions by a machine, there is also no alternative in the general case where the data communicated may depend on the values of program variables.

Let us now, like in Section 1.3, take into account how the actions performed by subprograms are processed by a machine and turn to the behaviour of subprograms upon parallel execution on a machine. We can describe the behaviour of machines on which subprograms are executed by transition systems as well. We will give a simple example illustrating this later. Let T_{P_1} and T_{P_2} be transition systems describing the behaviour of two subprograms P_1 and P_2 upon abstract execution. If we suppose that we also have the transition systems of the appropriate machines available, the behaviour of P_1 and P_2 upon parallel execution on a machine can in many cases best be described in one of the following ways, depending on the way in which the programming language used supports interaction between subprograms executed in parallel:

$$\partial_H((T_{P_1} \parallel_{\gamma'} T_{P_2}) \parallel_\gamma T_M)$$

or

$$\partial_H(\partial_{H_1}(T_{P_1} \parallel_\gamma T_{M_1}) \parallel_\gamma \partial_{H_2}(T_{P_2} \parallel_\gamma T_{M_2}))\ ,$$

where the communication function γ' is undefined for any two actions, and the communication function γ, the sets of actions H, H_1 and H_2, and the transition systems T_M, T_{M_1} and T_{M_2} all depend on the way in which the programming language used supports interaction between subprograms executed in parallel. The transition systems T_M, T_{M_1} and T_{M_2} are supposed to describe the behaviour of appropriate machines.

The first way of description applies if the programming language only supports shared variables as a means to interact. The second way of description applies if the programming language supports synchronous communication or asynchronous communication, but does not support shared variables. Synchronous communication can be fully represented by the communication function γ, while asynchronous communication cannot be fully represented by the communication function (as explained below). In the case where only shared variables are supported, P_1 and P_2 are executed on the same machine: T_M. In the cases where shared variables are not supported, P_1 and P_2 are executed on different machines: T_{M_1} and T_{M_2}, respectively. The machines process the actions performed by the subprograms. In the case of asynchronous communication, they are also involved in the communication between subprograms. In that case, each machine buffers the data sent to the subprogram that the machine executes till the subprogram consumes the data. Actually, the first way of description can be applied in the case of asynchronous communication as well, but it is rather clumsy.

The second way of description shows that, in the case where no abstraction from the processing of actions by a machine is made, parallel execution of subprograms corresponds directly to (encapsulated) parallel composition if synchronous communication or asynchronous communication is supported by the programming language used, and moreover shared variables are not supported. This makes it a compositional way of description, which has advantages in analysis. The compositionality is missing in the first way of description, which applies if only shared variables are supported.

Here is an example that illustrates how the behaviour of machines on which subprograms are executed can be described by transition systems.

Example 2.3.2. We consider again the program from Example 2.3.1 concerning Peterson's mutual exclusion protocol. The behaviour of a machine on which the procedures `p0` and `p1` can be executed in parallel, is described by a transition system as follows. As states of the machine, we have triples $(c0, c1, t)$, where $c0, c1 \in \mathbb{B}$ and $t \in \{0, 1\}$. These states can be viewed as follows: $(c0, c1, t)$ is the storage that keeps the values of the program variables `c0`, `c1`, and `t` in that order. The initial state is $(\mathsf{ff}, \mathsf{ff}, 0)$. As actions, we have an action corresponding to each atomic statement of the procedures as well as each test of the procedures and its opposite. However, these actions differ from the actions of the transition system describing the behaviour of the procedures upon abstract execution: the former actions are actions of processing the latter actions. The difference is indicated by overlining the former actions. As transitions, we have the following:

- for each $c0, c1, t$:
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{c0 := false}}} (\mathsf{ff}, c1, t)$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{c0 := true}}} (\mathsf{tt}, c1, t)$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{c1 := false}}} (c0, \mathsf{ff}, t)$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{c1 := true}}} (c0, \mathsf{tt}, t)$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{t := 0}}} (c0, c1, 0)$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{t := 1}}} (c0, c1, 1)$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{true}}} (c0, c1, t)$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{c1 = false\ OR\ t = 1}}} (c0, c1, t)$ if $c1 = \mathsf{ff}$ or $t = 1$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{NOT\ (c1 = false\ OR\ t = 1)}}} (c0, c1, t)$ if $c1 \neq \mathsf{ff}$ and $t \neq 1$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{c0 = false\ OR\ t = 0}}} (c0, c1, t)$ if $c0 = \mathsf{ff}$ or $t = 0$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{NOT\ (c0 = false\ OR\ t = 0)}}} (c0, c1, t)$ if $c0 \neq \mathsf{ff}$ and $t \neq 0$,
  - a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{enter0}}} (c0, c1, t)$,

– a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{leave0}}} (c0, c1, t)$,
– a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{enter1}}} (c0, c1, t)$,
– a transition $(c0, c1, t) \xrightarrow{\overline{\mathsf{leave1}}} (c0, c1, t)$.

Although the subprocesses that are executed on the machine never terminate successfully, still the machine itself should be able to deal with successfully terminating processes. Therefore, all states are considered to be successfully terminating. The transition system for the machine is represented graphically in Figure 2.5. We call this transition system T_M. The behaviour of the procedures `p0` and `p1` upon parallel execution on a machine can now be described as follows:

$$\partial_H((T_{\mathtt{p0}} \parallel_{\gamma'} T_{\mathtt{p1}}) \parallel_{\gamma} T_M)$$

where

$$H = \mathrm{act}(T_{\mathtt{p0}}) \cup \mathrm{act}(T_{\mathtt{p1}}) \cup \mathrm{act}(T_M) \ ,$$

the communication function γ' is undefined for any two actions, and the communication function γ is defined such that

$$\gamma(a, \overline{a}) = \gamma(\overline{a}, a) = a^*$$

for all actions $a \in \mathrm{act}(T_{\mathtt{p0}}) \cup \mathrm{act}(T_{\mathtt{p1}})$, and it is undefined otherwise. The transition system for the execution of Peterson's protocol on the machine is too large to depict.

Notice that most procedures, written in the same programming language as `p0` and `p1`, cannot be executed on the machine of which the behaviour is described by the transition system T_M presented above. This machine can only deal with actions that can possibly be performed by the procedures `p0` and `p1`. However, because all actions of the machine are prevented from being performed on their own, T_M can safely be replaced by a transition system for a machine that can also deal with actions that can possibly be performed by other procedures.

2.4 Example: Alternating bit protocol

Here is a fairly typical example of the use of parallel composition and encapsulation in describing the behaviour of systems composed of components that act concurrently and interact which each other. The example concerns the ABP (Alternating Bit Protocol).

The ABP is a simple data communication protocol based on positive and negative acknowledgements. Data are labeled with an alternating bit from $B = \{0, 1\}$. The sender either transmits a new datum or retransmits the most

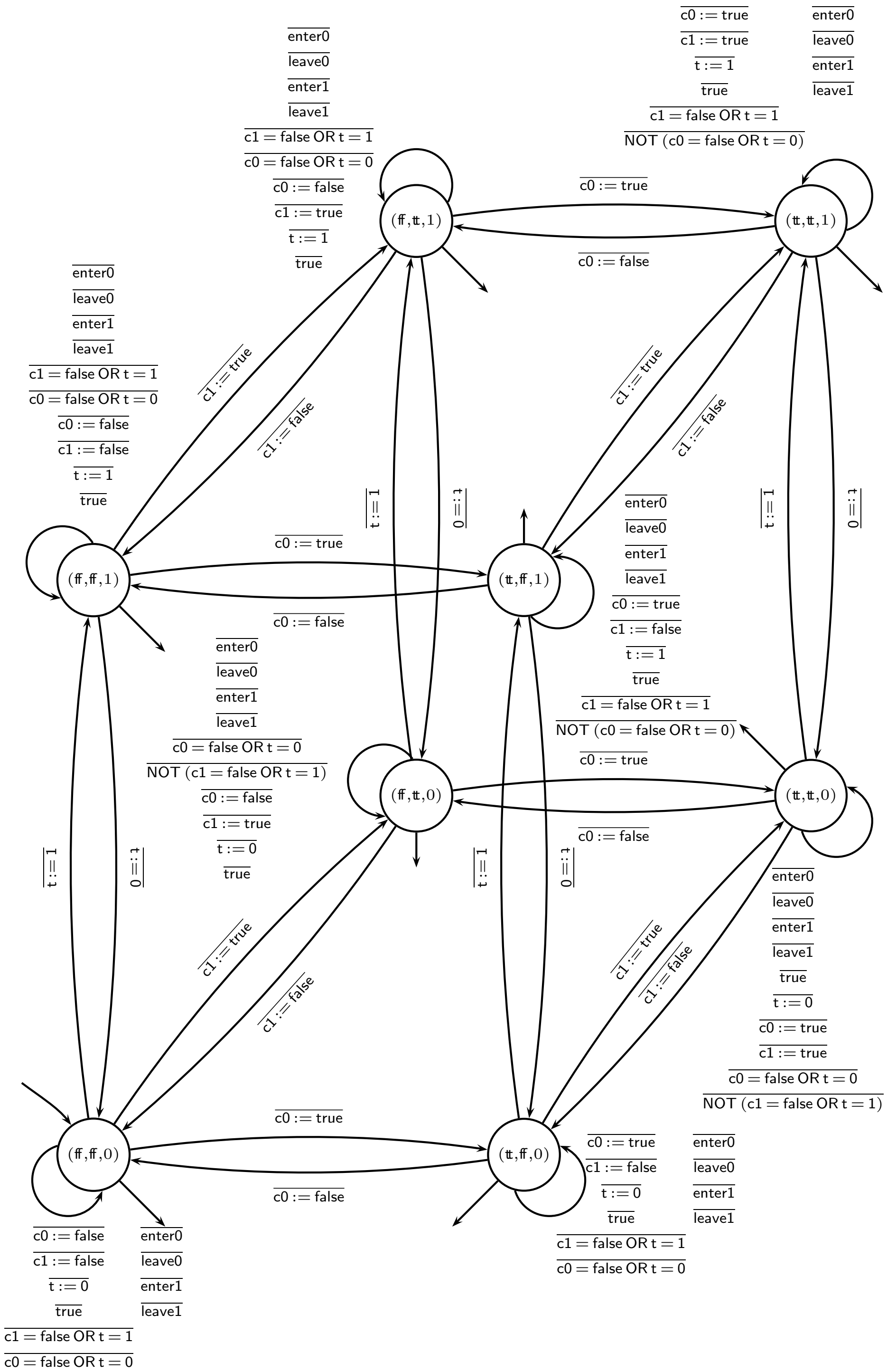


Fig. 2.5. Transition system for the machine executing Peterson's protocol

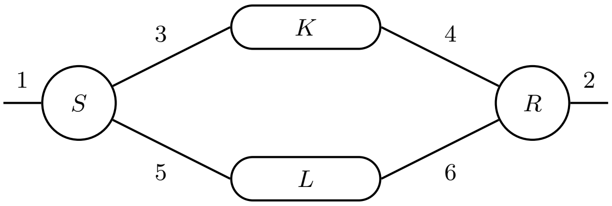


Fig. 2.6. Configuration of the ABP

recent datum depending on an acknowledgement represented by a bit. The alternating bit used with the most recent datum is considered to be a positive acknowledgement. The configuration of the ABP is shown in Figure 2.6. We have a sender process S, a receiver process R and two channels K and L. The process S waits until a datum d is offered at an external port (port 1). When a datum is offered at this port, S consumes it, packs it with an alternating bit b in a frame (d, b), and then delivers the frame at an internal port used for sending (port 3). Next, S waits until a bit b' is offered at an internal port used for receiving (port 5). When a bit is offered and it is the alternating bit b, S goes back to waiting for a datum. When a bit is offered and it is not the alternating bit b, S delivers the same frame again and goes back to waiting for a bit. The process S behaves the same when an error value is offered instead of a bit. The process R waits until a frame with a datum and an alternating bit (d, b) is offered at an internal port used for receiving (port 4). When a frame is offered at this port, R consumes it, unpacks it, and then delivers the datum d at an external port (port 2) if the alternating bit b is the right one and in any case the alternating bit b at an internal port for sending (port 6). When instead an error value is offered, R delivers the wrong bit. After that, R goes back to waiting for a frame, but the right bit changes if the alternating bit was the right one. The processes K and L pass on frames from an internal port of S to an internal port of R and bits from an internal port of R to an internal port of S, respectively. The processes K and L may corrupt frames and acknowledgements, respectively. In the case where this happens, K and L deliver an error value.

We assume a set of data D. Let $F = D \times B$ be the set of frames. For $d \in D$ and $b \in B$, we write d, b for the frame (d, b). For $b \in B$, we write $\overline{b}$ for the bit $1 - b$. We use the standardized notation for handshaking communication introduced in Section 2.2.

The behaviour of the sender S is described by a transition system as follows. As states of the sender, we have triples (d, b, i), where $d \in D \cup \{*\}$, $b \in B$ and $i \in \{0, 1, 2\}$, satisfying $d = *$ if and only if $i = 0$. State (d, b, i) is roughly a state in which the datum being passed on from the the sender to the receiver is d and the alternating bit is b. If $d = *$, no such datum is available. The initial state is $(*, 0, 0)$. As actions, we have $\mathsf{r}_1(d)$ for each

$d \in D$, $\mathsf{s}_3(f)$ for each $f \in F$, and $\mathsf{r}_5(b)$ for each $b \in B \cup \{*\}$. As transitions of the sender, we have the following:

- for each datum $d \in D$ and bit $b \in B$:
  - a transition $(*, b, 0) \xrightarrow{\mathsf{r}_1(d)} (d, b, 1)$,
  - a transition $(d, b, 1) \xrightarrow{\mathsf{s}_3(d,b)} (d, b, 2)$,
  - a transition $(d, b, 2) \xrightarrow{\mathsf{r}_5(b)} (*, \overline{b}, 0)$,
  - a transition $(d, b, 2) \xrightarrow{\mathsf{r}_5(\overline{b})} (d, b, 1)$,
  - a transition $(d, b, 2) \xrightarrow{\mathsf{r}_5(*)} (d, b, 1)$.

The transition system for the sender is represented graphically in Figure 2.7 for the case where only one datum is involved.

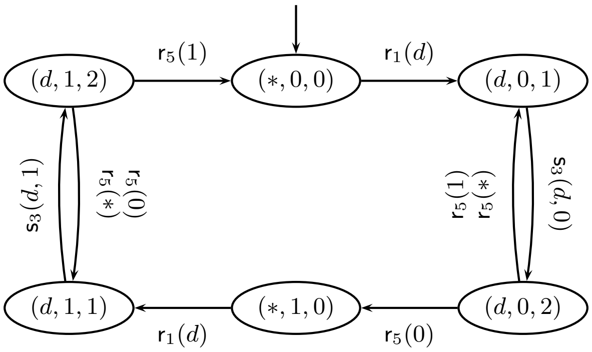


Fig. 2.7. Transition system for the sender

The behaviour of the receiver R is described by a transition system as follows. As states of the receiver, we have triples (d, b, i) where $d \in D \cup \{*\}$, $b \in B$ and $i \in \{0, 1, 2\}$, satisfying $d = *$ if and only if $i \neq 1$. State (d, b, i) is roughly a state in which the datum to be delivered is d and the right bit is b. If $d = *$, no such datum is available. The initial state is $(*, 0, 0)$. As actions, we have $\mathsf{s}_2(d)$ for each $d \in D$, $\mathsf{r}_4(f)$ for each $f \in F \cup \{*\}$, and $\mathsf{s}_6(b)$ for each $b \in B$. As transitions of the receiver, we have the following:

- for each datum $d \in D$ and bit $b \in B$:
  - a transition $(*, b, 0) \xrightarrow{\mathsf{r}_4(d,b)} (d, b, 1)$,
  - a transition $(*, b, 0) \xrightarrow{\mathsf{r}_4(d,\overline{b})} (*, \overline{b}, 2)$,
  - a transition $(d, b, 1) \xrightarrow{\mathsf{s}_2(d)} (*, b, 2)$;
- for each bit $b \in B$:
  - a transition $(*, b, 0) \xrightarrow{\mathsf{r}_4(*)} (*, \overline{b}, 2)$,
  - a transition $(*, b, 2) \xrightarrow{\mathsf{s}_6(b)} (*, \overline{b}, 0)$.

The transition system for the receiver is represented graphically in Figure 2.8 for the case where only one datum is involved.

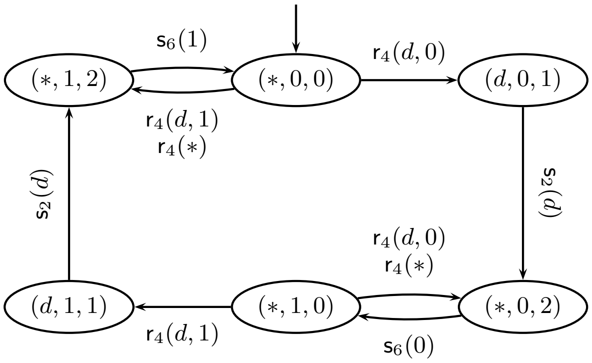


Fig. 2.8. Transition system for the receiver

The behaviour of the data transmission channel K is described by a transition system as follows. As states of the channel, we have pairs (f, i), where $f \in F \cup \{*\}$ and $i \in \{0, 1, 2, 3\}$, satisfying $f = *$ if and only if $i = 0$. State (f, i) is roughly a state in which the frame to be transmitted is f. If $f = *$, no such frame is available. The initial state is $(*, 0)$. As actions, we have i, $\mathsf{r}_3(f)$ for each $f \in F$, and $\mathsf{s}_4(f)$ for each $f \in F \cup \{*\}$. As transitions of the channel, we have the following:

- for each frame $f \in F$:
  - a transition $(*, 0) \xrightarrow{\mathsf{r}_3(f)} (f, 1)$,
  - a transition $(f, 2) \xrightarrow{\mathsf{s}_4(f)} (*, 0)$,
  - a transition $(f, 3) \xrightarrow{\mathsf{s}_4(*)} (*, 0)$;
- for each frame $f \in F$ and $i \in \{2, 3\}$:
  - a transition $(f, 1) \xrightarrow{\mathsf{i}} (f, i)$.

Note that this transition system is not determinate: for each frame f we have both $(*, 0) \xrightarrow{\mathsf{r}_3(f)\ \mathsf{i}} (f, 2)$ and $(*, 0) \xrightarrow{\mathsf{r}_3(f)\ \mathsf{i}} (f, 3)$, but the actions that can be performed from $(f, 2)$ and $(f, 3)$ are different. The action i is an internal action that cannot be performed synchronously with any other action. Thus, the channel cannot be forced to leave all frames uncorrupted. The transition system for channel K is represented graphically in Figure 2.9 for the case where only one datum is involved.

The behaviour of the acknowledgement transmission channel L is described by a transition system as follows. As states of the channel, we have pairs (b, i), where $b \in B \cup \{*\}$ and $i \in \{0, 1, 2, 3\}$, satisfying $b = *$ if and only if $i = 0$. State (b, i) is roughly a state in which the bit to be transmitted is

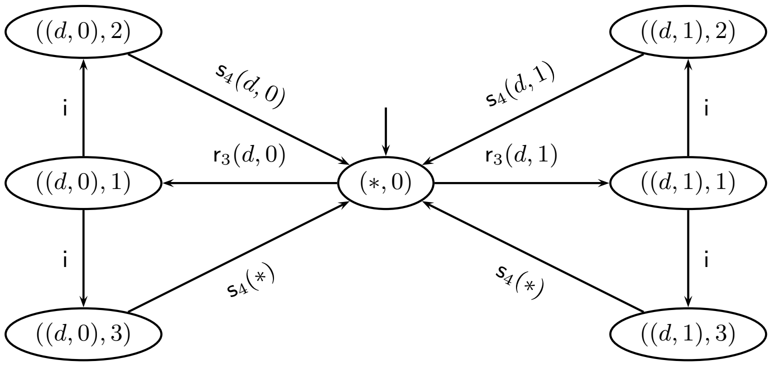


Fig. 2.9. Transition system for the data transmission channel

b. If $b = *$, no such bit is available. The initial state is $(*, 0)$. As actions, we have i, $\mathsf{s}_5(b)$ for each $b \in B \cup \{*\}$, and $\mathsf{r}_6(b)$ for each $b \in B$. As transitions of the channel, we have the following:

- for each bit $b \in B$:
  - a transition $(*, 0) \xrightarrow{\mathsf{r}_6(b)} (b, 1)$,
  - a transition $(b, 2) \xrightarrow{\mathsf{s}_5(b)} (*, 0)$,
  - a transition $(b, 3) \xrightarrow{\mathsf{s}_5(*)} (*, 0)$;
- for each bit $b \in B$ and $i \in \{2, 3\}$:
  - a transition $(b, 1) \xrightarrow{\mathsf{i}} (b, i)$.

Just as the transition system for channel K, the transition system for channel L is not determinate: for each bit b we have both $(*, 0) \xrightarrow{\mathsf{r}_6(b)\ \mathsf{i}} (b, 2)$ and $(*, 0) \xrightarrow{\mathsf{r}_6(b)\ \mathsf{i}} (b, 3)$, but the actions that can be performed from $(b, 2)$ and $(b, 3)$ are different. Like in the case of channel K, channel L cannot be forced to leave all acknowledgements uncorrupted. The transition system for channel L is represented graphically in Figure 2.10.

The behaviour of the whole system is described as follows:

$$\partial_H(S \parallel_\gamma K \parallel_\gamma L \parallel_\gamma R)$$

where

$$\begin{aligned} H = {} & \{\mathsf{s}_3(f), \mathsf{r}_3(f) \mid f \in F\} \cup \{\mathsf{s}_4(f), \mathsf{r}_4(f) \mid f \in F \cup \{*\}\} \\ & \cup \{\mathsf{s}_5(b), \mathsf{r}_5(b) \mid b \in B \cup \{*\}\} \cup \{\mathsf{s}_6(b), \mathsf{r}_6(b) \mid b \in B\} \end{aligned}$$

and the communication function γ is defined in the standard way for handshaking communication (see Section 2.2), i.e. such that

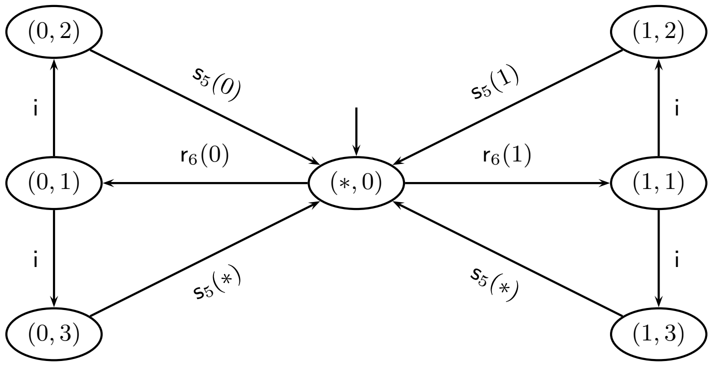


Fig. 2.10. Transition system for the acknowledgement transmission channel

$$\gamma(\mathsf{s}_i(d), \mathsf{r}_i(d)) = \gamma(\mathsf{r}_i(d), \mathsf{s}_i(d)) = \mathsf{c}_i(d)$$

for $i \in \{3, 4, 5, 6\}$ and $d \in D$, and it is undefined otherwise.

Parallel composition and encapsulation of the transition systems of S, K, L and R as described above results in the following transition system. As states, we have quadruples (s, k, l, r), where s, k, l and r are states of S, K, L and R, respectively. As initial state, we have $((*, 0, 0), (*, 0), (*, 0), (*, 0, 0))$. As actions, we have $\mathsf{r}_1(d)$ and $\mathsf{s}_2(d)$ for each $d \in D$, $\mathsf{c}_3(f)$ for each $f \in F$, $\mathsf{c}_4(f)$ for each $f \in F \cup \{*\}$, $\mathsf{c}_5(b)$ for each $b \in B \cup \{*\}$, $\mathsf{c}_6(b)$ for each $b \in B$, and i. As transitions, we have the following:

- for each datum $d \in D$ and bit $b \in B$:
  - $((*, b, 0), (*, 0), (*, 0), (*, b, 0)) \xrightarrow{\mathsf{r}_1(d)} ((d, b, 1), (*, 0), (*, 0), (*, b, 0))$,
  - $((d, b, 1), (*, 0), (*, 0), (*, b, 0)) \xrightarrow{\mathsf{c}_3(d,b)} ((d, b, 2), ((d, b), 1), (*, 0), (*, b, 0))$,
  - $((d, b, 2), ((d, b), 1), (*, 0), (*, b, 0)) \xrightarrow{\mathsf{i}} ((d, b, 2), ((d, b), 2), (*, 0), (*, b, 0))$,
  - $((d, b, 2), ((d, b), 2), (*, 0), (*, b, 0)) \xrightarrow{\mathsf{c}_4(d,b)} ((d, b, 2), (*, 0), (*, 0), (d, b, 1))$,
  - $((d, b, 2), (*, 0), (*, 0), (d, b, 1)) \xrightarrow{\mathsf{s}_2(d)} ((d, b, 2), (*, 0), (*, 0), (*, b, 2))$,
  - $((d, b, 2), (*, 0), (*, 0), (*, b, 2)) \xrightarrow{\mathsf{c}_6(b)} ((d, b, 2), (*, 0), (b, 1), (*, \overline{b}, 0))$,
  - $((d, b, 2), (*, 0), (b, 1), (*, \overline{b}, 0)) \xrightarrow{\mathsf{i}} ((d, b, 2), (*, 0), (b, 2), (*, \overline{b}, 0))$,
  - $((d, b, 2), (*, 0), (b, 2), (*, \overline{b}, 0)) \xrightarrow{\mathsf{c}_5(b)} ((*, \overline{b}, 0), (*, 0), (*, 0), (*, \overline{b}, 0))$,

  - $((d, b, 2), ((d, b), 1), (*, 0), (*, b, 0)) \xrightarrow{\mathsf{i}} ((d, b, 2), ((d, b), 3), (*, 0), (*, b, 0))$,
  - $((d, b, 2), ((d, b), 3), (*, 0), (*, b, 0)) \xrightarrow{\mathsf{c}_4(*)} ((d, b, 2), (*, 0), (*, 0), (*, \overline{b}, 2))$,
  - $((d, b, 2), (*, 0), (*, 0), (*, \overline{b}, 2)) \xrightarrow{\mathsf{c}_6(\overline{b})} ((d, b, 2), (*, 0), (\overline{b}, 1), (*, b, 0))$,
  - $((d, b, 2), (*, 0), (\overline{b}, 1), (*, b, 0)) \xrightarrow{\mathsf{i}} ((d, b, 2), (*, 0), (\overline{b}, 2), (*, b, 0))$,
  - $((d, b, 2), (*, 0), (\overline{b}, 2), (*, b, 0)) \xrightarrow{\mathsf{c}_5(\overline{b})} ((d, b, 1), (*, 0), (*, 0), (*, b, 0))$,

- $((d,b,2),(*,0),(\overline{b},1),(*,b,0)) \xrightarrow{\mathsf{i}} ((d,b,2),(*,0),(\overline{b},3),(*,b,0))$,
- $((d,b,2),(*,0),(\overline{b},3),(*,b,0)) \xrightarrow{\mathsf{c}_5(*)} ((d,b,1),(*,0),(*,0),(*,b,0))$,

- $((d,b,2),(*,0),(b,1),(*,\overline{b},0)) \xrightarrow{\mathsf{i}} ((d,b,2),(*,0),(b,3),(*,\overline{b},0))$,
- $((d,b,2),(*,0),(b,3),(*,\overline{b},0)) \xrightarrow{\mathsf{c}_5(*)} ((d,b,1),(*,0),(*,0),(*,\overline{b},0))$,
- $((d,b,1),(*,0),(*,0),(*,\overline{b},0)) \xrightarrow{\mathsf{c}_3(d,b)} ((d,b,2),((d,b),1),(*,0),(*,\overline{b},0))$,
- $((d,b,2),((d,b),1),(*,0),(*,\overline{b},0)) \xrightarrow{\mathsf{i}} ((d,b,2),((d,b),2),(*,0),(*,\overline{b},0))$,
- $((d,b,2),((d,b),2),(*,0),(*,\overline{b},0)) \xrightarrow{\mathsf{c}_4(d,b)} ((d,b,2),(*,0),(*,0),(*,b,2))$,
- $((d,b,2),((d,b),1),(*,0),(*,\overline{b},0)) \xrightarrow{\mathsf{i}} ((d,b,2),((d,b),3),(*,0),(*,\overline{b},0))$,
- $((d,b,2),((d,b),3),(*,0),(*,\overline{b},0)) \xrightarrow{\mathsf{c}_4(*)} ((d,b,2),(*,0),(*,0),(*,b,2))$.

The transition system for the whole protocol is represented graphically in Figure 2.11 for the case where only one datum is involved. This transition system does not reflect the configuration of the protocol, but is useful for analysis of the protocol. The transition system for the whole protocol shows, for example, that data are delivered in the order in which they were offered, without any loss, if it is assumed that cycles of communication actions at internal ports and the action i are eventually left.

Exercise 2.4.1 (Simple protocol). We describe a protocol for reliable communication of data from a set D between a sender S and a receiver R over a transmission channel C. The configuration of this protocol is given in Figure 2.12. Using standard communication, the transition system for the sender is given by $S = (S_S, A_S, \rightarrow_S, \downarrow_S, s_S)$ where

$$
\begin{aligned}
S_S &= D \cup \{\perp\},\\
A_S &= \{\mathsf{r}_1(d), \mathsf{s}_3(d) \mid d \in D\},\\
\rightarrow_S &= \{(\perp, \mathsf{r}_1(d), d), (d, \mathsf{s}_3(d), \perp) \mid d \in D\},\\
\downarrow_S &= \emptyset,\\
s_S &= \perp .
\end{aligned}
$$

Upon receipt of a message d offered to the sender by the environment (this is the action $\mathsf{r}_1(d)$), the sender transmits this message to the channel ($\mathsf{s}_3(d)$). This behaviour is repeated ad infinitum.

The transition system for the receiver is given by $R = (S_R, A_R, \rightarrow_R, \downarrow_R, s_R)$ where

$$
\begin{aligned}
S_R &= D \cup \{\perp\},\\
A_R &= \{\mathsf{r}_4(d), \mathsf{s}_2(d) \mid d \in D\},\\
\rightarrow_R &= \{(\perp, \mathsf{r}_4(d), d), (d, \mathsf{s}_2(d), \perp) \mid d \in D\},\\
\downarrow_R &= \emptyset,\\
s_R &= \perp .
\end{aligned}
$$

The receiver waits for a message d from the channel ($\mathsf{r}_4(d)$) and then forwards this message to the environment ($\mathsf{s}_2(d)$). This behaviour is repeated.

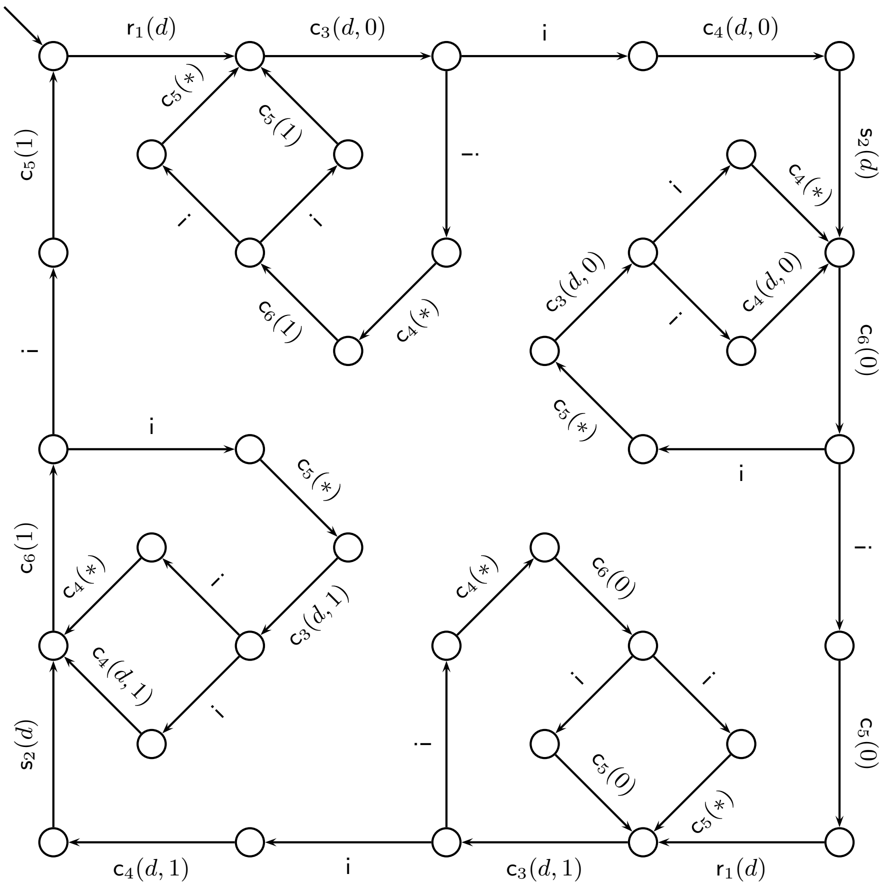


Fig. 2.11. Transition system for the ABP

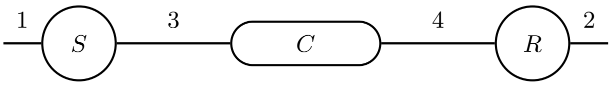


Fig. 2.12. Configuration of the simple protocol

The transmission channel for the transmission channel is given by $C = (S_C, A_C, \rightarrow_C, \downarrow_C, S_C)$ where

$$\begin{aligned}
S_C &= D \cup \{\perp\}, \\
A_C &= \{\mathsf{r}_3(d), \mathsf{s}_4(d) \mid d \in D\}, \\
\rightarrow_C &= \{(\perp, \mathsf{r}_3(d), d), (d, \mathsf{s}_4(d), \perp) \mid d \in D\}, \\
\downarrow_C &= \emptyset, \\
s_C &= \perp .
\end{aligned}$$

The channel accepts a message d from the sender ($\mathsf{r}_3(d)$) and offers it to the receiver ($\mathsf{s}_4(d)$). This behaviour is repeated.

1. Draw these transition systems for $D = \{0, 1\}$.
2. Let γ be the standard communication function for handshaking: for $d \in D$

$$\gamma(\mathsf{s}_3(d), \mathsf{r}_3(d)) = \mathsf{c}_3(d),$$
$$\gamma(\mathsf{s}_4(d), \mathsf{r}_4(d)) = \mathsf{c}_4(d),$$

   and γ is undefined otherwise. Compute the parallel composition of the transition systems S, C, and R under γ, i.e., compute $S \parallel_\gamma C \parallel_\gamma R$.
3. Apply encapsulation of the actions from H where

$$H = \{\mathsf{s}_3(d), \mathsf{r}_3(d), \mathsf{s}_4(d), \mathsf{r}_4(d) \mid d \in D\}$$

   to the transition system that results from $S \parallel_\gamma C \parallel_\gamma R$, i.e., compute $\partial_H(S \parallel_\gamma C \parallel_\gamma R)$.

2.5 Bisimulation and trace equivalence

In Section 1.5, some equivalences have been defined for comparing transition systems. An important property of parallel composition and encapsulation of transition systems is that they preserve these equivalences, by which we mean the following.

Property 2.5.1. Let T_1 and T_2 be transition systems with A as set of actions, let T_1' and T_2' be transition systems with A' as set of actions, let γ be a communication function on a set of actions that includes $A \cup A'$, and let H be a set of actions. Then the following hold:

- if $T_1 \cong T_2$ and $T_1' \cong T_2'$, then $T_1 \parallel_\gamma T_1' \cong T_2 \parallel_\gamma T_2'$;
- if $T_1 \equiv_\mathrm{l} T_2$ and $T_1' \equiv_\mathrm{l} T_2'$, then $T_1 \parallel_\gamma T_1' \equiv_\mathrm{l} T_2 \parallel_\gamma T_2'$;
- if $T_1 \equiv_\mathrm{tr} T_2$ and $T_1' \equiv_\mathrm{tr} T_2'$, then $T_1 \parallel_\gamma T_1' \equiv_\mathrm{tr} T_2 \parallel_\gamma T_2'$;
- if $T_1 \leftrightarrows T_2$ and $T_1' \leftrightarrows T_2'$, then $T_1 \parallel_\gamma T_1' \leftrightarrows T_2 \parallel_\gamma T_2'$;
- if $T_1 \cong T_2$, then $\partial_H(T_1) \cong \partial_H(T_2)$;
- if $T_1 \equiv_\mathrm{l} T_2$, then $\partial_H(T_1) \equiv_\mathrm{l} \partial_H(T_2)$;
- if $T_1 \equiv_\mathrm{tr} T_2$, then $\partial_H(T_1) \equiv_\mathrm{tr} \partial_H(T_2)$;
- if $T_1 \leftrightarrows T_2$, then $\partial_H(T_1) \leftrightarrows \partial_H(T_2)$.

Hence, a parallel composition of transition systems is bisimulation equivalent to a parallel composition of transition systems obtained by replacing the constituent transition systems by ones that are bisimulation equivalent. This property is actually what justifies such replacements. It underlies many techniques for the analysis of process behaviour. If an equivalence is preserved by an operation, the equivalence is called a *congruence* with respect to the operation. Let us now illustrate how the congruence properties can be used.

Example 2.5.1. We consider again the split and merge connections from Examples 1.5.9 and 1.5.10. Both kinds of connections are used as connections between nodes in networks. Suppose that the behaviour of a particular network is described by

$$\partial_H(T_1 \parallel_\gamma \ldots \parallel_\gamma T_k \parallel_\gamma T_{k+1} \parallel_\gamma \ldots \parallel_\gamma T_n) ,$$

where $T_1, \ldots, T_n$ are transition systems describing the behaviour of the nodes and connections that occur in the network. Suppose further that T_k is the first transition system for a merge connection given in Example 1.5.10 and that T'_k is the second transition system for a merge connection given in Example 1.5.10. Recall that the two transition systems for a merge connection are bisimulation equivalent. Hence, replacement of T_k by T'_k yields a network that is bisimulation equivalent to the original network.

Now, suppose instead that T_k is the transition system for the split connection given in Example 1.5.9 and that T'_k is the transition system for the split-like connection given in Example 1.5.9. Recall that those two transition systems are trace equivalent, but not bisimulation equivalent. Hence, replacement of T_k by T'_k yields a network that is trace equivalent to the original network. However, the networks are not bisimulation equivalent. Because of the premature choice of the output port in the case of T'_k, the replacement may even introduce new possibilities to become inactive in the network. Such changes remain unnoticed under trace equivalence, because trace equivalence does not tell us anything about possibilities to become inactive.

The following properties of parallel composition hold because of the conditions imposed on the communication function (see Def. 2.2.1). The first of these is called *commutativity* of parallel composition, the second one *associativity* of parallel composition.

Property 2.5.2. Let T_1, T_2 and T_3 be transition systems with A_1, A_2 and A_3, respectively, as set of actions. Let γ be a communication function on a set of actions that includes $A_1 \cup A_2 \cup A_3$. Then the following holds:

$$T_1 \parallel_\gamma T_2 \leftrightarrows T_2 \parallel_\gamma T_1 ,$$
$$(T_1 \parallel_\gamma T_2) \parallel_\gamma T_3 \leftrightarrows T_1 \parallel_\gamma (T_2 \parallel_\gamma T_3) .$$

The fact that parallel composition is associative indicates that it is allowed to omit parentheses from both $(T_1 \parallel_\gamma T_2) \parallel_\gamma T_3$ and $T_1 \parallel_\gamma (T_2 \parallel_\gamma T_3)$ and to write $T_1 \parallel_\gamma T_2 \parallel_\gamma T_3$ instead.

Exercise 2.5.1. Give a transition system T_u such that the properties $T \parallel_\gamma T_u \leftrightarrows T$ and $T_u \parallel_\gamma T \leftrightarrows T$ hold for arbitrary transition system T and arbitrary communication function γ. Such a transition system T_u is called a *unit element* for parallel composition.

Find out whether T_u is also a unit element for parallel composition with respect to language equivalence, trace equivalence, and isomorphism.

Exercise 2.5.2. Recall the definition of reduction of a transition system from Definition 1.5.2. Prove that $\mathrm{red}(T \parallel_\gamma T') \cong \mathrm{red}(T) \parallel_\gamma \mathrm{red}(T')$ and $\mathrm{red}(\partial_H(T)) \cong \partial_H(\mathrm{red}(T))$ for arbitrary transition systems T and T', arbitrary communication function γ and arbitrary set of actions H.

Exercise 2.5.3. Give a counterexample for $\partial_H(T_1 \parallel_\gamma T_2) \leftrightarrows \partial_H(T_1) \parallel_\gamma \partial_H(T_2)$, i.e., find H, γ, T_1, and T_2 such that $\partial_H(T_1 \parallel_\gamma T_2) \not\leftrightarrows \partial_H(T_1) \parallel_\gamma \partial_H(T_2)$. Can you also give a counterexample in case $\leftrightarrows$ is replaced by either one of $\equiv_{\mathrm{l}}$, $\equiv_{\mathrm{tr}}$, or $\cong$?

Exercise 2.5.4. Is it the case that $T_1 \parallel_\gamma T_2 \cong T_2 \parallel_\gamma T_1$? If so, give a proof; if not, give a counterexample. Repeat this exercise for associativity of parallel composition. What can be said about commutativity and associativity of parallel composition w.r.t. language equivalence and trace equivalence?

Exercise 2.5.5. Is it possible to give a transition system T_z such that $T \parallel_\gamma T_z \leftrightarrows T_z$ and $T_z \parallel_\gamma T \leftrightarrows T_z$ for any transition system T and any communication function γ? Such a transition system T_z would be called a *zero element* for parallel composition.

2.6 Petri nets and transition systems

For a better understanding of the notion of transition system, we looked in the previous chapter and in Section 2.3 into its connections with the familiar notions of program and automaton. For the interested reader, we now look into its connections with the notion of Petri net. Sometimes, the notion of Petri net is considered to be the fundamental notion for the description of process behaviour. We believe that it is too complicated to be acceptable as a fundamental notion.

Petri nets can be regarded as a generalized kind of transition system. In this section, we restrict our attention to the kind of Petri nets known as *place/transition nets* with arc weight 1. They are illustrative for almost any other kind of Petri nets. If no confusion can arise, we will call them simply *nets*. The crucial differences between nets and transition systems are the following. In transition systems a distinction is made between successfully terminating states and terminal states, whereas in Petri nets there is no such distinction. We can deal with this difference by only considering transition systems without successfully terminating states[1]. Furthermore, in transition systems, choices between behaviours and sequentiality of behaviours are regarded as the basic aspects of process behaviour, whereas in nets, concurrency of behaviours is also regarded as a basic aspect of process behaviour. Nets

[1] Although in this book we do not consider nets with terminating states, nets can be easily extended to incorporate terminating states. This requires the addition of a set of markings indicating the terminating states.

support the direct description of concurrency because they can deal with states that are distributed over several places.

Definition 2.6.1. A net N is a tuple (P, T, I, O, ℓ, m_0) where

- P is a set of *places*;
- T is a set of *transitions*;
- $I \subseteq P \times T$ and $O \subseteq T \times P$ are the *input* and *output* relations, respectively;
- $\ell : T \to A$ is a transition-labelling function;
- $m_0 \in P \to \mathbb{N}$ is the *initial marking*.

In the standard definition of the notion of place/transition net, the pre- and post-sets of each transition are given by a *flow relation* $F \subseteq (P \times T) \cup (T \times P)$. Moreover, there is a *arc weight function* $W : F \to \mathbb{N}$ in the standard definition. Because, we restrict ourselves to the case where the arc weight is invariably 1, the arc weight function is superfluous.

A net can be considered to distribute the states of a transition system over several places as follows. Each place contains zero, one or more *tokens*. The numbers of tokens contained in the different places make up the states of a net, also called markings.

Example 2.6.1 (Binary variable). We consider a binary variable. A binary variable holds at any moment either the value 0 or the value 1. Initially, it holds the value 0. The current value of the variable can be read by means of the action $\mathsf{read}(b)$. Here b represents the current value of the variable. Hence $\mathsf{read}(b)$ is enabled only if the current value of the variable is b. The value of the variable can be changed by means of the action $\mathsf{write}(b)$. Here b represents the value the variable will hold afterwards. The behaviour of such a binary variable can be described by the net $N = (P, T, I, O, \ell, m_0)$ where

- $P = \{0, 1\}$;
- $T = \{r0, r1, 0w0, 0w1, 1w0, 1w1\}$;
- $I = \{(0, r0), (1, r1), (0, 0w0), (0, 0w1), (1, 1w1), (1, 1w0)\}$;
- $O = \{(r0, 0), (r1, 1), (0w0, 0), (0w1, 1), (1w1, 1), (1w0, 0)\}$;
- $\ell(r0) = \mathsf{read}(0)$, $\ell(r1) = \mathsf{read}(1)$, $\ell(0w0) = \ell(1w0) = \mathsf{write}(0)$, and $\ell(0w1) = \ell(1w1) = \mathsf{write}(1)$;
- $m_0(0) = 1$ and $m_0(1) = 0$.

The net for the binary variable is represented graphically in Figure 2.13. The places and transitions are represented as follows. Places are represented as circles and transitions t as boxes. In many examples, as a way of referring to these places, the places are labeled by their names. For transitions, usually, the name is not displayed. Instead these are labeled by the value associated with them by the labeling function ℓ. The elements of the input and output relation are represented by directed arrows: for $(p, t) \in I$ this results in an arrow from the place p to the transition t, and vice versa, for $(t, p) \in O$ an arrow is drawn from transition t to place p. The initial marking is represented by putting bullets (called tokens) into the circles: for a place p, $m_0(p)$ tokens are put in the circle representing that place.

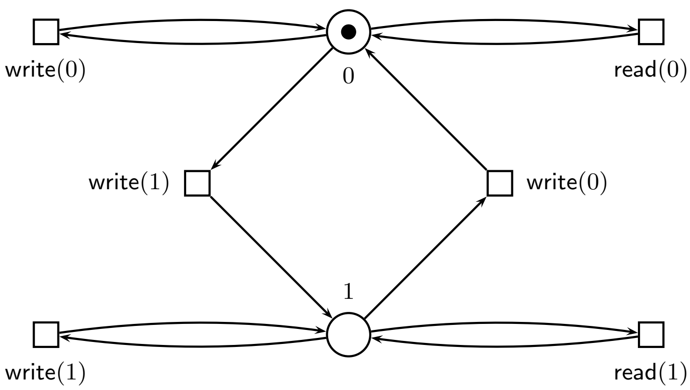


Fig. 2.13. Net for the binary variable

Example 2.6.2 (Distributed state). A system where two independent tasks, task1 and task2 have to be performed can easily be given as the net given in Figure 2.14. Formally, this is the net (P, T, I, O, ℓ, m_0) where

- $P = \{b_1, e_1, b_2, e_2\}$,
- $T = \{t_1, t_2\}$;
- $I = \{(b_1, t_1), (b_2, t_2)\}$;
- $O = \{(t_1, e_1), (t_2, e_2)\}$;
- $\ell(t_1) = \mathsf{task1}$ and $\ell(t_2) = \mathsf{task2}$;
- $m_0(b_1) = m_0(b_2) = 1$ and $m_0(e_1) = m_0(e_2) = 0$.

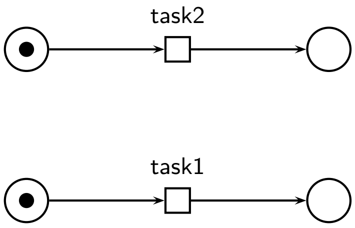


Fig. 2.14. Net with a distributed state

Example 2.6.3 (Print service). Consider a system where a number of printers, say P, can be used to print a large amount of documents. We introduce a place containing a token for each of the documents to be printed. In order to control the number of documents that is printed in parallel, we also introduce a place containing a token for each printer that is not currently printing, i.e., the available printers. Then the net from Figure 2.15 describes this system, where it is assumed that initially there are two printers available and five documents need to be printed.

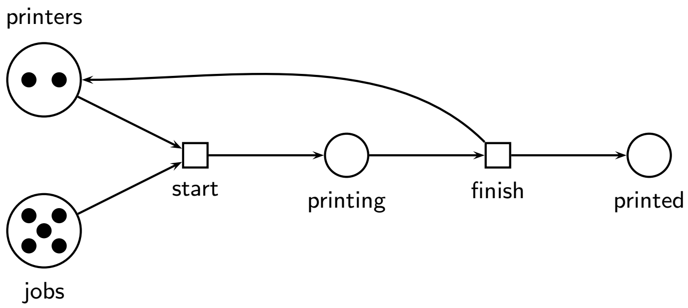


Fig. 2.15. Net for the print service

The *pre-set* of a transition $t \in T$, notation ${}^\bullet t$ is the set of all places that act as an input place for this transition. The *post-set* of a transition $t \in T$, notation $t^\bullet$ is the set of all places that act as an output place for this transition. Formally,

$${}^\bullet t = \{p \in P \mid (p,t) \in I\} \qquad t^\bullet = \{p \in P \mid (t,p) \in O\}.$$

A transition t is enabled in a marking if there is at least one token in each place from the pre-set of t.

Definition 2.6.2 (Enabledness). Let N be the net (P, T, I, O, ℓ, m_0). A transition $t \in T$ is *enabled* in a marking m, notation $m[t\rangle$, if and only if $m(p) > 0$ for all $p \in {}^\bullet t$.

By firing t, one token is removed from each place from the pre-set of t and one token is inserted in each place from the post-set of t. This informal explanation can be made more precise as follows.

Definition 2.6.3 (Firing). Let N be the net (P, T, I, O, ℓ, m_0). We define the *firing relation*, $\_[\_\rangle\_ \subseteq (P \to \mathbb{N}) \times T \times (P \to \mathbb{N})$, as follows: $m[t\rangle m'$ if and only if $m[t\rangle$ and $m' : P \to \mathbb{N}$ is for all $p \in P$ defined by:

$$m'(p) = \begin{cases} m(p) & \text{if } p \notin {}^\bullet t \text{ and } p \notin t^\bullet, \\ m(p) - 1 & \text{if } p \in {}^\bullet t \text{ and } p \notin t^\bullet, \\ m(p) + 1 & \text{if } p \notin {}^\bullet t \text{ and } p \in t^\bullet, \\ m(p) & \text{if } p \in {}^\bullet t \text{ and } p \in t^\bullet. \end{cases}$$

Several transitions in one net can be enabled at the same time. For example the transitions representing the two tasks in Example 2.6.2 can both fire in the initial marking. It is also possible that a transition is enabled more than once in a certain marking: in the net representing the print service (Example 2.6.3) two jobs can be started simultaneously. In this book, we interpret nets in such a way that simultaneity of action execution is interpreted as their unrestricted interleaving. This way of looking at processes is called *interleaving semantics*. Another approach is to allow sets or even multisets of actions to be executed in one transition. This type of interpretation is called *true concurrency*.

Example 2.6.4 (Binary variable). Consider again the net for the binary variable from Example 2.6.1. In the initial marking there are three transitions enabled. These are the transitions $r0$, $0w0$, and $0w1$. So, we have $m_0[r0\rangle$, $m_0[0w0\rangle$, and $m_0[0w1\rangle$. The state that results from firing transition $0w1$ is characterized by the marking zero tokens in state 0 and one token in state 1. So we have $m_0[0w1\rangle m_1$ where m_1 is a marking such that $m_1(0) = 0$ and $m_1(1) = 1$. If we restrict ourselves to the states (i.e., markings) that can be reached from the initial marking through a series of firings, the firing relation can be given as follows:

$$\begin{aligned} \_[\_\rangle\_ = \ & \{(m_0, r0, m_0), (m_0, 0w0, m_0), (m_0, 0w1, m_1), \\ & (m_1, r1, m_1), (m_1, 1w1, m_1), (m_1, 1w0, m_0)\}. \end{aligned}$$

Essentially, the meaning of a net in terms of the actions that can be executed can very well be captured as a transition system. The states of this transition system are the markings of the net. Obviously, the initial state of the transition system corresponds to the initial marking of the net. The transitions of the transition system are obtained by considering the firing relation between markings of the net. The following example illustrates this translation of a net into a transition system.

Example 2.6.5. The transition system associated with the net for the binary variable is given in Figure 2.16. The labels of the states represent markings as follows: m_0 is the marking $m_0(0) = 1$ and $m_0(1) = 0$, and m_1 is the marking $m_1(0) = 0$ and $m_1(1) = 1$. The transitions are precisely the firings given before.

Here is a formal definition of the translation of a net into a transition system.

Definition 2.6.4. Let N be the net (P, T, I, O, ℓ, m_0). The transition system associated with N is the transition system $\mathcal{T}(N) = (S, A, \rightarrow, \downarrow, s_0)$ where

- $S = P \rightarrow \mathbb{N}$;
- $A = \{\ell(t) \mid t \in T\}$;
- $\rightarrow = \{(m, a, m') \mid \exists_{t \in T} a = \ell(t) \wedge m[t\rangle m'\}$;
- $\downarrow = \emptyset$;
- $s_0 = m_0$.

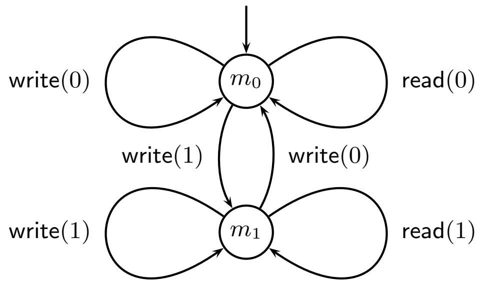


Fig. 2.16. Transition system for the binary variable

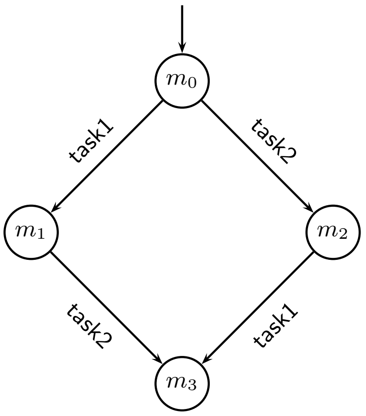


Fig. 2.17. Transition system for the net from Example 2.6.2

Example 2.6.6. The transition system associated with the net of Example 2.6.2 is given in Figure 2.17.

As explained before, the main difference between a net and a transition system without successfully terminating states is that a net allows for a distributed state. Thus, in a sense, such a transition system can be considered a net where a distributed state is not used.

Example 2.6.7. The net associated with the transition system from Figure 2.17 is given in Figure 2.18.

Formally, the translation from a transition system without successfully terminating states to a net can be defined as follows.

Definition 2.6.5. Let T be the transition system $(S, A, \rightarrow, \downarrow, s_0)$ such that $\downarrow = \emptyset$. Then the net associated with T is the net $\mathcal{N}(T) = (P, T, I, O, \ell, m_0)$ where

- $P = S$;
- $T = \rightarrow$;

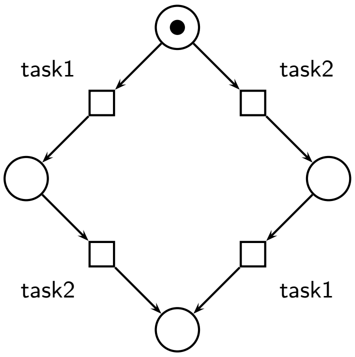


Fig. 2.18. Net for the transition system from Figure 2.17

- $I = \{(s, (s, a, s')) \mid s \in S \wedge (s, a, s') \in \rightarrow\}$;
- $O = \{((s, a, s'), s') \mid (s, a, s') \in \rightarrow \wedge s' \in S\}$;
- $\ell((s, a, s')) = a$ for each $(s, a, s') \in \rightarrow$;
- $m_0(s) = 1$ if $s = s_0$ and $m_0(s) = 0$, otherwise.

Observe that in this translation the resulting net does not contain anymore the actions that are not involved in a transition of the transition system.

The net from Example 2.6.2 has been translated into a transition system in Example 2.6.6 and back to a net in Example 2.6.7. Observe that the original net and the net after the translations are not the same. However, the transition systems of these nets are isomorphic.

Property 2.6.1. For any net N the following holds:

$$(\mathcal{T} \circ \mathcal{N} \circ \mathcal{T})(N) \cong \mathcal{T}(N).$$

If a transition system is translated into a net and then back into a transition system, the two transition systems are isomorphic.

Property 2.6.2. For any transition system T without successfully terminating states the following holds:

$$(\mathcal{T} \circ \mathcal{N})(T) \cong T.$$

Let us look at an example of the use of nets in describing process behaviour.

Example 2.6.8 (Milner's scheduling problem). We consider a system of scheduled processes. It consists of processes $P_1, \ldots, P_n$ $(n > 1)$, each wishing to perform a certain task repeatedly, and a scheduler ensuring that they start their task in cyclic order, beginning with P_1. The behaviour of this system can be described by a net as follows. As places of the system, we have the pairs (i, idle), (i, busy), (i, sch) for $1 \leq i \leq n$.

$$P = \{(i, \mathsf{idle}), (i, \mathsf{busy}), (i, \mathsf{sch}) \mid 1 \leq i \leq n\}$$

As transitions, we have $\mathsf{start}(i)$ (start task i) and $\mathsf{finish}(i)$ (finish task i) for $1 \leq i \leq n$.

$$T = \{\mathsf{start}(i), \mathsf{finish}(i) \mid 1 \leq i \leq n\}.$$

For simplicity we assume that the labelling of these transitions associates with each transition its identity: for all $t \in T$

$$\ell(t) = t.$$

If the marking of the net includes both (i, idle) and (i, sch), process P_i can start performing its task. If its marking includes (i, busy), process P_i can finish performing its task. The input and output relation are the following:

$$I = \{((i, \mathsf{idle}), \mathsf{start}(i)), ((i, \mathsf{sch}), \mathsf{start}(i)), ((i, \mathsf{busy}), \mathsf{finish}(i)) \mid 1 \leq i \leq n\}$$

and

$$O = \{(\mathsf{start}(i), (i, \mathsf{busy})), (\mathsf{start}(i), (nxt(i), \mathsf{sch})), (\mathsf{finish}(i), (i, \mathsf{idle})) \mid 1 \leq i \leq n\}$$

where $nxt(i) = i + 1$ if $i < n$ and $nxt(n) = 1$. As initial marking, we have, for $1 \leq i \leq n$

$$\begin{aligned} m_0((i, \mathsf{idle})) &= 1, \\ m_0((i, \mathsf{sch})) &= \begin{cases} 1 & \text{if } i = 1, \\ 0 & \text{otherwise,} \end{cases} \\ m_0((i, \mathsf{busy})) &= 0. \end{aligned}$$

The net for the system of scheduled processes is represented graphically in Figure 2.19 for the case where $n = 3$. The behaviour of the system is much easier to grasp from this net than from the transition system associated with the net because the structure of the system is clearly reflected in the net.

Exercise 2.6.1. Consider the net from Example 2.6.3.

1. Give the transition system associated with the net from Example 2.6.3 where the initial marking is consider to be m_0 with $m_0(\mathsf{printers}) = m_0(\mathsf{jobs}) = 2$ and $m_0(\mathsf{printing}) = m_0(\mathsf{printed}) = 0$.
2. Also give the transition system for the case that the initial marking is m'_0 with $m'_0(\mathsf{printers}) = 3$, $m'_0(\mathsf{jobs}) = 2$ and $m'_0(\mathsf{printing}) = m'_0(\mathsf{printed}) = 0$.
3. Finally, construct a bisimulation relation between these two transition systems. What does this tell us?

Exercise 2.6.2. Adapt the net describing Milner's scheduling problem (Example 2.6.8) in such a way that processes also have to finish their tasks in cyclic order. Translate the adapted net into a transition system.

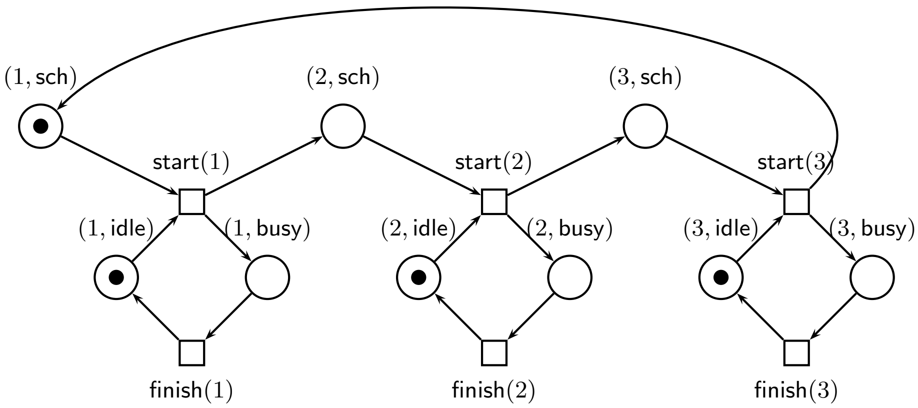


Fig. 2.19. Net for the system of scheduled processes

2.7 Petri nets and parallel composition

For a better understanding of the notion of parallel composition of transition systems, we looked in a previous section into its connections with the familiar notion of parallel execution of programs. Is there a corresponding notion for nets as well? For the interested reader, we now show that parallel composition can also be defined on nets.

Definition 2.7.1. Let $N = (P, T, I, O, \ell, m_0)$ and $N' = (P', T', I', O', \ell', m'_0)$ be nets such that $P \cap P' = \emptyset$ and $T \cap T' = \emptyset$. Let γ be a communication function. The *parallel composition* of N and N' under γ, written $N \parallel_\gamma N'$, is the net $N'' = (P'', T'', I'', O'', \ell'', m''_0)$ where

- $P'' = P \cup P'$;
- $T'' = T \cup T' \cup \{(t, t') \mid \gamma(\ell(t), \ell'(t')) \text{ defined}\}$;
- $I'' = I \cup I' \cup \{(p'', (t, t')) \mid \gamma(\ell(t), \ell'(t')) \text{ defined} \wedge ((p'', t) \in I \vee (p'', t') \in I')\}$;
- $O'' = O \cup O' \cup \{((t, t'), p'') \mid \gamma(\ell(t), \ell'(t')) \text{ defined } \wedge ((t, p'') \in O \vee (t', p'') \in O')\}$;
- ℓ'' is for all $t'' \in T''$ defined by

$$\ell''(t'') = \begin{cases} \ell(t'') & \text{if } t'' \in T, \\ \ell'(t'') & \text{if } t'' \in T', \\ \gamma(\ell(t), \ell'(t')) & \text{if } t'' = (t, t') \text{ for some } t \in T \text{ and } t' \in T'; \end{cases}$$

- m''_0 is for all $p'' \in P''$ defined by

$$m''_0(p'') = \begin{cases} m_0(p'') & \text{if } p'' \in P, \\ m'_0(p'') & \text{if } p'' \in P'. \end{cases}$$

Example 2.7.1. Consider the nets $N = (P, T, I, O, \ell, m_0)$ where

$$\begin{aligned}
&P = \{0, 1, 2, 3\},\\
&T = \{0, 1, 2, 3\},\\
&I = \{(0, 0), (1, 1), (0, 2), (2, 3)\},\\
&O = \{(0, 1), (1, 3), (2, 2), (3, 3)\},\\
&\ell(0) = \ell(2) = a,\ \ell(1) = d,\ \text{and } \ell(3) = e,\\
&m_0(0) = 1 \text{ and } m_0(1) = m_0(2) = m_0(3) = 0
\end{aligned}$$

and $N' = (P', T', I', O', \ell', m'_0)$ where

$$\begin{aligned}
&P' = \{x, y, z\},\\
&T' = \{x, y\},\\
&I' = \{(x, x), (y, y)\},\\
&O' = \{(x, y), (y, z)\},\\
&\ell'(x) = b,\ \text{and } \ell'(y) = f,\\
&m'_0(x) = 1 \text{ and } m'_0(y) = m'_0(z) = 0.
\end{aligned}$$

Let γ be such that $\gamma(a, b) = c$ and γ is undefined otherwise. The parallel composition of N and N' under γ, written $N \parallel_\gamma N'$, is the net $N'' = (P'', T'', I'', O'', \ell'', m''_0)$ where

$$\begin{aligned}
&P'' = \{0, 1, 2, 3, x, y, z\};\\
&T'' = \{0, 1, 2, 3, x, y, (0, x), (2, x)\};\\
&I'' = \{\, (0, 0), (1, 1), (0, 2), (2, 3), (x, x), (y, y),\\
&\qquad (0, (0, x)), (x, (0, x)), (0, (2, x)), (x, (2, x))\};\\
&O'' = \ \{(0, 1), (1, 3), (2, 2), (3, 3), (x, y), (y, z),\\
&\qquad ((0, x), 1), ((0, x), y), ((2, x), 2), ((2, x), y)\};\\
&\ell''(0) = \ell''(2) = a,\ \ell''(1) = d,\ \ell''(3) = e,\ \ell''(x) = b,\ \ell''(y) = f,\ \text{and}\\
&\quad \ell''((0, x)) = \ell''((2, x)) = c,\\
&m''_0(0) = m''_0(x) = 1 \text{ and } m''_0(1) = m''_0(2) = m''_0(3) = m''_0(y) = m''_0(z) = 0.
\end{aligned}$$

The nets N and N' and their parallel composition under γ are represented graphically in Figure 2.20. The names of the places are also given in the figure, for the transitions only the labelling is given.

Let us give a definition of encapsulation on nets as well.

Definition 2.7.2 (Encapsulation). Let $N = (P, T, I, O, \ell, m_0)$ be a net. Let $H \subseteq A$. The *encapsulation* of N with respect to H, written $\partial_H(N)$, is the net $N' = (P', T', I', O', \ell', m'_0)$ where

- $P' = P$;
- $T' = \{t \in T \mid \ell(t) \notin H\}$;
- $I' = \{(p, t) \in I \mid t \in T'\}$;
- $O' = \{(t, p) \in O \mid t \in T'\}$;
- ℓ' is for all $t' \in T'$ defined by $\ell'(t') = \ell(t')$;

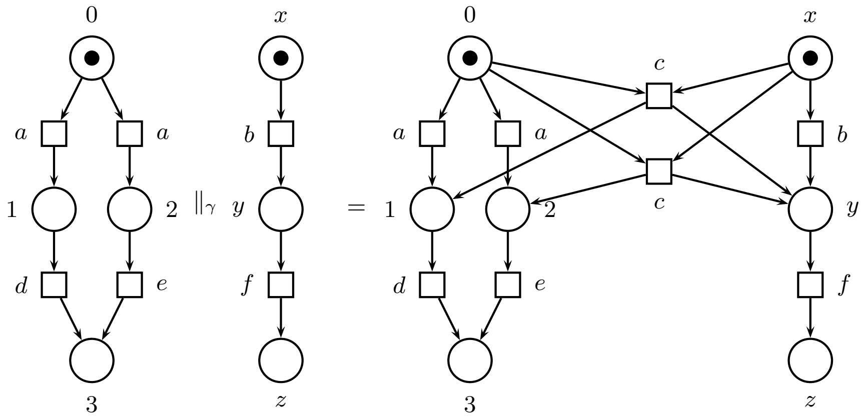


Fig. 2.20. Parallel composition of nets

- m'_0 is for all $p' \in P'$ defined by $m'_0(p') = m_0(p')$.

Observe that in encapsulating a net the transitions labelled with actions from the encapsulation set are removed. As a consequence, such actions do not appear in the net anymore. This is different from the encapsulation on transition systems.

Example 2.7.2. Let us consider the encapsulation of the parallel composition of the nets N and N' from the previous example with respect to the actions a and b, i.e., $\partial_{\{a,b\}}(N'')$. The result is the net $N''' = (P''', T''', I''', O''', \ell''', m'''_0)$ where

$P''' = \{0, 1, 2, 3, x, y, z\}$;
$T''' = \{1, 3, y, (0, x), (2, x)\}$;
$I''' = \{(1, 1), (2, 3), (y, y), (0, (0, x)), (x, (0, x)), (0, (2, x)), (x, (2, x))\}$;
$O''' = \{(1, 3), (3, 3), (y, z), ((0, x), 1), ((0, x), y), ((2, x), 2), ((2, x), y)\}$;
$\ell'''(1) = d,\ \ell'''(3) = e,\ \ell'''(y) = f,$ and $\ell'''((0, x)) = \ell'''((2, x)) = c$;
$m'''_0(0) = m'''_0(x) = 1$ and $m'''_0(1) = m'''_0(2) = m'''_0(3) = m'''_0(y) = m'''_0(z) = 0$.

The application of encapsulation to the net N'' is represented graphically in Figure 2.21.

Here is another example of the use of parallel composition and encapsulation of nets in describing process behaviour.

Example 2.7.3. We consider again the system of scheduled processes from Example 2.6.8. It consists of processes $P_1, \ldots, P_n$ $(n > 1)$, each wishing to perform a certain task repeatedly, and a scheduler ensuring that they start their task in cyclic order, beginning with P_1.

The behaviour of process P_i, for $1 \leq i \leq n$, can be described by a net as follows. As places of P_i, we have the pairs (i, idle) and (i, busy). As initial marking, we have a token in the place (i, idle) only. As actions, we have

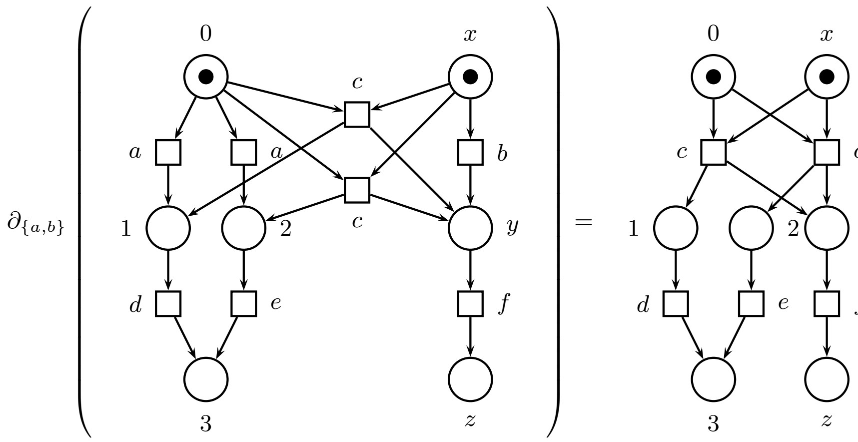


Fig. 2.21. Encapsulation of a net

$\mathsf{request}(i)$ (request to start task i) and $\mathsf{finish}(i)$. With each of these actions we associate a transition. The behaviour of process P_i can thus be represented formally as a net $P_i = (P(i), T(i), I(i), O(i), \ell(i), m_0(i))$ where

$P(i) = \{(i, \mathsf{idle}), (i, \mathsf{busy})\}$;
$T(i) = \{(i, 0), (i, 1)\}$;
$I(i) = \{((i, \mathsf{idle}), (i, 0)), ((i, \mathsf{busy}), (i, 1))\}$;
$O(i) = \{((i, 0), (i, \mathsf{busy})), ((i, 1), (i, \mathsf{idle}))\}$;
$\ell(i)$ is defined by $\ell(i)((i, 0)) = \mathsf{request}(i)$ and $\ell(i)((i, 1)) = \mathsf{finish}(i)$;
$m_0(i)$ is defined by $m_0(i)((i, \mathsf{idle})) = 1$ and $m_0(i)((i, \mathsf{busy})) = 0$.

The behaviour of scheduler S can be described by a net as follows. As places of the scheduler, we have the pairs (i, sch) for $1 \leq i \leq n$. As initial marking, we have a token in the place $(1, \mathsf{sch})$. As actions, we have $\mathsf{grant}(i)$ (grant to start task i) for $1 \leq i \leq n$. With each of these actions we associate a transition. The behaviour of the scheduler can be represented by the net $S = (P', T', I', O', \ell', m'_0)$ where

$P' = \{(i, \mathsf{sch}) \mid 1 \leq i \leq n\}$;
$T' = \{i \mid 1 \leq i \leq n\}$;
$I' = \{((i, \mathsf{sch}), i) \mid 1 \leq i \leq n\}$;
$O' = \{(i, (nxt(i), \mathsf{sch})) \mid 1 \leq i \leq n\}$,
where $nxt(i) = i + 1$ if $i < n$ and $nxt(n) = 1$;
ℓ' is defined by $\ell'(i) = \mathsf{grant}(i)$ for all $1 \leq i \leq n$;
m'_0 is defined by $m'_0((1, \mathsf{sch})) = 1$ and $m'_0((i, \mathsf{sch})) = 0$ for all $1 < i \leq n$.

Graphical representations of the nets P_i and S can be found in Figure 2.22. Using these nets and the composition mechanisms introduced before, the

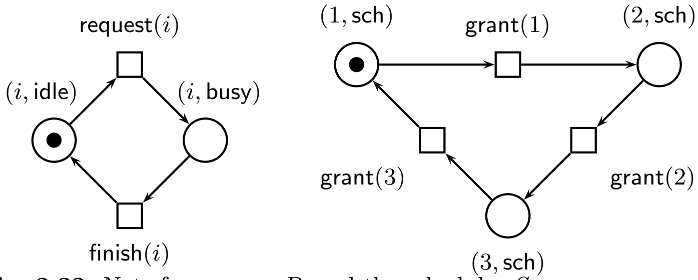


Fig. 2.22. Nets for process P_i and the scheduler S

behaviour of the whole system is described as follows:

$$\partial_H(P_1 \parallel_\gamma \ldots \parallel_\gamma P_n \parallel_\gamma S)$$

where

$$H = \{\mathsf{request}(i), \mathsf{grant}(i) \mid 1 \leq i \leq n\}$$

and the communication function γ is defined such that

$$\gamma(\mathsf{request}(i), \mathsf{grant}(i)) = \gamma(\mathsf{grant}(i), \mathsf{request}(i)) = \mathsf{start}(i)$$

for $1 \leq i \leq n$, and it is undefined otherwise. The net obtained from the nets P_1, …, P_n and S by parallel composition and encapsulation as described above is the same as the net described in Example 2.6.8.

In Section 2.6, we associated a transition system $\mathcal{T}(N)$ with each net N. It happens that this association is useful in showing the close connection between parallel composition of nets and parallel composition of transition systems, and between encapsulation of nets and encapsulation of transition systems.

Property 2.7.1. Let $N = (P, T, I, O, \ell, m_0)$ and $N' = (P', T', I', O', \ell', m'_0)$ be nets such that $P \cap P' = \emptyset$ and $T \cap T' = \emptyset$, let γ be a communication function on actions A, and let $H \subseteq A$. Then we have that

$$\mathcal{T}(N \parallel_\gamma N') \cong \mathcal{T}(N) \parallel_\gamma \mathcal{T}(N'),$$
$$\mathcal{T}(\partial_H(N)) \cong \partial_H(\mathcal{T}(N)).$$

In words, the transition system associated with a parallel composition of nets is isomorphic to the parallel composition of the transition systems associated with those nets; and analogously for encapsulation.

Example 2.7.4. We consider once again the system of scheduled processes from Examples 2.6.8 and 2.7.3. Associating a transition system with the net describing the behaviour of process P_i ($1 \leq i \leq n$) is trivial because each transition has only a single input and a single output place associated with it and because there is only one token in the initial marking. The resulting transition system, $\mathcal{T}(P_i)$, can be described as follows. As states, we simply have the places of the net. As initial state, we have the state (i, idle). As actions, we have $\mathsf{request}(i)$ and $\mathsf{finish}(i)$. As transitions, we have the following:

$$(i, \mathsf{idle}) \xrightarrow{\mathsf{request}(i)} (i, \mathsf{busy}),$$
$$(i, \mathsf{busy}) \xrightarrow{\mathsf{finish}(i)} (i, \mathsf{idle}).$$

Associating a transition system with the net describing the behaviour of the scheduler S is equally trivial. The resulting transition system, $\mathcal{T}(S)$, can be described as follows. As states, we have the places of the net. As initial state, we have the state $(1, \mathsf{sch})$. As actions, we have $\mathsf{grant}(i)$ for $1 \leq i \leq n$. As transitions, we have the following (for $1 \leq i \leq n$):

$$(i, \mathsf{sch}) \xrightarrow{\mathsf{grant}(i)} (nxt(i), \mathsf{sch}).$$

So, the transition systems associated with the nets $P_1, \ldots, P_n$ and S are simply obtained by taking the singleton sets of places as states. In order words, those nets are essentially transition systems. However, their parallel composition as nets yields the net from Example 2.6.8, which is not quite a transition system – because there are transitions with more than one input ir output place and because the initial marking contains more than one token. The transition system described by

$$\partial_H(\mathcal{T}(P_1) \parallel_\gamma \ldots \parallel_\gamma \mathcal{T}(P_n) \parallel_\gamma \mathcal{T}(S)),$$

where H and γ are as in Example 2.7.3, is isomorphic to the transition system associated with the net from Example 2.6.8.

3. Abstraction

Preferably, the design of a complex system starts from a description of its behaviour at a high level of abstraction, i.e. a description serving as a specification of the system to be developed, and ends in a description of the behaviour at a low level of abstraction together with a proof that the behaviour described at the start is essentially the same as the behaviour described at the end after abstraction from actions that have been added during the design process. This chapter deals with this issue of abstraction by introducing the notions of abstraction from internal actions and branching bisimulation equivalence. First of all, we explain informally what abstraction from internal actions is and what branching bisimulation equivalence is, and give a simple example of their use in comparing descriptions of process behaviour (Section 3.1). After that, we define the notions of abstraction from internal actions and branching bisimulation equivalence in a mathematically precise way (Section 3.2). We also use abstraction from internal actions and branching bisimulation equivalence to show that a merge connection with a feedback wire behaves as a sink (Section 3.3), and to show that the simple data communication protocol from Section 2.4 behaves as a buffer of capacity one (Section 3.4).

3.1 Informal explanation

Abstraction from internal actions is an important notion. Frequently, the behaviour of a system is first described at a high level of abstraction, and then as a system composed of interacting components. It should be shown that the two descriptions are equivalent after abstraction from actions added for the interactions between the components. The need for abstraction from certain actions became already apparent in the preceding chapter, while analyzing systems described using transition systems.

Abstraction from internal actions is a means to express that certain actions must be considered to be unobservable. It turns actions from a certain set into a special action, denoted by τ, which is called the silent step. Unlike other actions, the act of performing a silent step is considered to be unobservable. Let us give an example of the use of abstraction.

Example 3.1.1 (Bounded buffers). We consider again the system composed of two bounded buffers from Example 2.1.1. In that example, parallel composition and encapsulation of the two buffers, buffer 1 and buffer 2, resulted in the following transition system. As states, we have pairs (σ_1, σ_2) where σ_i $(i = 1, 2)$ is a sequence of data of which the length is not greater than l_i. As initial state, we have (ϵ, ϵ). As actions, we have $\mathsf{add}_1(d)$, $\mathsf{rem}_2(d)$ and $\mathsf{trf}(d)$ for each datum d. As transitions, we have the following:

- for each datum d and each state (σ_1, σ_2) with the length of σ_1 less than l_1, a transition $(\sigma_1, \sigma_2) \xrightarrow{\mathsf{add}_1(d)} (d\,\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1, \sigma_2\,d)$, a transition $(\sigma_1, \sigma_2\,d) \xrightarrow{\mathsf{rem}_2(d)} (\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1\,d, \sigma_2)$ with the length of σ_2 less than l_2, a transition $(\sigma_1\,d, \sigma_2) \xrightarrow{\mathsf{trf}(d)} (\sigma_1, d\,\sigma_2)$.

In Figure 3.1 this transition system is represented graphically for the case where $l_1 = l_2 = 1$ and the only data involved are the natural numbers 0 and 1. At the end of Example 2.1.1, there was a need to abstract from the

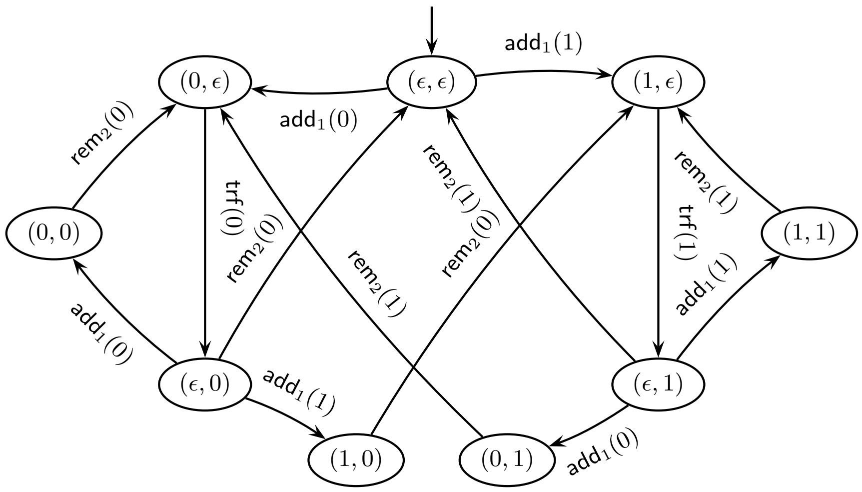


Fig. 3.1. Transition system for encapsulation of two parallel bounded buffers

internal transfer actions $\mathsf{trf}(d)$. The following transition system is the result of abstraction from the actions $\mathsf{trf}(d)$ for $d \in D$. We have the same states as before. As actions that occur, we have $\mathsf{add}_1(d)$ and $\mathsf{rem}_2(d)$ for each datum d. As transitions, we have the following:

- for each datum d and each state (σ_1, σ_2) with the length of σ_1 less than l_1, a transition $(\sigma_1, \sigma_2) \xrightarrow{\mathsf{add}_1(d)} (d\,\sigma_1, \sigma_2)$;

- for each datum d and each state $(\sigma_1, \sigma_2\, d)$, a transition $(\sigma_1, \sigma_2\, d) \xrightarrow{\mathsf{rem}_2(d)} (\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1\, d, \sigma_2)$ with the length of σ_2 less than l_2, a transition $(\sigma_1\, d, \sigma_2) \xrightarrow{\tau} (\sigma_1, d\, \sigma_2)$.

This transition system is represented graphically in Figure 3.2 for the case where $l_1 = l_2 = 1$ and the only data involved are the natural numbers 0 and 1.

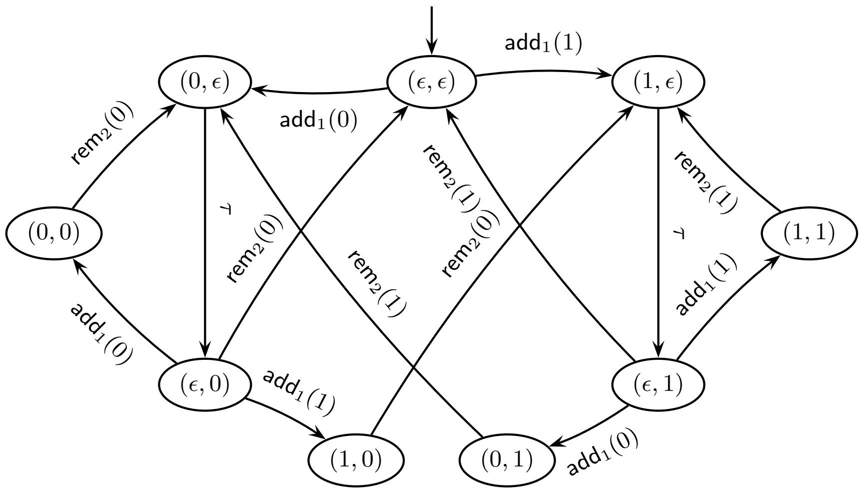


Fig. 3.2. Transition system for abstraction of two parallel bounded buffers

As mentioned above, the act of performing a silent step is considered to be unobservable. However, the act of performing a silent step can sometimes be inferred because a process may proceed as a different process after performing a silent step. In other words, the capabilities of a transition system may change by performing a silent step. Let us look at an example of this phenomenon.

Example 3.1.2. We consider the following two transition systems, of which the second is actually a split connection (see Example 1.5.9). We assume a set of data D. As actions of both transition system, we have $\mathsf{r}_1(d)$, $\mathsf{s}_2(d)$ and $\mathsf{s}_3(d)$ for each $d \in D$. As states of the first transition system, we have pairs (d, i), where $d \in D \cup \{*\}$ and $i \in \{0, 1, 2\}$, with $(*, 0)$ as initial state. As transitions of the first transition system, we have the following:

- for each $d \in D$:
  - a transition $(*, 0) \xrightarrow{\mathsf{r}_1(d)} (d, 1)$,
  - a transition $(d, 1) \xrightarrow{\mathsf{s}_2(d)} (*, 0)$,

– a transition $(d, 1) \xrightarrow{\tau} (d, 2)$,
– a transition $(d, 2) \xrightarrow{\mathsf{s}_3(d)} (*, 0)$.

As states of the second transition system, we have pairs (d, i), where $d \in D \cup \{*\}$ and $i \in \{0, 1\}$, with $(*, 0)$ as initial state. As transitions of the second transition system, we have the following:

- for each $d \in D$:
  – a transition $(*, 0) \xrightarrow{\mathsf{r}_1(d)} (d, 1)$,
  – a transition $(d, 1) \xrightarrow{\mathsf{s}_2(d)} (*, 0)$,
  – a transition $(d, 1) \xrightarrow{\mathsf{s}_3(d)} (*, 0)$.

The transition systems given in this example are represented graphically in Figure 3.3 for the case where $D = \{0, 1\}$. The first transition system has a

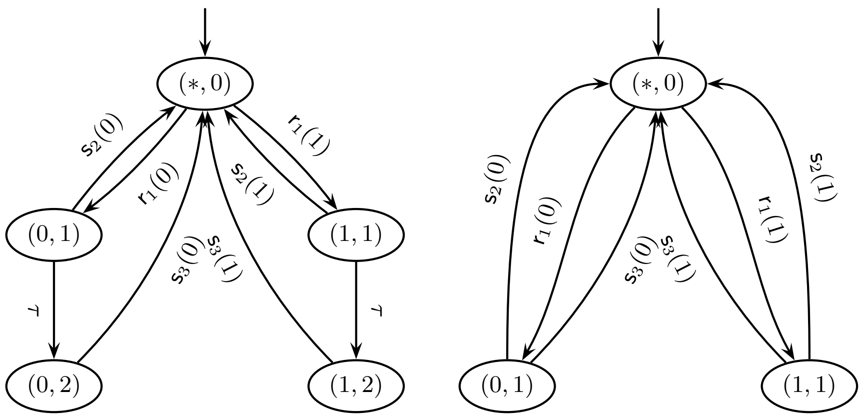


Fig. 3.3. Transition systems of Example 3.1.2

state, viz. state $(d, 2)$, in which it is able to perform action $\mathsf{s}_3(d)$ without being able to perform action $\mathsf{s}_2(d)$ instead; whereas the second transition system does not have such a state. This means the following for the observable behaviour of these transition system. In the case of the first transition system, after $\mathsf{r}_1(d)$ has been performed, two observations are possible. The act of performing $\mathsf{s}_2(d)$ can be observed and, after τ has been performed, the act of performing $\mathsf{s}_3(d)$ can be observed. However, before anything has been observed, it may have become impossible to observe the act of performing $\mathsf{s}_2(d)$. In the case of the second transition system, it remains possible to observe the act of performing $\mathsf{s}_2(d)$ as long as nothing has been observed. So the observable behaviour of the two transition systems differs.

The purpose of abstraction from internal actions is to be able to identify transition systems that have the same observable behaviour. The preceding example shows that an equivalence based on the idea to simply leave out all unobservable actions does not work. Still, in many cases, the act of performing a silent step cannot be inferred, because the process concerned proceeds as the same process after performing a silent step. In such cases, we sometimes say that the silent step is inert. Here is an example of an inert silent step.

Example 3.1.3. We consider the following two transition systems. We assume a set of data D. As actions of both transition systems, we have $\mathsf{r}_1(d)$ and $\mathsf{s}_2(d)$ for each $d \in D$. As states of the first transition system, we have pairs (d, i), where $d \in D \cup \{*\}$ and $i \in \{0, 1, 2\}$, with $(*, 0)$ as initial state. As transitions of the first transition system, we have the following:

- for each $d \in D$:
  - a transition $(*, 0) \xrightarrow{\mathsf{r}_1(d)} (d, 1)$,
  - a transition $(d, 1) \xrightarrow{\tau} (d, 2)$,
  - a transition $(d, 2) \xrightarrow{\mathsf{s}_2(d)} (*, 0)$.

As states of the second transition system, we have pairs (d, i), where $d \in D \cup \{*\}$ and $i \in \{0, 1\}$, with $(*, 0)$ as initial state. As transitions of the second transition system, we have the following:

- for each $d \in D$:
  - a transition $(*, 0) \xrightarrow{\mathsf{r}_1(d)} (d, 1)$,
  - a transition $(d, 1) \xrightarrow{\mathsf{s}_2(d)} (*, 0)$.

The transition systems given in this example are represented graphically in Figure 3.4 for the case where $D = \{0, 1\}$. Initially, only the act of performing

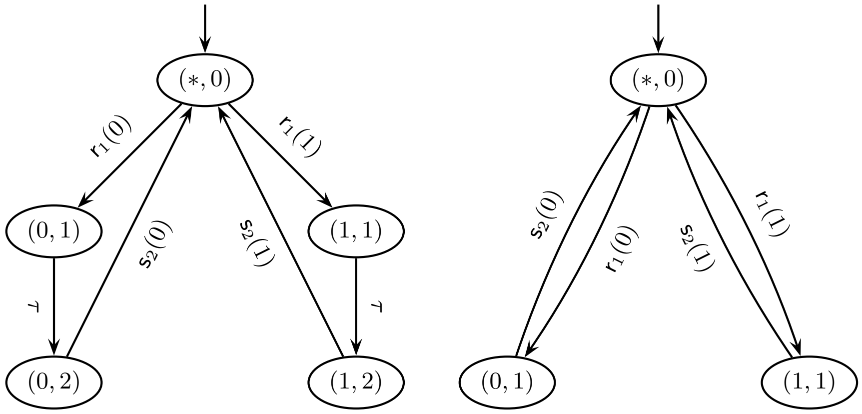


Fig. 3.4. Transition systems of Example 3.1.3

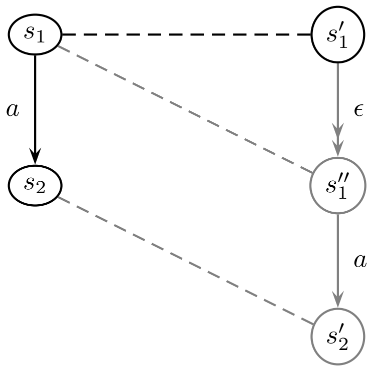
$\mathsf{r}_1(d)$ can be observed. After this has been observed, only the act of performing

$\mathsf{s}_2(d)$ can be observed. There is no way to infer the act of performing the silent step in between. So the observable behavior of these transition systems is the same.

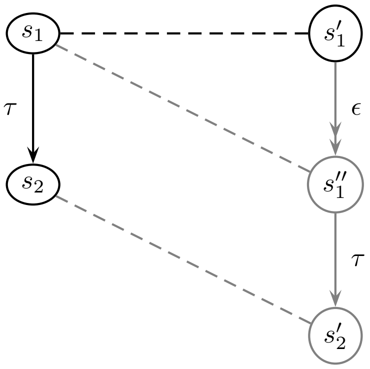
What we understand from the preceding two examples is that a silent step can only be left out if no capabilities get lost by performing it. According to this understanding, we adapt the notion of bisimulation equivalence as follows. Two transition systems T and T' are branching bisimulation equivalent if their states can be related such that:

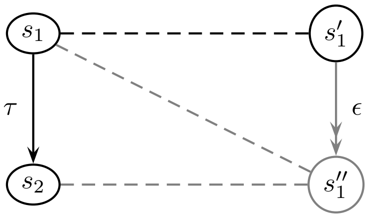
- the initial states are related;

- if states s_1 and s_1' are related and in T a transition with label $a \in A$ is possible from s_1 to some s_2, then in T' a transition with label a is possible from some s_1'' to some s_2' such that a sequence of zero or more transitions with the silent step as label is possible from s_1' to s_1'', s_1 and s_1'' are related, and s_2 and s_2' are related;

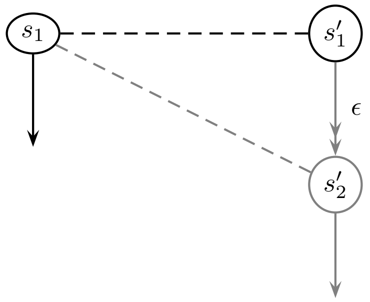


- if states s_1 and s_1' are related and in T a transition with label τ is possible from s_1 to some s_2, then in T' a sequence of zero or more transitions with the silent step as label is possible from s_1' to s_1'' and s_1 and s_1'' are related and either of the following holds:
  - from s_1'' there is a transition with label τ to some state s_2' and s_2 and s_2' are related, or
  - s_2 and s_1'' are related;





- if states s_1 and s'_1 are related and s_1 is a successfully terminating state in T, then in T' a sequence of zero or more transitions with the silent step as label is possible from s'_1 to s'_2 such that s'_2 is a successfully terminating state and s_1 and s'_2 are related;



- likewise, with the role of T and T' reversed.

We could have required s_1 to be related to all states between s'_1 and s''_1 as well, but that turns out to be equivalent. Let us return for a while to the preceding two examples.

Example 3.1.4. We consider again the transition systems of Example 3.1.3. Are those transition systems identified by branching bisimulation equivalence? Yes, they are: relate state $(*, 0)$ of the first transition system to state $(*, 0)$ of the second transition system, and for each $d \in D$, relate the states $(d, 1)$ and $(d, 2)$ of the first transition system to the state $(d, 1)$ of the second transition system. In this way, the states of the two transition systems are related as required for branching bisimulation equivalence. These transition systems and the pairs of related states are represented graphically in Figure 3.5.

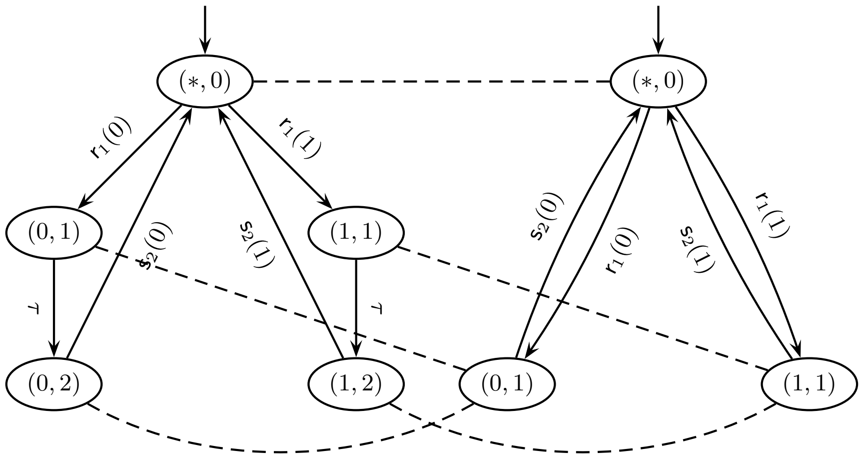


Fig. 3.5. Branching bisimulation relation for the transition systems of Example 3.1.3

Example 3.1.5. We also consider again the transition systems of Example 3.1.2. These are represented graphically in Figure 3.6. Are those transition systems identified by branching bisimulation equivalence? No, they are not. In order to be able to relate, as required, the state $(*, 0)$ of the first transition system to the state $(*, 0)$ of the second transition system (the topmost dashed line in the figure), the state $(0, 1)$ of the first transition system has to be related to the state $(0, 1)$ of the second transition system (the other dashed line in the figure). This is only possible it is possible to relate state $(0, 2)$ of the first transition system to state $(0, 1)$ of the second transition system (the dotted line in the figure). This is not possible since the transition labelled with $\mathsf{s}_2(0)$ from state $(0, 1)$ of the second transition system cannot be mimicked from state $(0, 2)$ in the first transition system.

3.2 Formal definitions

With the previous section, we have prepared the way for the formal definitions of the notions of abstraction from internal actions and branching bisimulation equivalence. However, we have to adapt the definitions of the notion from Chapter 1 and Chapter 2 to the presence of the silent step first. In the adapted definitions, we write A_τ for $A \cup \{\tau\}$.

Definition 3.2.1. A *transition system* T is a quintuple $(S, A, \rightarrow, \downarrow, s_0)$ where

- S is a set of *states*;
- A is a set of *actions*;
- $\rightarrow \subseteq S \times A_\tau \times S$ is a set of *transitions*;
- $\downarrow \subseteq S$ is a set of *successfully terminating states*;

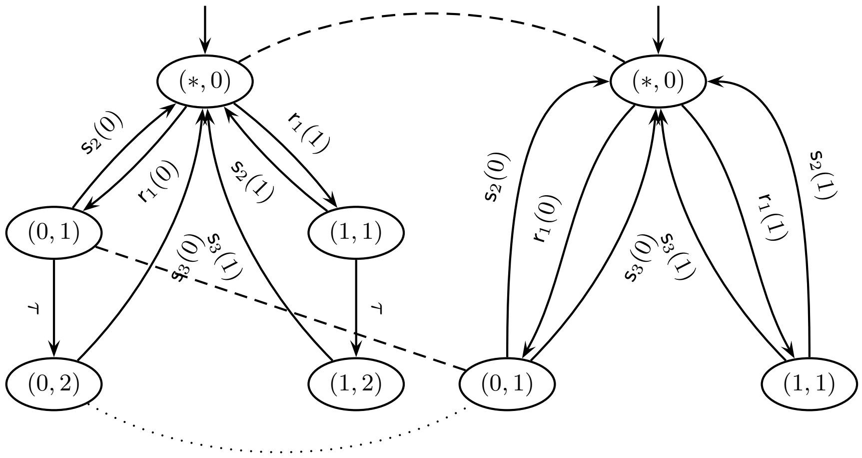


Fig. 3.6. Transition systems of Example 3.1.2

- $s_0 \in S$ is the *initial state*.

The set $\twoheadrightarrow \subseteq S \times A^* \times S$ of *generalized transitions* of T is the smallest subset of $S \times A^* \times S$ satisfying: for all $s, s' \in S$, $a \in A$ and $\sigma, \sigma' \in A^*$

- $s \xrightarrow{\epsilon}\!\!\!\!\twoheadrightarrow s$ for each $s \in S$;
- if $s \xrightarrow{\tau} s'$, then $s \xrightarrow{\epsilon}\!\!\!\!\twoheadrightarrow s'$;
- if $s \xrightarrow{a} s'$, then $s \xrightarrow{a}\!\!\!\!\twoheadrightarrow s'$;
- if $s \xrightarrow{\sigma}\!\!\!\!\twoheadrightarrow s'$ and $s' \xrightarrow{\sigma'}\!\!\!\!\twoheadrightarrow s''$, then $s \xrightarrow{\sigma\,\sigma'}\!\!\!\!\twoheadrightarrow s''$.

A state $s \in S$ is called a *reachable* state of T if there is a $\sigma \in A^*$ such that $s_0 \xrightarrow{\sigma}\!\!\!\!\twoheadrightarrow s$. The set of all reachable states of a transition system T is denoted reach(T). A state $s \in S$ is called a *terminal* state of T if not $s \downarrow$ and there are no $a \in A_\tau$ and $s' \in S$ such that $s \xrightarrow{a} s'$. A state $s \in S$ is called a *deadlock* state of T if s is a reachable state of T and a terminal state of T.

Notice that transitions labeled with the silent step may be included in the set of transitions of a transition system, although the silent step is never included in the set of actions.

As we have changed the notion of transition system in this chapter, we need to reconsider also the equivalences that we defined on the type of transition systems in Section 1.5. Therefore, the relevant definitions are repeated below. The differences with the original definitions in Section 1.5 are minor. Most notably a is now silently assumed to be an action or the silent step (i.e., $a \in A_\tau$). Another difference is in the definition of the generalized transition relation $\twoheadrightarrow$. With the adapted definition of the generalized transitions of a transition system (Definition 3.2.1), this means that in the case of language equivalence and trace equivalence we simply leave out all unobservable actions. Because it does not matter in the case of these equivalences at which stages choices occur, this is all right.

Definition 3.2.2 (Isomorphy). Two transition systems $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ are *isomorphic*, notation $T \cong T'$, if and only if there exists a bijective function $\hbar : \text{reach}(T) \rightarrow \text{reach}(T')$ such that for all $s, s_1, s_2 \in S$, $s', s'_1, s'_2 \in S'$ and $a \in A_\tau$

1. $\hbar(s_0) = s'_0$;
2. whenever $\hbar(s_1) = s'_1$ and $\hbar(s_2) = s'_2$, then $s_1 \xrightarrow{a} s_2$ if and only if $s'_1 \xrightarrow{a}{}' s'_2$;
3. whenever $\hbar(s) = s'$, then $s \downarrow$ if and only if $s' \downarrow'$.

Definition 3.2.3 (Language equivalence). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. A *terminating* trace of T is a sequence $\sigma \in A^*$ such that $s_0 \xrightarrow{\sigma}\!\!\!\!\rightarrow s$ and $s \downarrow$ for some $s \in S$. We write $\text{lang}(T)$ for the set of all terminating traces of T. Two transition systems T and T' are *language equivalent*, written $T \equiv_{\text{l}} T'$, if $\text{lang}(T) = \text{lang}(T')$.

Definition 3.2.4 (Trace equivalence). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. A *trace* of T is a sequence $\sigma \in A^*$ such that $s_0 \xrightarrow{\sigma}\!\!\!\!\rightarrow s$ for some $s \in S$. We write $\text{traces}(T)$ for the set of all traces of T. Then two transition systems T and T' are *trace equivalent*, written $T \equiv_{\text{tr}} T'$, if $\text{traces}(T) = \text{traces}(T')$ and $\text{lang}(T) = \text{lang}(T')$.

Definition 3.2.5 (Bisimulation equivalence). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ be transition systems. Then a *bisimulation* B between T and T' is a binary relation $B \subseteq S \times S'$ such that the following conditions hold: for $a \in A_\tau$

1. $B(s_0, s'_0)$;
2. whenever $B(s_1, s'_1)$ and $s_1 \xrightarrow{a} s_2$, then there is a state s'_2 such that $s'_1 \xrightarrow{a}{}' s'_2$ and $B(s_2, s'_2)$;
3. whenever $B(s_1, s'_1)$ and $s'_1 \xrightarrow{a}{}' s'_2$, then there is a state s_2 such that $s_1 \xrightarrow{a} s_2$ and $B(s_2, s'_2)$;
4. whenever $B(s, s')$ and $s \downarrow$, then $s' \downarrow'$;
5. whenever $B(s, s')$ and $s' \downarrow'$, then $s \downarrow$.

The two transition systems T and T' are *bisimulation equivalent* (also called *bisimilar*), written $T \leftrightarroweq T'$, if there exists a bisimulation B between T and T'. A bisimulation between T and T is called an *autobisimulation* on T.

All of these relations on transition systems are again equivalences. The relations between these equivalences remain the same.

Property 3.2.1. Isomorphy, language equivalence, trace equivalence, and bisimulation equivalence are equivalence relations.

Property 3.2.2. Any two isomorphic transition systems are also bisimulation equivalent. Any two bisimulation equivalent transition systems are also trace equivalent. Any two trace equivalent transition systems are also language equivalent.

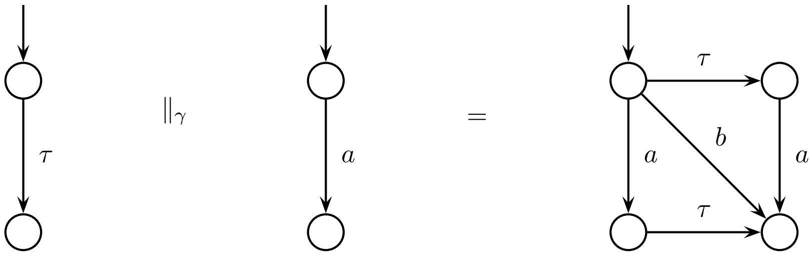
If we restrict to transition systems in which no silent steps occur, the equivalences as defined in Chapter 1 and their adapted definitions in this chapter coincide.

Definition 3.2.6. Let A be a set of actions. A *communication function* on A is a partial function $\gamma : A_\tau \times A_\tau \to A$ satisfying for $a, b, c \in A_\tau$:

- $\gamma(a, \tau)$ and $\gamma(\tau, a)$ are undefined;
- if $\gamma(a, b)$ is defined, then $\gamma(b, a)$ is defined and $\gamma(a, b) = \gamma(b, a)$;
- if $\gamma(a, b)$ and $\gamma(\gamma(a, b), c)$ are defined, then $\gamma(b, c)$ and $\gamma(a, \gamma(b, c))$ are defined and $\gamma(\gamma(a, b), c) = \gamma(a, \gamma(b, c))$.

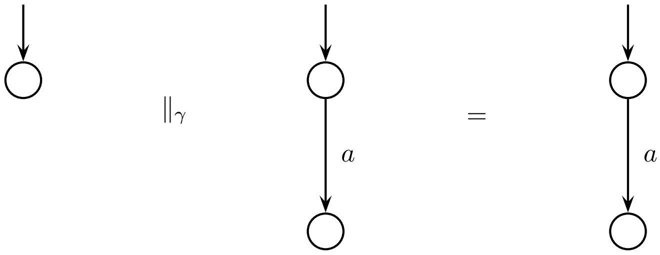
Notice that we consider the silent step to be an action that cannot be performed synchronously with other actions. The reason for this is that it would otherwise be observable.

Example 3.2.1. Suppose, for the sake of this example only, that we do allow the silent step to occur synchronously with other action. More specifically, assume that $\gamma(a, \tau) = \gamma(\tau, a) = b$ for some different actions a and b. Consider the following parallel composition



Although the silent step in the first transition system is not observable, as soon as this transition system is composed in parallel with a transition system in which the communication partner of the silent step (in this case a) occurs, the silent step becomes observable, through the result of the communication (the action b).

A related problem with communications in which the silent step is involved is the congruence properties we have encountered before. If it is our desire to have the property that branching bisimulation equivalence (to be defined shortly) is a congruence with respect to parallel composition, then the following problem occurs. The first transition system above is branching bisimulation equivalent to the transition system with only one reachable state, no transitions, and no successfully terminating states (this transition system could be called deadlock). To have the congruence property, the parallel composition of this deadlock transition system with the second transition system from above should be branching bisimulation equivalent to the result of the above parallel composition. One can easily establish that the results of the two parallel compositions are not branching bisimulation equivalent.



Definition 3.2.7 (Parallel composition). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ be transition systems. Let γ be a communication function on a set of actions that includes $A \cup A'$. The *parallel composition* of T and T' under γ, written $T \|_\gamma T'$, is the transition system $(S'', A'', \rightarrow'', \downarrow'', s''_0)$ where

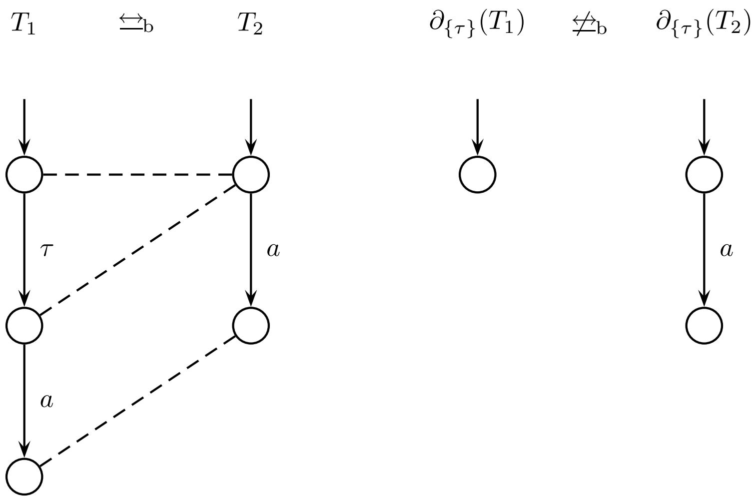
- $S'' = S \times S'$;
- $A'' = A \cup A' \cup \{\gamma(a, a') \mid a \in A, a' \in A', \gamma(a, a') \text{ is defined}\}$;
- $\rightarrow''$ is the smallest subset of $S'' \times A''_\tau \times S''$ such that:
  - if $s_1 \xrightarrow{a} s_2$ and $s' \in S'$, then $(s_1, s') \xrightarrow{a}'' (s_2, s')$;
  - if $s'_1 \xrightarrow{b}' s'_2$ and $s \in S$, then $(s, s'_1) \xrightarrow{b}'' (s, s'_2)$;
  - if $s_1 \xrightarrow{a} s_2$, $s'_1 \xrightarrow{b}' s'_2$ and $\gamma(a, b)$ is defined, then $(s_1, s'_1) \xrightarrow{\gamma(a,b)}'' (s_2, s'_2)$;
- $\downarrow'' = \downarrow \times \downarrow'$;
- $s''_0 = (s_0, s'_0)$.

Definition 3.2.8 (Encapsulation). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. Let H be a set of actions. The *encapsulation* of T with respect to H, written $\partial_H(T)$, is the transition system $(S', A', \rightarrow', \downarrow', s'_0)$ where

- $S' = S$;
- $A' = A$;
- $\rightarrow' = \rightarrow \cap (S \times (A_\tau \setminus H) \times S)$;
- $\downarrow' = \downarrow$;
- $s'_0 = s_0$.

The definitions of parallel composition and encapsulation are, just like the definition of transition system above, nothing else but simple adjustments of the earlier definitions to cover transitions labeled with the silent step. In the definition of the communication function, the possibility of communication/synchronization involving the silent step was excluded. Similarly, it is not allowed to encapsulate the silent step.

Example 3.2.2. Suppose, for the sake of this example only, that we do allow the encapsulation of the silent step. Consider the following transition systems.



The first two transition systems (T_1 and T_2) are branching bisimulation equivalent. The third and fourth transition system are the result of encapsulation of the silent step applied to the first and second transition system respectively. Obviously, these transition systems are not branching bisimulation equivalent. Thus, the internal step that was supposed to be unobservable in the first transition system, has materialized through the application of encapsulation. The congruence property of branching bisimulation equivalence with respect to encapsulation would be violated!

Here is an example of silent steps in parallel composition and encapsulation.

Example 3.2.3. We consider the following two transition systems. As actions of the first transition system, we have $\mathsf{s}_1(0)$ and $\mathsf{s}_2(0)$. As states of the first transition system, we have natural numbers $i \in \{0, 1, 2, 3\}$, with 0 as initial state. As transitions of the first transition system, we have the following:

$$0 \xrightarrow{\tau} 1,\ 0 \xrightarrow{\tau} 2,\ 1 \xrightarrow{\mathsf{s}_1(0)} 3,\ 2 \xrightarrow{\mathsf{s}_2(0)} 3.$$

As actions of the second transition system, we have only $\mathsf{r}_1(0)$. As states of the second transition system, we have natural numbers $i \in \{0, 1\}$, with 0 as initial state. As transitions of the second transition system, we have the following:

$$0 \xrightarrow{\mathsf{r}_1(0)} 1.$$

The transition systems given in this example are represented graphically in Figure 3.7. Parallel composition of these transitions systems with respect to the standard communication function γ results in the transition system represented in Figure 3.7 as well. Subsequent encapsulation with respect to

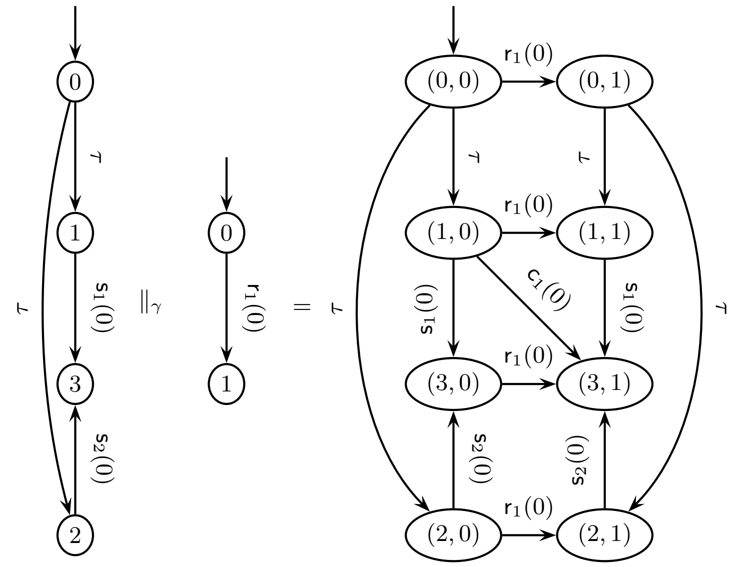


Fig. 3.7. Transition systems of Example 3.2.3

actions $\mathsf{s}_1(0)$, $\mathsf{r}_1(0)$, $\mathsf{s}_2(0)$ and $\mathsf{r}_2(0)$ results in the transition system in Figure 3.8. It has only one additional[1] action: $\mathsf{c}_1(0)$. As reachable states of the resulting transition system, we have the pairs $(0,0)$, $(1,0)$, $(2,0)$ and $(3,1)$, with $(0,0)$ as initial state. As transitions of the resulting transition system, we have the following:

$$(0,0) \xrightarrow{\tau} (1,0),\ (0,0) \xrightarrow{\tau} (2,0),\ (1,0) \xrightarrow{\mathsf{c}_1(0)} (3,1).$$

The resulting transition system is capable of either first performing a silent step, next performing a communication action and then becoming inactive or first performing a silent step and then becoming inactive. In the case where the send actions of the first transition system were not preceded by a silent step, the resulting transition system would only have the first alternative.

Parallel composition and encapsulation of transition systems are still preserved by the equivalences introduced in Chapter 1 and redefined in this section.

Property 3.2.3. Let T_1, T_2, T_1', and T_2' be transition systems, let γ be a communication function, and let H be a set of actions. Then the following hold:

[1] Recall from the formal definitions of parallel composition that the actions of the transition systems that are composed by means of parallel composition are also actions of the resulting transition system. The same holds for encapsulation.

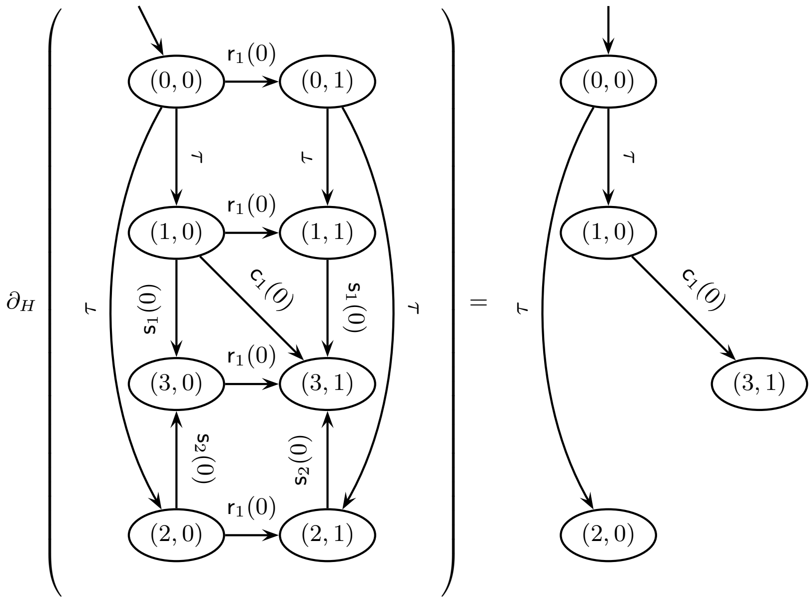


Fig. 3.8. Transition systems of Example 3.2.3

- if $T_1 \cong T_2$ and $T_1' \cong T_2'$, then $T_1 \parallel_\gamma T_1' \cong T_2 \parallel_\gamma T_2'$;
- if $T_1 \equiv_{\mathrm{l}} T_2$ and $T_1' \equiv_{\mathrm{l}} T_2'$, then $T_1 \parallel_\gamma T_1' \equiv_{\mathrm{l}} T_2 \parallel_\gamma T_2'$;
- if $T_1 \equiv_{\mathrm{tr}} T_2$ and $T_1' \equiv_{\mathrm{tr}} T_2'$, then $T_1 \parallel_\gamma T_1' \equiv_{\mathrm{tr}} T_2 \parallel_\gamma T_2'$;
- if $T_1 \leftrightarroweq T_2$ and $T_1' \leftrightarroweq T_2'$, then $T_1 \parallel_\gamma T_1' \leftrightarroweq T_2 \parallel_\gamma T_2'$;
- if $T_1 \cong T_2$, then $\partial_H(T_1) \cong \partial_H(T_2)$;
- if $T_1 \equiv_{\mathrm{l}} T_2$, then $\partial_H(T_1) \equiv_{\mathrm{l}} \partial_H(T_2)$;
- if $T_1 \equiv_{\mathrm{tr}} T_2$, then $\partial_H(T_1) \equiv_{\mathrm{tr}} \partial_H(T_2)$;
- if $T_1 \leftrightarroweq T_2$, then $\partial_H(T_1) \leftrightarroweq \partial_H(T_2)$.

Let us now look at the formal definitions of abstraction from internal actions and branching bisimulation equivalence.

Definition 3.2.9. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. Let I be a set of actions. The *abstraction* of T with respect to I, written $\tau_I(T)$, is the transition system $(S', A', \rightarrow', \downarrow', s_0')$ where

- $S' = S$;
- $A' = A$;
- $\rightarrow'$ is the smallest subset of $S \times A'_\tau \times S$ such that:
  - if $s_1 \xrightarrow{a} s_2$ and $a \in I$, then $s_1 \xrightarrow{\tau}' s_2$,
  - if $s_1 \xrightarrow{a} s_2$ and $a \notin I$, then $s_1 \xrightarrow{a}' s_2$.
- $\downarrow' = \downarrow$;
- $s_0' = s_0$.

In the introduction of this chapter some examples of abstraction have already been discussed. In the following two sections more examples are treated in detail.

In the definition of branching bisimulation equivalence below we use the notations $s \xrightarrow{(a)} s'$ and $s \xrightarrow{\epsilon}\!\!\!\!\rightarrow s'$ which will be defined first. Recall from the definition of the generalized transition relation that $s \xrightarrow{\epsilon}\!\!\!\!\rightarrow s'$ indicates that state s' can be reached from state s by performing zero or more consecutive silent steps (τ-transitions).

Definition 3.2.10. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. The relation $\_ \xrightarrow{(\_)} \_ \subseteq S \times A_\tau \times S$ is, for all $s, s' \in S$ and $a \in A_\tau$, defined by $s \xrightarrow{(a)} s'$ if and only if $s \xrightarrow{a} s'$, or $a = \tau$ and $s = s'$.

Definition 3.2.11 (Branching bisimulation equivalence). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ be transition systems. Then a *branching bisimulation* B between T and T' is a binary relation $B \subseteq S \times S'$ such that the following conditions hold:

1. $B(s_0, s'_0)$;
2. whenever $B(s_1, s'_1)$ and $s_1 \xrightarrow{a} s_2$, then there are states $s, s'_2 \in S'$ such that $s'_1 \xrightarrow{\epsilon}\!\!\!\!\rightarrow' s$ and $s \xrightarrow{(a)}{}' s'_2$ and $B(s_1, s)$ and $B(s_2, s'_2)$;
3. whenever $B(s_1, s'_1)$ and $s'_1 \xrightarrow{a}{}' s'_2$, then there are states $s, s_2 \in S$ such that $s_1 \xrightarrow{\epsilon}\!\!\!\!\rightarrow s$ and $s \xrightarrow{(a)} s_2$ and $B(s, s'_1)$ and $B(s_2, s'_2)$;
4. whenever $B(s_1, s'_1)$ and $s_1 \downarrow$, then there is a state $s \in S'$ such that $s'_1 \xrightarrow{\epsilon}\!\!\!\!\rightarrow' s$ and $s \downarrow'$ and $B(s_1, s)$;
5. whenever $B(s_1, s'_1)$ and $s'_1 \downarrow'$, then there is a state $s \in S$ such that $s_1 \xrightarrow{\epsilon}\!\!\!\!\rightarrow' s$ and $s \downarrow$ and $B(s, s'_1)$.

The two transition systems T and T' are *branching bisimulation equivalent*, written $T \leftrightarrows_{\mathrm{b}} T'$, if there exists a branching bisimulation B between T and T'. A branching bisimulation between T and T is called a *branching auto-bisimulation* on T.

Here is an example of transition systems that are branching bisimulation equivalent.

Example 3.2.4. We consider again the transition system presented at the end of Example 3.1.1 concerning abstraction of two encapsulated parallel bounded buffers. As states, we have pairs (σ_1, σ_2) where σ_i $(i = 1, 2)$ is a sequence of data of which the length of is not greater than l_i. As actions, we have $\mathsf{add}_1(d)$ and $\mathsf{rem}_2(d)$ for each datum d. As transitions, we have the following:

- for each datum d and each state (σ_1, σ_2) with the length of σ_1 less than l_1, a transition $(\sigma_1, \sigma_2) \xrightarrow{\mathsf{add}_1(d)} (d\,\sigma_1, \sigma_2)$;
- for each datum d and each state $(\sigma_1, \sigma_2\,d)$, a transition $(\sigma_1, \sigma_2\,d) \xrightarrow{\mathsf{rem}_2(d)} (\sigma_1, \sigma_2)$;

- for each datum d and each state $(\sigma_1\, d, \sigma_2)$ with the length of σ_2 less than l_2, a transition $(\sigma_1\, d, \sigma_2) \xrightarrow{\tau} (\sigma_1, d\,\sigma_2)$.

Next, we consider the following transition system. As states, we have sequences of data of which the length is not greater than $l_1 + l_2$. We have the same actions as before. As transitions, we have the following:

- for each datum d and each state σ with the length of σ less than $l_1 + l_2$, a transition $\sigma \xrightarrow{\mathsf{add}_1(d)} d\,\sigma$;
- for each datum d and each state $\sigma\, d$, a transition $\sigma\, d \xrightarrow{\mathsf{rem}_2(d)} \sigma$.

These two transition systems are branching bisimulation equivalent. Take the following relation:

$$B = \{((\sigma_1, \sigma_2), \sigma_1\, \sigma_2) \mid |\sigma_1| \leq l_1, |\sigma_2| \leq l_2\}\ .$$

It is easy to see that B is a branching bisimulation. The important point here is that, for each transition $(\sigma_1\, d, \sigma_2) \xrightarrow{\tau} (\sigma_1, d\,\sigma_2)$ of the first transition system, the conditions imposed on a branching bisimulation permit that the states $(\sigma_1\, d, \sigma_2)$ and $(\sigma_1, d\,\sigma_2)$ are both related to the state $\sigma_1\, d\,\sigma_2$ of the second transition system.

In Figure 3.9, for the case that the capacities of the bounded buffers are both 1 and the only data involved are 0 and 1, the abstracted encapsulated parallel composition of the bounded buffers is given as well as the second transition system explained above. The branching bisimulation B is depicted as well.

Similar to the equivalences discussed before, branching bisimulation equivalence is an equivalence.

Property 3.2.4. Branching bisimulation equivalence is an equivalence relation.

Branching bisimulation equivalence distinguishes strictly less transition systems than bisimulation equivalence and strictly more transition systems than trace equivalence.

Property 3.2.5. Any two bisimulation equivalent transition systems are also branching bisimulation equivalent. Any two branching bisimulation equivalent transition systems are also trace equivalent.

Just as bisimulation equivalence, branching bisimulation equivalence is preserved by parallel composition and encapsulation. Moreover, it is preserved by abstraction.

Property 3.2.6. Let T_1, T_2, T_1' and T_2' be transition systems, let γ be a communication function, and H and I sets of actions. Then the following holds:

if $T_1 \leftrightarrows_{\mathrm{b}} T_2$ and $T_1' \leftrightarrows_{\mathrm{b}} T_2'$, then $T_1 \parallel_\gamma T_1' \leftrightarrows_{\mathrm{b}} T_2 \parallel_\gamma T_2'$;
if $T_1 \leftrightarrows_{\mathrm{b}} T_2$, then $\partial_H(T_1) \leftrightarrows_{\mathrm{b}} \partial_H(T_2)$ and $\tau_I(T_1) \leftrightarrows_{\mathrm{b}} \tau_I(T_2)$.

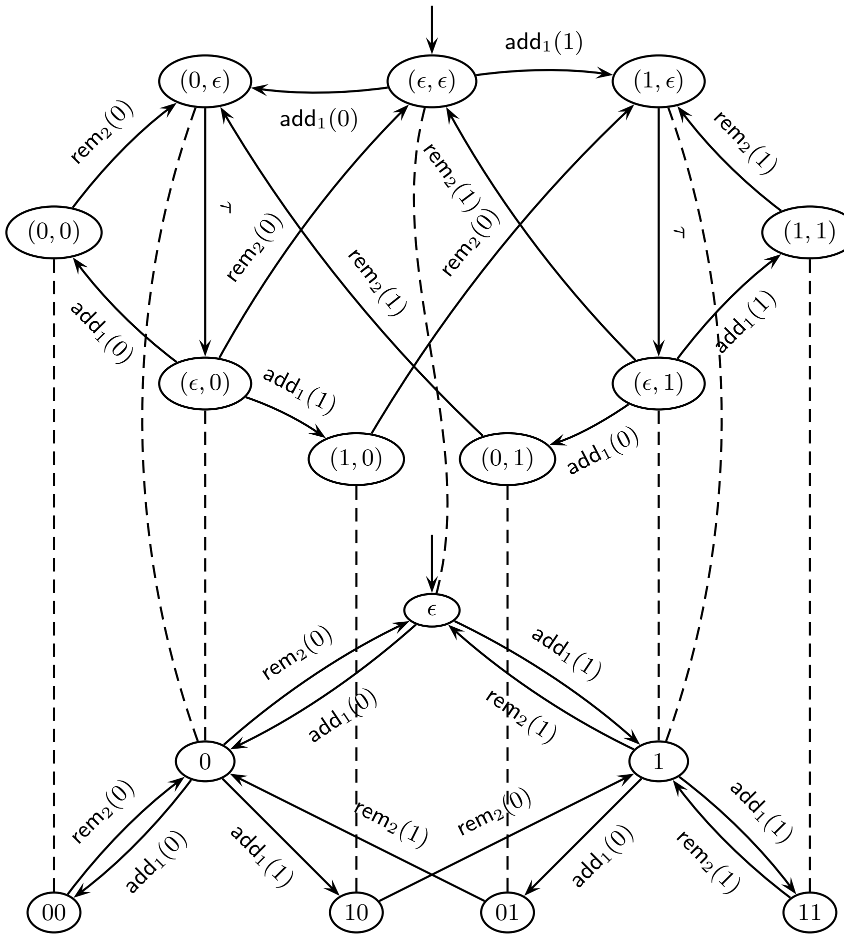
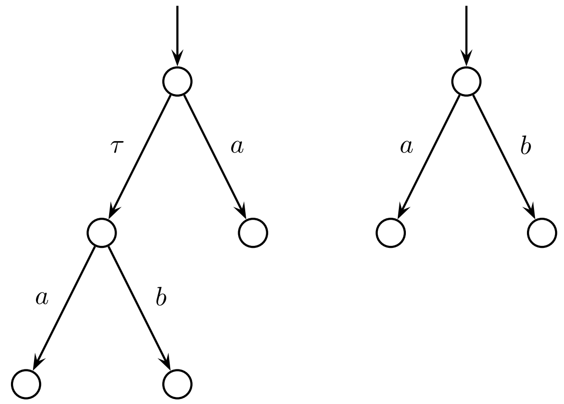


Fig. 3.9. Branching bisimulation

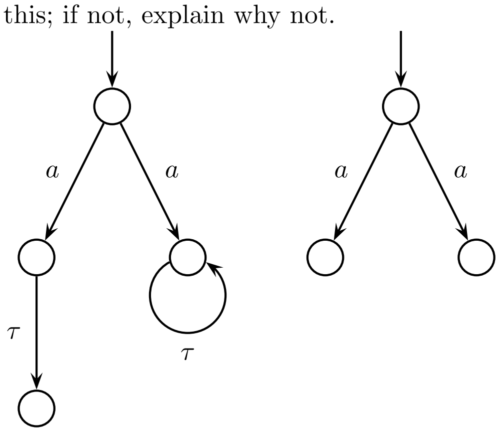
Property 3.2.7. Let T_1, T_2 and T_3 be transition systems. Let γ be a communication function. Then the following holds:

$$T_1 \parallel_\gamma T_2 \leftrightarroweq_{\mathrm{b}} T_2 \parallel_\gamma T_1 \ ,$$
$$(T_1 \parallel_\gamma T_2) \parallel_\gamma T_3 \leftrightarroweq_{\mathrm{b}} T_1 \parallel_\gamma (T_2 \parallel_\gamma T_3) \ .$$

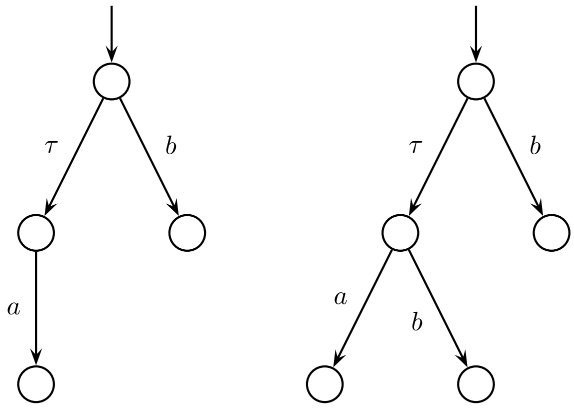
Exercise 3.2.1. Determine whether the following two transition systems are branching bisimulation equivalent; if so, give a branching bisimulation proving this; if not, explain why not.



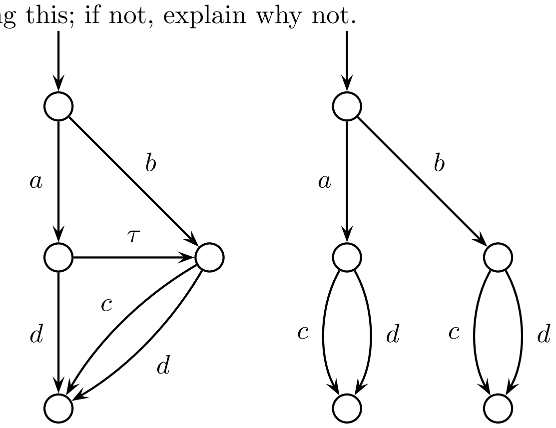
Exercise 3.2.2. Determine whether the following two transition systems are branching bisimulation equivalent; if so, give a branching bisimulation proving this; if not, explain why not.



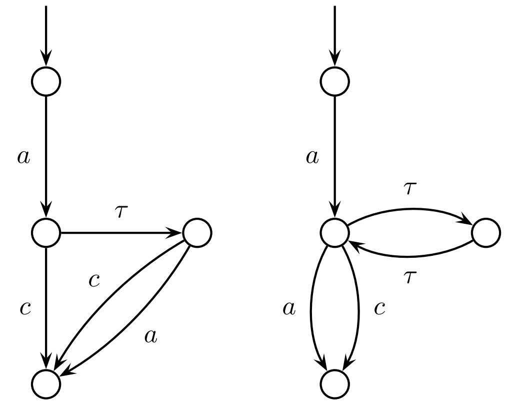
Exercise 3.2.3. Determine whether the following two transition systems are branching bisimulation equivalent; if so, give a branching bisimulation proving this; if not, explain why not.



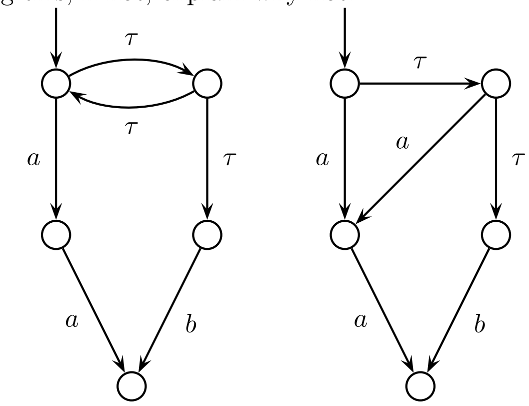
Exercise 3.2.4. Determine whether the following two transition systems are branching bisimulation equivalent; if so, give a branching bisimulation proving this; if not, explain why not.



Exercise 3.2.5. Determine whether the following two transition systems are branching bisimulation equivalent; if so, give a branching bisimulation proving this; if not, explain why not.



Exercise 3.2.6. Determine whether the following two transition systems are branching bisimulation equivalent; if so, give a branching bisimulation proving this; if not, explain why not.



Exercise 3.2.7. Verify whether the transition system given as a unit element for parallel composition in Exercise 2.5.1 is also a unit element for parallel composition with respect to branching bisimulation equivalence. Can you give at least two different (with respect to branching bisimulation equivalence) transition systems that satisfy $\tau_I(T) \mathrel{\underline{\leftrightarrow}}_{\mathrm{b}} T$ for any set of actions I?

Exercise 3.2.8. Give a counterexample for $\tau_I(T_1 \parallel_\gamma T_2) \mathrel{\underline{\leftrightarrow}}_{\mathrm{b}} \tau_I(T_1) \parallel_\gamma \tau_I(T_2)$, i.e., find I, γ, T_1, and T_2 such that $\tau_I(T_1 \parallel_\gamma T_2) \not\mathrel{\underline{\leftrightarrow}}_{\mathrm{b}} \tau_I(T_1) \parallel_\gamma \tau_I(T_2)$.

Exercise 3.2.9. Either prove or disprove (by means of a counterexample) the following statements about abstraction, for all transition systems T and all sets of actions I and J:

1. $\tau_I(\tau_I(T)) \mathrel{\underline{\leftrightarrow}}_{\mathrm{b}} \tau_I(T)$;
2. $\tau_I(\tau_J(T)) \mathrel{\underline{\leftrightarrow}}_{\mathrm{b}} \tau_J(\tau_I(T)) \mathrel{\underline{\leftrightarrow}}_{\mathrm{b}} \tau_{I \cup J}(T)$.

Exercise 3.2.10. Give a counterexample for $\tau_I(\partial_H(T)) \leftrightarrows_{\mathrm{b}} \partial_H(\tau_I(T))$. Which of the following statements hold?

1. $\tau_I(\partial_H(T)) \leftrightarrows_{\mathrm{b}} \tau_{I\setminus H}(\partial_H(T))$;
2. if $I \cap H = \emptyset$ then $\tau_I(\partial_H(T)) \leftrightarrows_{\mathrm{b}} \partial_H(\tau_I(T))$;
3. $\tau_I(\tau_J(T)) \leftrightarrows_{\mathrm{b}} \tau_{I\setminus J}(\tau_J(T))$.

3.3 Example: Merge connection with feedback wire

We consider again the merge connections from Example 1.5.10. Two transition systems describing the behaviour of a merge connection are given in that example. For clearness' sake, the second one is given here again. We assume a set of data D. The behaviour of a merge connection with input ports k and l and output port m, $\mathit{Merge}^{kl,m}$, is described by the following transition system. As states, we have $*$ and the data $d \in D$, with $*$ as initial state. As actions, we have $\mathsf{s}_i(d)$ and $\mathsf{r}_i(d)$ for $i = k, l, m$ and $d \in D$. As transitions, we have the following:

- for each $d \in D$: $* \xrightarrow{\mathsf{r}_k(d)} d$, $* \xrightarrow{\mathsf{r}_l(d)} d$, $d \xrightarrow{\mathsf{s}_m(d)} *$.

This transition system is represented graphically in Figure 3.10 for the case where $D = \{0, 1\}$.

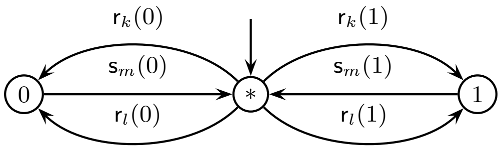


Fig. 3.10. Transition system for the merge connection

Wires, which were not mentioned before, constitute another important kind of connection used between nodes in networks. A wire is reminiscent of a buffer with unbounded capacity. The behaviour of a wire with input port m and output port l, $\mathit{Wire}^{m,l}$, is described by the following transition system. As states, we have all sequences $\sigma \in D^*$, with ϵ as initial state. As actions, we have $\mathsf{r}_m(d)$ and $\mathsf{s}_l(d)$ for each $d \in D$. As transitions of a wire, we have the following:

- for each $d \in D$ and $\sigma \in D^*$: $\sigma \xrightarrow{\mathsf{r}_m(d)} d\,\sigma$, $\sigma\, d \xrightarrow{\mathsf{s}_l(d)} \sigma$.

This transition system is represented graphically in Figure 3.11 for the case where $D = \{0, 1\}$.

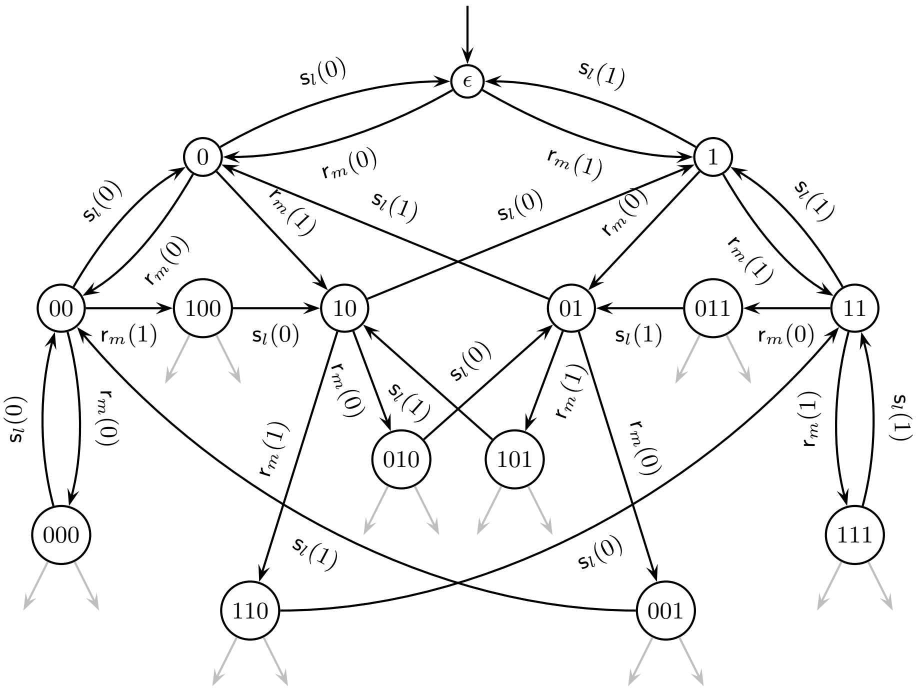


Fig. 3.11. Transition system for the wire

Let us look at the following transition system:

$$\tau_I(\partial_H(Merge^{kl,m} \parallel_\gamma Wire^{m,l}))$$

where

$$H = \{\mathsf{s}_i(d), \mathsf{r}_i(d) \mid i \in \{m, l\}, d \in D\} ,$$

$$I = \{\mathsf{c}_i(d) \mid i \in \{m, l\}, d \in D\}$$

and the communication function γ is defined in the standard way for handshaking communication (see Section 2.2), i.e. such that

$$\gamma(\mathsf{s}_i(d), \mathsf{r}_i(d)) = \gamma(\mathsf{r}_i(d), \mathsf{s}_i(d)) = \mathsf{c}_i(d)$$

for all $d \in D$, and it is undefined otherwise. Thus, the data delivered by the merge connection at port m is feed back to the input port l.

Parallel composition, encapsulation and abstraction of the transition systems $Merge^{kl,m}$ and $Wire^{m,l}$ as described above results in the following transtion system. As states, we have pairs (d, σ) where d and σ are states

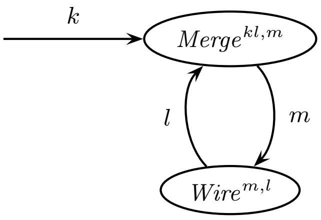


Fig. 3.12. Connection diagram for merge connection with feedback wire

of $Merge^{kl,m}$ and $Wire^{m,l}$, respectively. As initial state, we have $(*, \epsilon)$. As actions we have $\mathsf{r}_k(d)$ for each $d \in D$. As transitions we have the following:

- for each $d \in D$ and $\sigma \in D^*$: $(*, \sigma) \xrightarrow{\mathsf{r}_k(d)} (d, \sigma)$, $(d, \sigma) \xrightarrow{\tau} (*, d\,\sigma)$, and $(*, \sigma\, d) \xrightarrow{\tau} (d, \sigma)$.

This transition system is represented graphically in Figure 3.13 for the case where $D = \{0, 1\}$.

Let us also look at the following transition system. As states, we have only $*$. Consequently, $*$ is the initial state. As actions we have $\mathsf{r}_k(d)$ for each $d \in D$. As transitions we have the following:

- for each $d \in D$, a transition $* \xrightarrow{\mathsf{r}_k(d)} *$.

This transition system describes the behaviour of a special node in a network, viz. a *sink*. A sink consumes data, but does not deliver it anywhere.

These two transition systems are branching bisimulation equivalent. Take the following relation:

$$B = \{((d, \sigma), *) \mid d \in D \cup \{*\}, \sigma \in D^*\} .$$

It is easy to see that B is a branching bisimulation. The important point here is that, for each transition $(d, \sigma) \xrightarrow{\tau} (*, d\,\sigma)$ of the first transition system, the conditions imposed on a branching bisimulation permit that the states (d, σ) and $(*, d\,\sigma)$ are both related to the state $*$ of the second transition system; and for each transition $(*, \sigma\, d) \xrightarrow{\tau} (d, \sigma)$ of the first transition system, the conditions imposed on a branching bisimulation permit that the states $(*, d\,\sigma)$ and (d, σ) are both related to the state $*$ of the second transition system.

3.4 Example: Alternating bit protocol

We continue with the example of Section 2.4 concerning the ABP. At the end of that section, we presented the transition system that was the result of parallel composition and encapsulation of the transition systems of the sender S,

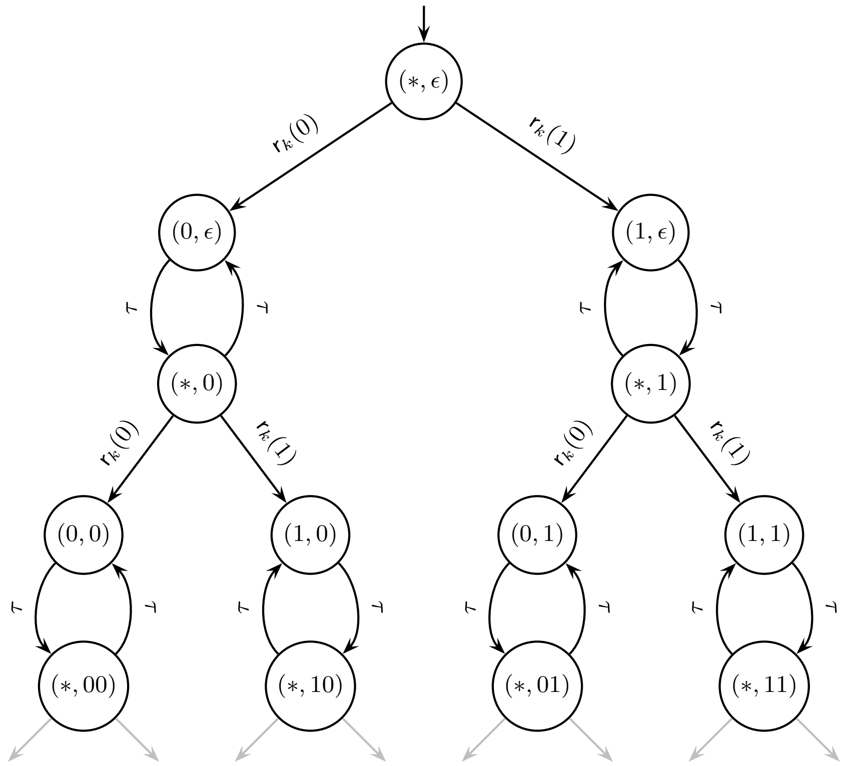


Fig. 3.13. Transition system for the merge connection with feedback wire

the data transmission channel K, the acknowledgement transmission channel L and the receiver R as described earlier in that section.

Most transitions of that transition system concern internal actions. The behaviour of the ABP after abstraction from the internal actions is described as follows:

$$\tau_I(\partial_H(S \parallel_\gamma K \parallel_\gamma L \parallel_\gamma R))$$

where

$$\begin{aligned} I = &\{\mathsf{c}_3(f) \mid f \in F\} \cup \{\mathsf{c}_4(f) \mid f \in F \cup \{*\}\} \\ &\cup \{\mathsf{c}_5(b) \mid b \in B \cup \{*\}\} \cup \{\mathsf{c}_6(b) \mid b \in B\} \cup \{\mathsf{i}\} \end{aligned}$$

and H and γ are as in Section 2.4. Parallel composition, encapsulation and abstraction of the transition systems of S, K, L and R as described above results in the following transition system. We have the same states as before. As actions, we have $\mathsf{r}_1(d)$ and $\mathsf{s}_2(d)$ for each $d \in D$. As transitions, we have:

- for each datum $d \in D$ and bit $b \in B$:

- $((*,b,0),(*,0),(*,0),(*,b,0)) \xrightarrow{\mathsf{r}_1(d)} ((d,b,1),(*,0),(*,0),(*,b,0))$,
- $((d,b,1),(*,0),(*,0),(*,b,0)) \xrightarrow{\tau} ((d,b,2),((d,b),1),(*,0),(*,b,0))$,
- $((d,b,2),((d,b),1),(*,0),(*,b,0)) \xrightarrow{\tau} ((d,b,2),((d,b),2),(*,0),(*,b,0))$,
- $((d,b,2),((d,b),2),(*,0),(*,b,0)) \xrightarrow{\tau} ((d,b,2),(*,0),(*,0),(d,b,1))$,
- $((d,b,2),(*,0),(*,0),(d,b,1)) \xrightarrow{\mathsf{s}_2(d)} ((d,b,2),(*,0),(*,0),(*,b,2))$,
- $((d,b,2),(*,0),(*,0),(*,b,2)) \xrightarrow{\tau} ((d,b,2),(*,0),(b,1),(*,\overline{b},0))$,
- $((d,b,2),(*,0),(b,1),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,2),(*,0),(b,2),(*,\overline{b},0))$,
- $((d,b,2),(*,0),(b,2),(*,\overline{b},0)) \xrightarrow{\tau} ((*,\overline{b},0),(*,0),(*,0),(*,\overline{b},0))$,

- $((d,b,2),((d,b),1),(*,0),(*,b,0)) \xrightarrow{\tau} ((d,b,2),((d,b),3),(*,0),(*,b,0))$,
- $((d,b,2),((d,b),3),(*,0),(*,b,0)) \xrightarrow{\tau} ((d,b,2),(*,0),(*,0),(*,\overline{b},2))$,
- $((d,b,2),(*,0),(*,0),(*,\overline{b},2)) \xrightarrow{\tau} ((d,b,2),(*,0),(\overline{b},1),(*,b,0))$,
- $((d,b,2),(*,0),(\overline{b},1),(*,b,0)) \xrightarrow{\tau} ((d,b,2),(*,0),(\overline{b},2),(*,b,0))$,
- $((d,b,2),(*,0),(\overline{b},2),(*,b,0)) \xrightarrow{\tau} ((d,b,1),(*,0),(*,0),(*,b,0))$,
- $((d,b,2),(*,0),(\overline{b},1),(*,b,0)) \xrightarrow{\tau} ((d,b,2),(*,0),(\overline{b},3),(*,b,0))$,
- $((d,b,2),(*,0),(\overline{b},3),(*,b,0)) \xrightarrow{\tau} ((d,b,1),(*,0),(*,0),(*,b,0))$,

- $((d,b,2),(*,0),(b,1),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,2),(*,0),(b,3),(*,\overline{b},0))$,
- $((d,b,2),(*,0),(b,3),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,1),(*,0),(*,0),(*,\overline{b},0))$,
- $((d,b,1),(*,0),(*,0),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,2),((d,b),1),(*,0),(*,\overline{b},0))$,
- $((d,b,2),((d,b),1),(*,0),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,2),((d,b),2),(*,0),(*,\overline{b},0))$,
- $((d,b,2),((d,b),2),(*,0),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,2),(*,0),(*,0),(*,b,2))$,
- $((d,b,2),((d,b),1),(*,0),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,2),((d,b),3),(*,0),(*,\overline{b},0))$,
- $((d,b,2),((d,b),3),(*,0),(*,\overline{b},0)) \xrightarrow{\tau} ((d,b,2),(*,0),(*,0),(*,b,2))$.

The transition system for the whole protocol is represented graphically in Figure 3.14 for the case where only one datum is involved.

Next, we consider the following transition system. As states, we have $d \in D \cup \{*\}$. The initial state is $*$. As actions, we have $\mathsf{r}_1(d)$ and $\mathsf{s}_2(d)$ for each $d \in D$. As transitions, we have the following:

- for each datum $d \in D$:
  - a transition $* \xrightarrow{\mathsf{r}_1(d)} d$,
  - a transition $d \xrightarrow{\mathsf{s}_2(d)} *$.

This transition system describes the behaviour of a bounded buffer with capacity 1, but the actions of a bounded buffer as introduced in Example 1.1.3 have been renamed.

The two transition systems presented above are branching bisimulation equivalent. For each $d \in D$, we define the sets $L(d)$ and $D(d)$ of states of the first transition system to be related to states d and $*$, respectively, of the second transition system:

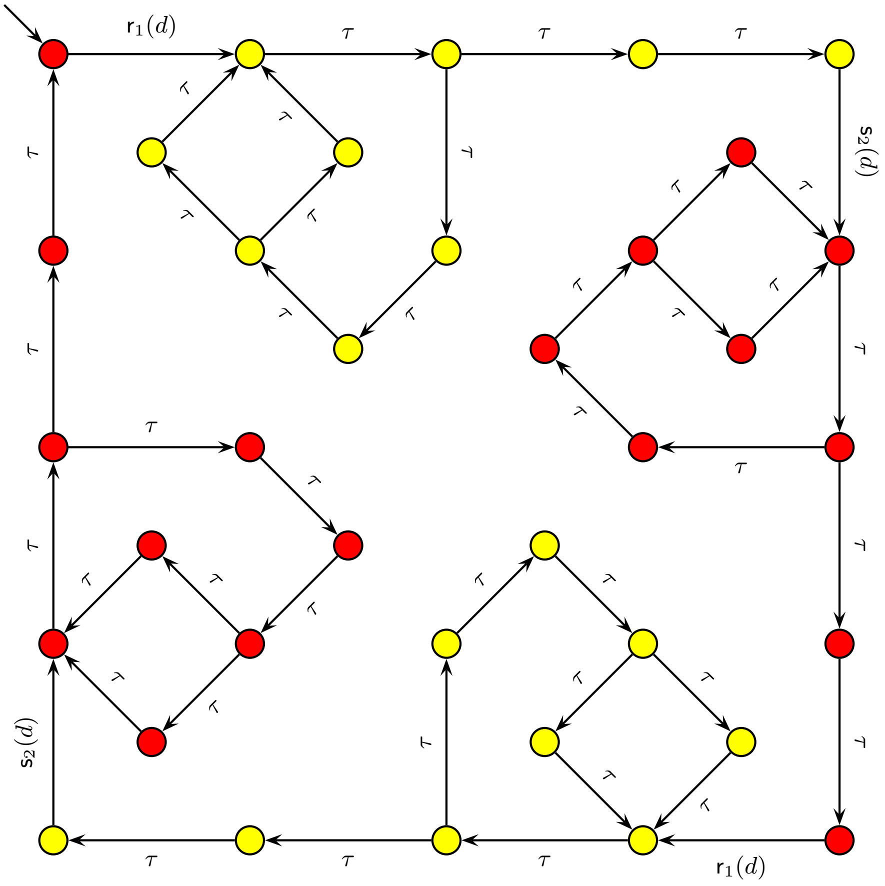


Fig. 3.14. Transition system for the ABP after abstraction from internal actions

$$
\begin{aligned}
&L(d)\\
&= \{((d,b,1),(*,0),(*,0),(*,b,0)),((d,b,2),((d,b),1),(*,0),(*,b,0)),\\
&\quad ((d,b,2),((d,b),2),(*,0),(*,b,0)),((d,b,2),(*,0),(*,0),(d,b,1)),\\
&\quad ((d,b,2),((d,b),1),(*,0),(*,b,0)),((d,b,2),((d,b),3),(*,0),(*,b,0)),\\
&\quad ((d,b,2),(*,0),(*,0),(*,\overline{b},2)),((d,b,2),(*,0),(\overline{b},1),(*,b,0)),\\
&\quad ((d,b,2),(*,0),(\overline{b},2),(*,b,0)),((d,b,2),(*,0),(\overline{b},1),(*,b,0)),\\
&\quad ((d,b,2),(*,0),(\overline{b},3),(*,b,0))\}\ ;
\end{aligned}
$$

$$
\begin{aligned}
&D(d)\\
&= \{((d,b,2),(*,0),(*,0),(*,b,2)),((d,b,2),(*,0),(b,1),(*,\overline{b},0)),\\
&\quad ((d,b,2),(*,0),(b,2),(*,\overline{b},0)),((*,b,0),(*,0),(*,0),(*,b,0)),\\
&\quad ((d,b,2),(*,0),(b,1),(*,\overline{b},0)),((d,b,2),(*,0),(b,3),(*,\overline{b},0)),\\
&\quad ((d,b,1),(*,0),(*,0),(*,\overline{b},0)),((d,b,2),((d,b),1),(*,0),(*,\overline{b},0)),\\
&\quad ((d,b,2),((d,b),2),(*,0),(*,\overline{b},0)),((d,b,2),((d,b),1),(*,0),(*,\overline{b},0)),\\
&\quad ((d,b,2),((d,b),3),(*,0),(*,\overline{b},0))\}\ .
\end{aligned}
$$

In Figure 3.14, the states from the sets $L(d)$ and $D(d)$ are coloured light and dark respectively. Next we define the relation B as follows:

$$B = \{(s,d) \mid d \in D, s \in L(d)\} \cup \{(s,*) \mid s \in \bigcup_{d\in D} D(d)\}\ .$$

It is straightforward to see that the conditions imposed on branching bisimulation equivalence permit that all states in $L(d)$ are related to state d and that all states in $\bigcup_{d\in D} D(d)$ are related to state $*$. In other words, the two transition systems presented above are branching bisimulation equivalent. That justifies the claim that, after abstraction from internal actions, the ABP behaves the same as a bounded buffer with capacity 1.

As a corollary, we have that the relation B' defined by

$$B' = \bigcup_{d\in D}(L(d) \times L(d)) \cup (\bigcup_{d\in D} D(d) \times \bigcup_{d\in D} D(d))$$

is a branching autobisimulation on the first transition system presented above. It is easy to show by means of B' that, although the transition systems for the channels are not determinate, the transition system for the whole protocol is determinate.

Exercise 3.4.1. Consider the simple protocol for reliable communication of data from Exercise 2.4.1. Apply abstraction of the actions from I where $I = \{\mathsf{c}_3(d), \mathsf{c}_4(d) \mid d \in D\}$ to the transition system that results from $\partial_H(S \parallel_\gamma C \parallel_\gamma R)$ that has been computed in Exercise 2.4.1. Draw the resulting transition system for the case that $D = \{0,1\}$. Prove that the protocol is branching bisimulation equivalent to the transition system for the one-place buffer (also with $D = \{0,1\}$).

Observe that this indicates that the "external behaviour" of the protocol described in this exercise is correct in the sense that any datum received from the environment (via channel 1) is delivered to the environment (via channel 3) in an alternating fashion.

4. Composition

In Chapter 2, we have seen that, by means of parallel composition, a transition system can be composed of others that act concurrently and interact with each other. This is not the only conceivable way of composition. This chapter treats several basic ways in which transition systems can be composed of others that do not interact with each other. Alternative composition is used to describe that a transition system is composed of two others that act the one or the other. Sequential composition is used to describe that a transition system is composed of two others that act successively. Iteration is used to describe that the behaviour of a transition system is repeated a number of times. Many transition systems can be composed using these three ways of composition. Thus, they support mastering the complexity of large transition systems. First of all, we explain informally what alternative composition, sequential composition and iteration are, and give simple examples of their use in describing process behaviour (Section 4.1). After that, we define alternative composition, sequential composition and iteration in a mathematically precise way (Section 4.2). We also use these operations to define the components of the simple data communication protocol from Section 2.4 (Section 4.3). Next, we have another look at branching bisimulation equivalence (Section 4.4) and discuss some properties of the compositions. Finally, we look at some miscellaneous issues (Section 4.5).

4.1 Informal explanation

The alternative composition of two transition systems T and T' is a transition system describing that there is a choice between the behaviour described by T and the behaviour described by T'. The choice is resolved at the instant that one of them performs its first action or decides to terminate successfully. The sequential composition of two transition systems T and T' is a transition system describing that the behaviour described by T and the behaviour described by T' follow each other. The iteration of transition system T is a transition system describing that initially there is a choice between the behaviour described by T and successful termination, and upon successful termination of T there is this choice again. Often, we need to describe that a transition system simply acts repeatedly for ever. The no-exit iteration of transition system T is a transition system describing that initially there is the behaviour described by T, and upon successful termination of T the behaviour is again as initially. Here are a couple of examples.

Example 4.1.1. We consider the simple telephone system from Example 1.1.1. Recall that in this telephone system each telephone is provided with a process, called its basic call process, to establish and maintain connections with other telephones. Actions of this process include receiving an off-hook or on-hook signal from the telephone, receiving a dialed number from the telephone, sending a signal to start or to stop emitting a dial tone, ring tone or ring-back tone to the telephone, and receiving an alert signal from another telephone – indicating an incoming call. Initially, there is a choice between the following two alternatives:

- receiving an off-hook signal from the telephone followed by a process of which the first action is sending a signal to start emitting a dial tone to the telephone;
- receiving an alert signal from another telephone followed by a process of which the first action is sending a signal to start emitting a ring tone to the telephone.

In either case the basic call process goes back to waiting for another off-hook or alert signal after the call is terminated. Therefore, the behaviour of the basic call process of a telephone can be described as the no-exit iteration of a process that is itself the alternative composition of two subprocesses, one reacting to an off-hook signal sent to the basic call process and the other reacting to an alert signal sent to the basic call process. The first one of these subprocesses first goes through a dialling phase and after that through a calling phase. So, the behaviour of this process can itself be described as the sequential composition of a subprocess for the dialling phase and a subprocess for the calling phase. And so forth.

Example 4.1.2. In order to control telephone answering, the control component of an answering machine has to communicate with the recorder component of the answering machine, the telephone network, and the telephone connected with the answering machine. When an incoming call is detected, the answering is not started immediately:

- if the incoming call is broken off or the receiver of the telephone is lifted within a certain period, answering is discontinued;
- otherwise, an off-hook signal is issued to the network when this period has elapsed and after that a pre-recorded message is played.

Upon termination of the message, the recorder is started and a beep signal is issued to the network. The recorder is stopped when:

- either the call is broken off;

- or a certain time period has passed in the case where the call has not been broken off earlier.

Thereafter, an on-hook signal is issued to the network. The behaviour of the control component can be described as the no-exit iteration of a process that is itself the sequential composition of three subprocesses, one checking whether the receiver is not lifted when an incoming call is detected, one controlling the answering with the pre-recorded message, and one controlling the recording of a message from the caller. Each of these subprocesses must respond properly if the call is broken off prematurely. Therefore, the behaviour of each of them can be described as an alternative composition with one of the alternatives reacting to signals indicating that the call is broken off prematurely.

4.2 Formal definition of the compositions

We will always consider two transition systems the same if they are isomorphic. Because of this, the disjointness requirement on the sets of states that occur in the definitions of alternative composition and sequential composition given below does not cause any loss of generality. Moreover, it does not matter that an arbitrary fresh initial state is chosen in the case of alternative composition: up to isomorphism the result is independent of the particular choice.

Unreachable states, and transitions between them, are never really relevant to the behaviour described by the transition system. For example, transition systems that differ only with respect to unreachable states are isomorphic. In fact, we are only interested in the reachable part of transition systems. Let us now look at the formal definitions of alternative composition, sequential composition, and iteration.

Definition 4.2.1 (Alternative composition). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ be transition systems such that $S \cap S' = \emptyset$. The *alternative composition* of T and T', written $T + T'$, is the transition system $(S'', A'', \rightarrow'', \downarrow'', s''_0)$ where

- $S'' = \{s''_0\} \cup S \cup S'$;
- $A'' = A \cup A'$;
- $\rightarrow''$ is the smallest subset of $S'' \times A''_\tau \times S''$ such that:
  - for all $s \in S$ and $a \in A_\tau$: if $s_0 \xrightarrow{a} s$, then $s''_0 \xrightarrow{a}'' s$;
  - for all $s' \in S'$ and $a \in A_\tau$: if $s'_0 \xrightarrow{a}' s'$, then $s''_0 \xrightarrow{a}'' s'$;
  - for all $s_1, s_2 \in S$ and $a \in A_\tau$: if $s_1 \xrightarrow{a} s_2$, then $s_1 \xrightarrow{a}'' s_2$;
  - for all $s'_1, s'_2 \in S'$ and $a \in A_\tau$: if $s'_1 \xrightarrow{a}' s'_2$, then $s'_1 \xrightarrow{a}'' s'_2$;
- $\downarrow''$ is the smallest subset of S'' such that:
  - for all $s \in S$: if $s \downarrow$, then $s \downarrow''$;
  - for all $s' \in S'$: if $s' \downarrow'$, then $s' \downarrow''$;

  – if $s_0 \downarrow$ or $s'_0 \downarrow'$, then $s''_0 \downarrow''$;
- $s''_0 \notin S \cup S'$.

The following things should be noted about the definition of alternative composition. The alternative composition of transition systems T and T' has a fresh initial state $s''_0 \notin S \cup S'$. This fresh initial state adopts the transitions from the initial state of T and the transitions from the initial state of T'. However, the fresh initial state does not replace the initial states of T and T'. Thus, transitions to the initial state of T or T' do not lead to transitions to the fresh initial state. The latter transitions would imply that the choice, that should be there only initially, could come back later. Here is an example to illustrate that it is quite natural to look at certain real-life processes as the alternative composition of other processes.

Example 4.2.1. We consider a simple railroad crossing controller. An approach signal is sent to the controller as soon as a train passes a detector placed backward from the gate. An exit signal is sent to the controller as soon as the train passes another detector placed forward from the gate. The controller is able to receive approach and exit signals from the train detectors at any time. When the controller receives an approach signal, a lower signal must be sent to the gate. When the controller receives an exit signal, a raise signal must be sent to the gate. Suppose that A and E are the transition systems describing the behaviours of the subprocesses dedicated to receiving and handling an approach signal and an exit signal, respectively, in the case where the signal is received at the beginning of a cycle of the controller, i.e. when there is no previous signal being handled. Then the behaviour of one cycle of the controller is described by $A + E$.

Let us also give an example illustrating the details of alternative composition.

Example 4.2.2. We assume a set of data D, and two input ports k and l. For $d \in D$, let $R_k(d)$ and $R_l(d)$ be the transition systems $(S, A, \rightarrow, \downarrow, s_0)$ and $(S', A', \rightarrow', \downarrow', s'_0)$ where

$$\begin{array}{ll}
S \ = \{(k,*), (k,d)\}, & S' \ = \{(l,*), (l,d)\}, \\
A \ = \{\mathsf{r}_k(d)\}, & A' \ = \{\mathsf{r}_l(d)\}, \\
\rightarrow \ = \{(k,*) \xrightarrow{\mathsf{r}_k(d)} (k,d)\}, & \rightarrow' \ = \{(l,*) \xrightarrow{\mathsf{r}_l(d)} (l,d)\}, \\
\downarrow \ = \{(k,d)\}, & \downarrow' \ = \{(l,d)\}, \\
s_0 = (k,*), & s'_0 \ = (l,*).
\end{array}$$

The transition system $R_i(d)$ is capable of receiving d at port i and then terminating successfully ($i = k, l$). The alternative composition $R_k(d) + R_l(d)$ is the transition system $(S'', A'', \rightarrow'', \downarrow'', s''_0)$ where

$$\begin{aligned}
S'' &= \{(*,*),(k,*),(k,d),(l,*),(l,d)\},\\
A'' &= \{\mathsf{r}_k(d),\mathsf{r}_l(d)\},\\
\to'' &= \{(*,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (*,*) \xrightarrow{\mathsf{r}_l(d)} (l,d),\\
&\qquad (k,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (l,*) \xrightarrow{\mathsf{r}_l(d)} (l,d)\},\\
\downarrow'' &= \{(k,d),(l,d)\},\\
s_0'' &= (*,*).
\end{aligned}$$

This transition system is capable of receiving datum d at port k or l and then terminating successfully. Observe that this transition system has two unreachable states: $(k,*)$ and $(l,*)$. The alternative composition of $R_k(d)$ and $R_l(d)$ is represented graphically in Figure 4.1. As always the unreachable states are not represented graphically.

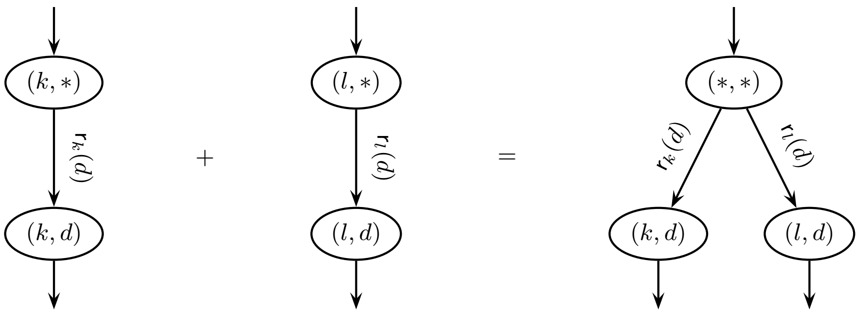


Fig. 4.1. Alternative composition of $R_k(d)$ and $R_l(d)$

Definition 4.2.2 (Sequential composition). Let $T = (S, A, \to, \downarrow, s_0)$ and $T' = (S', A', \to', \downarrow', s_0')$ be transition systems such that $S \cap S' = \emptyset$. The *sequential composition* of T and T', written $T \cdot T'$, is the transition system $(S'', A'', \to'', \downarrow'', s_0'')$ where

- $S'' = S \cup S'$;
- $A'' = A \cup A'$;
- $\to''$ is the smallest subset of $S'' \times A''_\tau \times S''$ such that:
  - for all $s_1, s_2 \in S$ and $a \in A_\tau$: if $s_1 \xrightarrow{a} s_2$, then $s_1 \xrightarrow{a}'' s_2$;
  - for all $s_1', s_2' \in S'$ and $a \in A_\tau'$: if $s_1' \xrightarrow{a}' s_2'$, then $s_1' \xrightarrow{a}'' s_2'$;
  - for all $s' \in S'$ and $a \in A_\tau'$: if $s_0' \xrightarrow{a}' s'$, then $s \xrightarrow{a}'' s'$ for every $s \in \downarrow$;
- $\downarrow''$ is the smallest subset of S'' such that:
  - for all $s' \in S'$: if $s' \downarrow'$, then $s' \downarrow''$;
  - for all $s \in S$: if $s_0' \downarrow'$ and $s \downarrow$, then $s \downarrow''$;
- $s_0'' = s_0$.

The definition of sequential composition is the first definition of a way in which transition systems can be composed where successfully terminating states are relevant to the transitions of the resulting transition system. Notice that, in the sequential composition of transition systems T and T', transitions from the initial state of T' and successful termination options from the initial state of T' are copied to transitions and successful termination options involving the successfully terminating states of T. Here is an example to illustrate that it is quite natural to look at certain real-life processes as the sequential composition of other processes.

Example 4.2.3. We look again at the railroad crossing controller from Example 4.2.1. Suppose that $R(\mathit{appr})$ and $R(\mathit{exit})$ are the transition systems describing the behaviours of the subprocesses dedicated to receiving an approach signal and an exit signal, respectively. Suppose that D and U are the transition systems describing the behaviours of the subprocesses dedicated to handling an approach signal and an exit signal, respectively, that is received at the beginning of a cycle of the controller. Then the behaviour of one cycle of the controller is described by $(R(\mathit{appr}) \cdot D) + (R(\mathit{exit}) \cdot U)$.

Let us also give an example illustrating the details of sequential composition.

Example 4.2.4. We assume a set of data D and one output port m. For $d \in D$, let $S_m(d)$ be the transition system $(S', A', \rightarrow', \downarrow', s'_0)$ where

$$\begin{aligned}
S' &= \{(m, d), (m, *)\}, \\
A' &= \{\mathsf{s}_m(d)\}, \\
\rightarrow' &= \{(m, d) \xrightarrow{\mathsf{s}_m(d)} (m, *)\}, \\
\downarrow' &= \{(m, *)\}, \\
s'_0 &= (m, d).
\end{aligned}$$

The transition system $S_m(d)$ is capable of sending datum d at port m and then terminating successfully. Let $R_k(d) + R_l(d)$ be as defined in Example 4.2.2. The sequential composition $(R_k(d) + R_l(d)) \cdot S_m(d)$ is the transition system $(S'', A'', \rightarrow'', \downarrow'', s''_0)$ where

$$\begin{aligned}
S'' &= \{(*, *), (k, *), (l, *), (k, d), (l, d), (m, d), (m, *)\}, \\
A'' &= \{\mathsf{r}_k(d), \mathsf{r}_l(d), \mathsf{s}_m(d)\}, \\
\rightarrow'' &= \{(*, *) \xrightarrow{\mathsf{r}_k(d)} (k, d), (*, *) \xrightarrow{\mathsf{r}_l(d)} (l, d), \\
&\qquad (k, *) \xrightarrow{\mathsf{r}_k(d)} (k, d), (l, *) \xrightarrow{\mathsf{r}_l(d)} (l, d), \\
&\qquad (m, d) \xrightarrow{\mathsf{s}_m(d)} (m, *), (k, d) \xrightarrow{\mathsf{s}_m(d)} (m, *), (l, d) \xrightarrow{\mathsf{s}_m(d)} (m, *)\}, \\
\downarrow'' &= \{(m, *)\}, \\
s''_0 &= (*, *).
\end{aligned}$$

This transition system is capable of receiving datum d at port k or l, next sending datum d at port m and then terminating successfully. The sequential composition of $R_k(d) + R_l(d)$ and $S_m(d)$ is represented graphically in Figure 4.2. Observe that the initial state of transition system $S_m(d)$ (i.e., the state (m, d)) is not reachable in the resulting transition system.

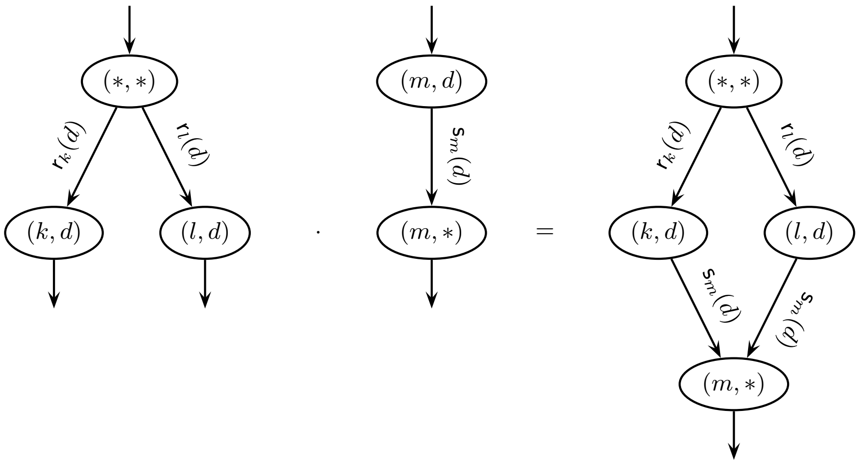


Fig. 4.2. Sequential composition of $R_k(d) + R_l(d)$ and $S_m(d)$

Definition 4.2.3 (Iteration). Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be a transition system. The *iteration* of T, written T^*, is the transition system $(S', A', \rightarrow', \downarrow', s'_0)$ where

- $S' = \{s'_0\} \cup S$;
- $A' = A$;
- $\rightarrow'$ is the smallest subset of $S' \times A'_\tau \times S'$ such that:
  - for all $s_1, s_2 \in S$ and $a \in A_\tau$: if $s_1 \xrightarrow{a} s_2$, then $s_1 \xrightarrow{a}' s_2$;
  - for all $s \in S$ and $a \in A_\tau$: if $s_0 \xrightarrow{a} s$, then $s'_0 \xrightarrow{a}' s$;
  - for all $s_1 \in S$ and $a \in A_\tau$: if $s_0 \xrightarrow{a} s_1$, then $s \xrightarrow{a}' s_1$ for each $s \in\downarrow$;
- $\downarrow' = \{s'_0\} \cup \downarrow$;
- $s'_0 \notin S$.

Like in the case of sequential composition, successfully terminating states are relevant to the transitions of the resulting transition system. In the case of iteration, the transitions from the initial state are copied to transitions from the successfully terminating states and to transitions from the new initial state. In this way, the choice, that is there initially, will come back after successful termination of T. Here is an example to illustrate that it is quite natural to look at certain real-life processes as the iteration of other processes.

Example 4.2.5. We look once more at the railroad crossing controller from Examples 4.2.1 and 4.2.3. In this example, we take into account that, because of fault tolerance considerations, approach signals should always cause the gate to go down, and exit signals should be ignored while the gate is going down. Suppose that $S(lower)$ is the transition system describing the behaviour of the subprocess dedicated to sending a lower signal. The behaviour of the subprocess dedicated to handling an approach signal that is received at the beginning of a cycle of the controller is described by

$(R(appr) + R(exit))^* \cdot S(lower)$. This is the transition system D referred to in Example 4.2.3.

Example 4.2.6. The iteration $((R_k(d) + R_l(d)) \cdot S_m(d))^*$ is the transition system $(S''', A''', \rightarrow''', \downarrow''', s_0''')$ where

$$\begin{aligned}
S''' &= \{*, (*,*), (k,*), (l,*), (k,d), (l,d), (m,d), (m,*)\},\\
A''' &= \{\mathsf{r}_k(d), \mathsf{r}_l(d), \mathsf{s}_m(d)\},\\
\rightarrow''' &= \{* \xrightarrow{\mathsf{r}_k(d)} (k,d), * \xrightarrow{\mathsf{r}_l(d)} (l,d),\\
&\quad (*,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (*,*) \xrightarrow{\mathsf{r}_l(d)} (l,d),\\
&\quad (k,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (l,*) \xrightarrow{\mathsf{r}_l(d)} (l,d),\\
&\quad (m,d) \xrightarrow{\mathsf{s}_m(d)} (m,*), (k,d) \xrightarrow{\mathsf{s}_m(d)} (m,*), (l,d) \xrightarrow{\mathsf{s}_m(d)} (m,*),\\
&\quad (m,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (m,*) \xrightarrow{\mathsf{r}_l(d)} (l,d)\},\\
\downarrow''' &= \{*, (m,*)\},\\
s_0''' &= *.
\end{aligned}$$

The transition system for $((R_k(d)+R_l(d)) \cdot S_m(d))^*$ is represented graphically in Figure 4.3.

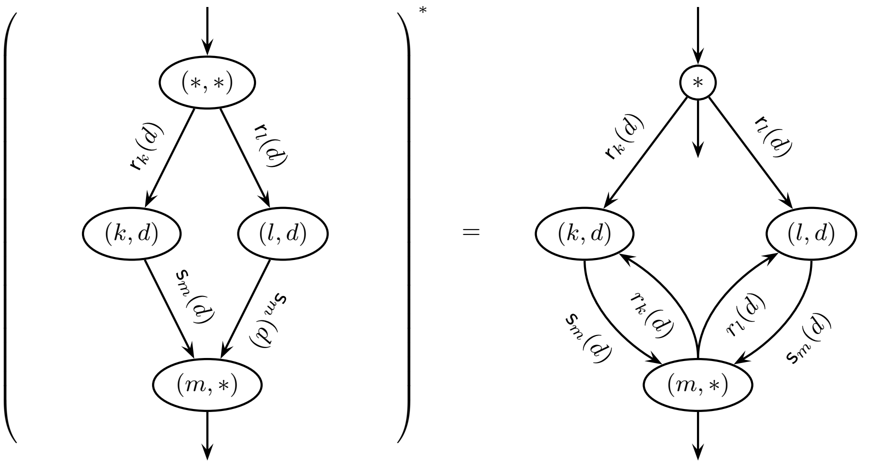


Fig. 4.3. Iteration of $(R_k(d) + R_l(d)) \cdot S_m(d)$

Definition 4.2.4 (No-exit iteration). Let T be a transition system. The *no-exit iteration* of T, written T^ω, is the transition system $T^* \cdot T'$ where T' is the transition system $(\{s_0\}, \emptyset, \emptyset, \emptyset, s_0)$.

Here is an example to illustrate that it is quite natural to look at certain real-life processes as the no-exit iteration of other processes.

Example 4.2.7. We look again at the railroad crossing controller from Examples 4.2.1, 4.2.3 and 4.2.5. The transition system $(R(appr) \cdot D) + (R(exit) \cdot U)$ from Example 4.2.3 describes the behaviour of one cycle of the controller. The behaviour of the controller is described by $((R(appr) \cdot D) + (R(exit) \cdot U))^{\omega}$.

Let us also give an example illustrating the details of no-exit iteration.

Example 4.2.8. Let $(R_k(d)$ and $R_l(d)) \cdot S_m(d)$ be as defined in Example 4.2.4. The no-exit iteration $((R_k(d) + R_l(d)) \cdot S_m(d))^{\omega}$ is the transition system $(S'''', A'''', \rightarrow'''', \downarrow'''', s_0'''')$ where

$$
\begin{aligned}
S'''' &= \{*, (*,*), (k,*), (l,*), (k,d), (l,d), (m,d), (m,*), **\},\\
A'''' &= \{\mathsf{r}_k(d), \mathsf{r}_l(d), \mathsf{s}_m(d)\},\\
\rightarrow'''' &= \{* \xrightarrow{\mathsf{r}_k(d)} (k,d), * \xrightarrow{\mathsf{r}_l(d)} (l,d),\\
&\quad (*,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (*,*) \xrightarrow{\mathsf{r}_l(d)} (l,d),\\
&\quad (k,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (l,*) \xrightarrow{\mathsf{r}_l(d)} (l,d),\\
&\quad (m,d) \xrightarrow{\mathsf{s}_m(d)} (m,*), (k,d) \xrightarrow{\mathsf{s}_m(d)} (m,*), (l,d) \xrightarrow{\mathsf{s}_m(d)} (m,*),\\
&\quad (m,*) \xrightarrow{\mathsf{r}_k(d)} (k,d), (m,*) \xrightarrow{\mathsf{r}_l(d)} (l,d)\},\\
\downarrow'''' &= \emptyset,\\
s_0'''' &= *.
\end{aligned}
$$

This transition system is branching bisimulation equivalent (it is even bisimulation equivalent) to the second transition system given for a merge connection in Example 1.5.10 in the case where D is a singleton set. The no-exit iteration of $(R_k(d) + R_l(d)) \cdot S_m(d)$ is represented graphically in Figure 4.4.

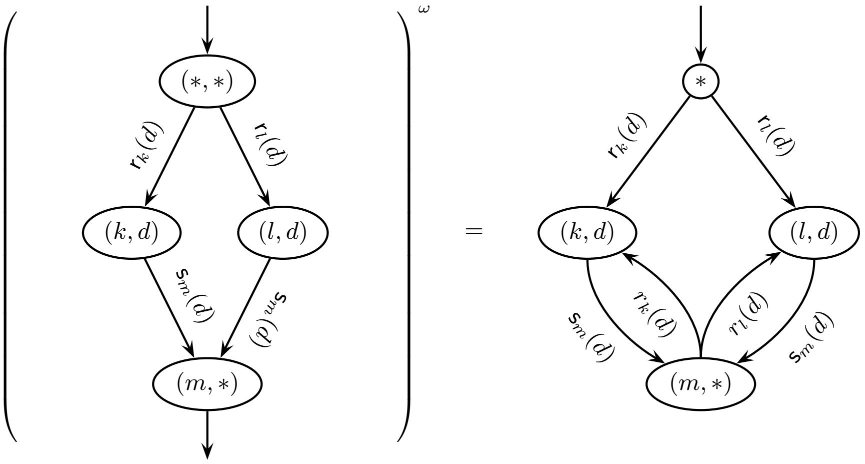


Fig. 4.4. No-exit iteration of $(R_k(d) + R_l(d)) \cdot S_m(d)$

Before we turn to more examples of the use of alternative composition, sequential composition and iteration, we will introduce atomic transition systems, i.e. transition systems that are capable of first performing a single action and then terminating successfully, and the inactive transition system, i.e. the transition system that is not capable of performing any action.

Definition 4.2.5. The *inactive* transition system is the transition system $(\{s_0\}, \emptyset, \emptyset, \emptyset, s_0)$ where s_0 is a fresh state. The inactive transition system is denoted by δ.

The *empty* transition system is the transition system $(\{s_0\}, \emptyset, \emptyset, \{s_0\}, s_0)$ where s_0 is a fresh state. The empty transition system is denoted by ϵ.

Let a be an action. The *atomic* transition system performing a is the transition system $(\{s_0, s_1\}, \{a\}, \{s_0 \xrightarrow{a} s_1\}, \{s_1\}, s_0)$ where s_0 and s_1 are fresh states. If no confusion can arise, the atomic transition system performing a is simply denoted by a.

The *atomic* transition system performing τ is the transition system $(\{s_0, s_1\}, \emptyset, \{s_0 \xrightarrow{\tau} s_1\}, \{s_1\}, s_0)$ where s_0 and s_1 are fresh states. If no confusion can arise, the atomic transition system performing τ is simply denoted by τ.

Bear in mind that it does not matter that arbitrary fresh states are chosen, as up to isomorphism the result is independent of the particular choice. Notice that the inactive transition system δ is used in the definition of no-exit iteration: $T^\omega = T^* \cdot \delta$.

Like for parallel composition, we use the convention of association to the left for alternative composition and sequential composition. The need to use parentheses is further reduced by ranking the precedence of the binary operations on transition systems. We adhere to the following precedence rules:

- the operation $+$ has lower precedence than all others;
- the operation $\cdot$ has higher precedence than all others;
- all other operations have the same precedence.

For example, we write $x \cdot z + y \cdot z$ for $(x \cdot z) + (y \cdot z)$.

Here are a couple of examples of the composition of transition systems starting from atomic transition systems. These examples show a way to present transition systems that is quite different from the way that we used before. It looks to be a more convenient way.

Example 4.2.9. We consider again the bounded buffer from Example 1.1.3. We restrict ourselves to the case where its capacity is 1 and it can only keep bits, i.e. $D = \{0, 1\}$. Using alternative composition, sequential composition and no-exit iteration, its behaviour can be described as follows:

$$(\mathsf{add}(0) \cdot \mathsf{rem}(0) + \mathsf{add}(1) \cdot \mathsf{rem}(1))^\omega \; .$$

Example 4.2.10. We consider again the split connection from Example 1.5.9 and the merge connection from Example 1.5.10. We restrict ourselves once more to the case where only bits are involved, i.e. $D = \{0, 1\}$. Using alternative composition, sequential composition and no-exit iteration, the behaviour of the split connection and the merge connection can be described as follows:

$$(\mathsf{r}_k(0) \cdot (\mathsf{s}_l(0) + \mathsf{s}_m(0)) + \mathsf{r}_k(1) \cdot (\mathsf{s}_l(1) + \mathsf{s}_m(1)))^\omega$$

and

$$((\mathsf{r}_k(0) + \mathsf{r}_l(0)) \cdot \mathsf{s}_m(0) + (\mathsf{r}_k(1) + \mathsf{r}_l(1)) \cdot \mathsf{s}_m(1))^\omega \ .$$

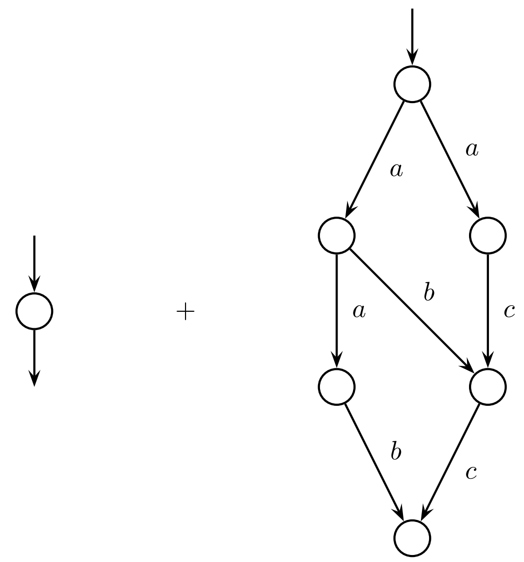
Here is another example, showing that the behaviour of simple PASCAL programs upon execution can also be described using alternative composition, sequential composition and iteration.

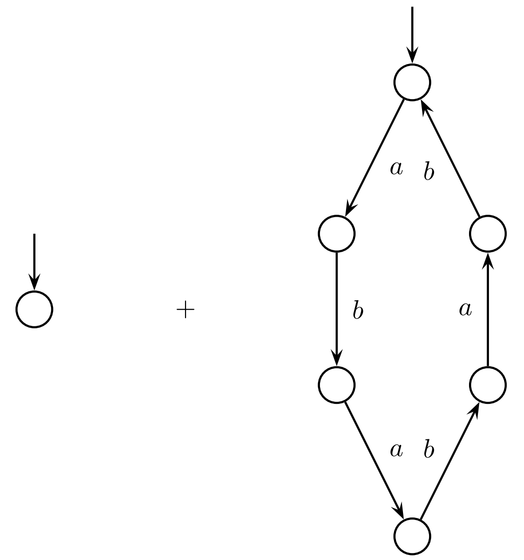
Example 4.2.11. We consider again the PASCAL program to calculate factorials from Example 1.3.1. Using alternative composition, sequential composition and iteration, the behaviour of this program upon abstract execution can be described as follows:

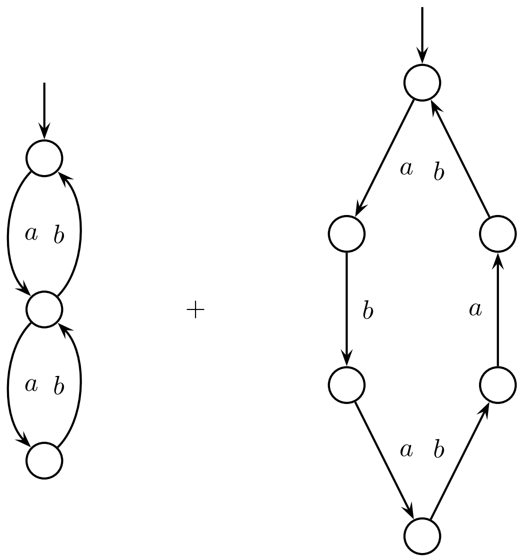
$$\begin{aligned}&(\mathsf{read(n)}) \cdot (\mathsf{i} := 0) \cdot (\mathsf{f} := 1) \cdot \\ &(((\mathsf{i} < \mathsf{n}) \cdot (\mathsf{i} := \mathsf{i} + 1) \cdot (\mathsf{f} := \mathsf{f} * \mathsf{i}))^* \cdot (\mathsf{NOT\,i} < \mathsf{n})) \cdot (\mathsf{write(f)}) \ .\end{aligned}$$

For reasons of readability, we have enclosed all atomic transition systems in parentheses. We cannot directly give a transition system describing the behaviour of a program upon execution on a machine by means of atomic transition systems, alternative composition, sequential composition and iteration. Nor we can give a transition system describing the behaviour of the machine on which the program is executed in this way. For the machine, as well as a category of simple programs, it is possible if we use in addition parallel composition, encapsulation and abstraction. However, it requires special tricks.

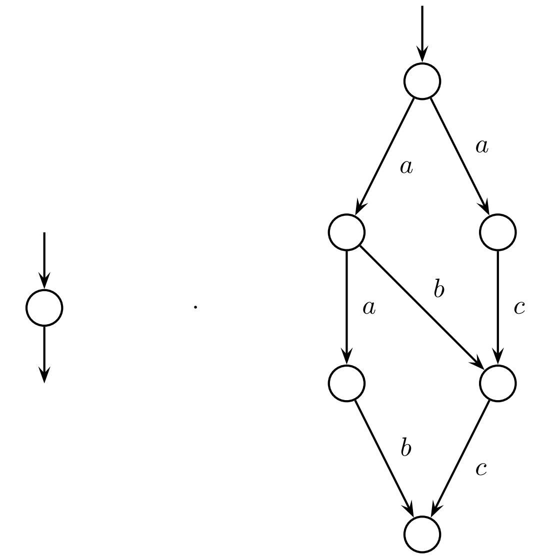
Exercise 4.2.1. Compute the following compositions using the formal definitions. Then give a graphical representation of the resulting transition systems.

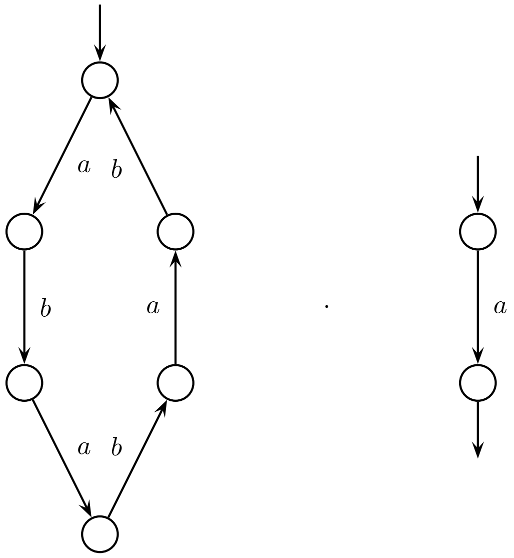

+
a
a
a
b
c
b
c

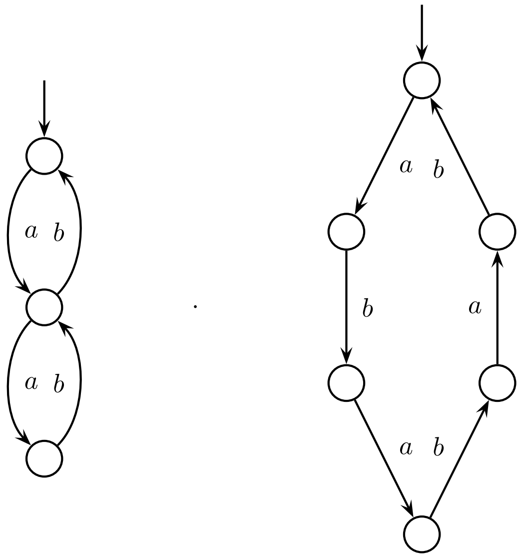

+
a
b
b
a
a
b



Exercise 4.2.2. Compute the following compositions using the formal definitions. Then give a graphical representation of the resulting transition systems.







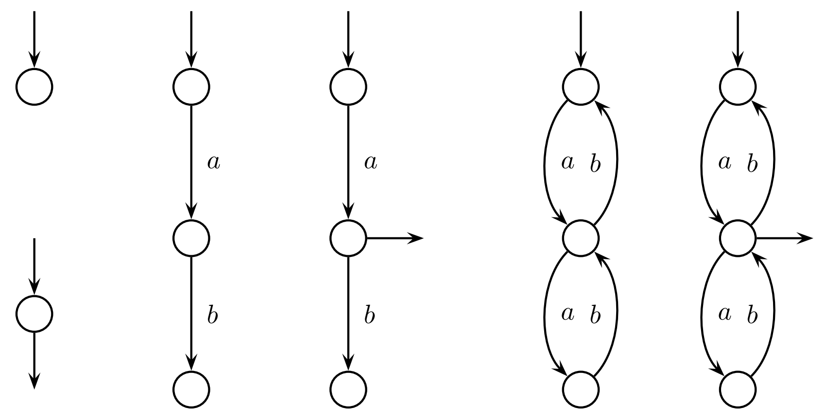
Exercise 4.2.3. Compute the following transition systems:

1. $a \cdot b + a \cdot \delta$
2. $a \cdot (b + \delta)$
3. $a \cdot b \cdot \epsilon \cdot d$
4. $a + b \cdot \epsilon + c \cdot d \cdot \delta + b$

Exercise 4.2.4. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ be an arbitrary transition system. Compute the transition system that results from the composition $T + \delta$.

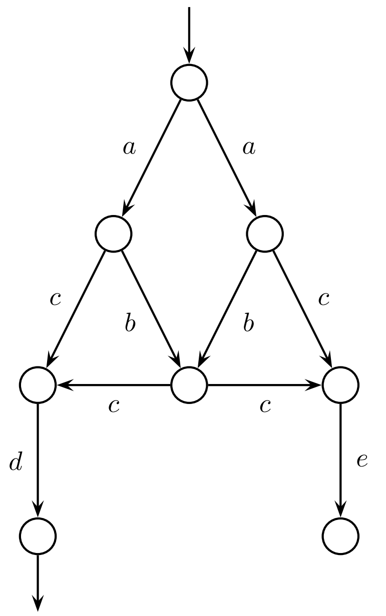
Under which of the equivalences defined so far are these transition systems equivalent?

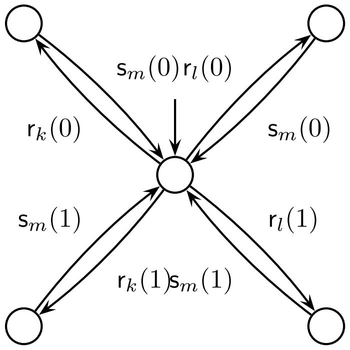
Exercise 4.2.5. Compute for each of the following transition systems the result of applying iteration to it. Do the same with respect to no-exit iteration.

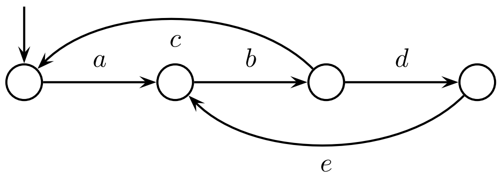


Exercise 4.2.6. Prove, using the formal definition of iteration, that for any transition system $T = (S, A, \rightarrow, \downarrow, s_0)$ it holds that $(T^*)^* \leftrightarrows T^*$.

Exercise 4.2.7. Give, for the following transition systems, an expression that is bisimulation equivalent with it.







Exercise 4.2.8. Let T_1, T_2 and T_3 be arbitrary transition systems. Consider the property $T_1 \cdot (T_2 + T_3) \leftrightarrows T_1 \cdot T_2 + T_1 \cdot T_3$. Give a counterexample that proves that the property does not hold. What about the property $T_1 \cdot (T_2 + T_3) \equiv_{\mathrm{tr}} T_1 \cdot T_2 + T_1 \cdot T_3$?

4.3 Example: Alternating bit protocol

We continue with the example of Section 2.4 and Section 3.4 concerning the ABP. Here, we describe the behaviour of the sender S, the data transmission channel K, the acknowledgement transmission channel L and the receiver R using alternative composition, sequential composition and iteration.

We restrict ourselves to the case where the set D of data is finite. Thus, we will use the following abbreviation. Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be an index set and T_i be a transition system for each $i \in \mathcal{I}$. Then we write $\sum_{i\in\mathcal{I}} T_i$ for $T_{i_1} + \ldots + T_{i_n}$. We further use the convention that $\sum_{i\in\mathcal{I}} T_i$ stands for δ if $\mathcal{I} = \emptyset$.

The behaviour of the sender S can be described as follows:

$$\left(\sum_{d\in D} \mathsf{r}_1(d) \cdot \mathsf{s}_3(d,0) \cdot (((\mathsf{r}_5(1) + \mathsf{r}_5(*)) \cdot \mathsf{s}_3(d,0))^* \cdot \mathsf{r}_5(0)) \cdot \right.$$
$$\left. \sum_{d\in D} \mathsf{r}_1(d) \cdot \mathsf{s}_3(d,1) \cdot (((\mathsf{r}_5(0) + \mathsf{r}_5(*)) \cdot \mathsf{s}_3(d,1))^* \cdot \mathsf{r}_5(1)) \right)^\omega$$

The behaviour of the receiver R can be described as follows:

$$\left(\left(\left(\left(\sum_{d\in D} \mathsf{r}_4(d,1) + \mathsf{r}_4(*) \right) \cdot \mathsf{s}_6(1) \right)^* \cdot \sum_{d\in D} \mathsf{r}_4(d,0) \right) \cdot \mathsf{s}_2(d) \cdot \mathsf{s}_6(0) \cdot \right.$$
$$\left. \left(\left(\left(\sum_{d\in D} \mathsf{r}_4(d,0) + \mathsf{r}_4(*) \right) \cdot \mathsf{s}_6(0) \right)^* \cdot \sum_{d\in D} \mathsf{r}_4(d,1) \right) \cdot \mathsf{s}_2(d) \cdot \mathsf{s}_6(1) \right)^\omega$$

The behaviour of the data transmission channel K can be described as follows:

$$\left(\sum_{f\in F} \mathsf{r}_3(f) \cdot (\mathsf{i} \cdot \mathsf{s}_4(f) + \mathsf{i} \cdot \mathsf{s}_4(*)) \right)^\omega$$

The behaviour of the acknowledgement transmission channel L can be described as follows:

$$\left(\sum_{b\in B} \mathsf{r}_6(b) \cdot (\mathsf{i} \cdot \mathsf{s}_5(b) + \mathsf{i} \cdot \mathsf{s}_5(*)) \right)^\omega$$

The transition systems for S, R, K and L presented above using alternative composition, sequential composition, iteration and no-exit iteration are (branching) bisimulation equivalent to the ones presented in Section 2.4.

Exercise 4.3.1. Compute the transition systems for the data transmission channel and the acknowledgement transmission channel using the above compositions. Are these isomorphic to the transition systems given for these processes in Section 2.4?

Exercise 4.3.2. Consider the transition systems from Exercise 2.2.5. Give expressions for the transition systems representing the sender and the three receivers.

4.4 Equivalences on processes

Now that we have introduced composition mechanisms that can be used in describing processes, the question arises to what extend the equivalences that have been introduced before are congruences with respect to these compositions. It turns out that all previously mentioned equivalences are a congruence with respect to sequential composition and no-exit iteration and all equivalences except for branching bisimulation equivalence are congruences for alternative composition and iteration.

Property 4.4.1. Let T_1, T_2, T_1', T_2' be transition systems. Then the following hold:

- if $T_1 \cong T_2$ and $T_1' \cong T_2'$, then $T_1 + T_1' \cong T_2 + T_2'$;
- if $T_1 \equiv_{\mathrm{l}} T_2$ and $T_1' \equiv_{\mathrm{l}} T_2'$, then $T_1 + T_1' \equiv_{\mathrm{l}} T_2 + T_2'$;
- if $T_1 \equiv_{\mathrm{tr}} T_2$ and $T_1' \equiv_{\mathrm{tr}} T_2'$, then $T_1 + T_1' \equiv_{\mathrm{tr}} T_2 + T_2'$;
- if $T_1 \leftrightarrows T_2$ and $T_1' \leftrightarrows T_2'$, then $T_1 + T_1' \leftrightarrows T_2 + T_2'$;
- if $T_1 \cong T_2$ and $T_1' \cong T_2'$, then $T_1 \cdot T_1' \cong T_2 \cdot T_2'$;
- if $T_1 \equiv_{\mathrm{l}} T_2$ and $T_1' \equiv_{\mathrm{l}} T_2'$, then $T_1 \cdot T_1' \equiv_{\mathrm{l}} T_2 \cdot T_2'$;
- if $T_1 \equiv_{\mathrm{tr}} T_2$ and $T_1' \equiv_{\mathrm{tr}} T_2'$, then $T_1 \cdot T_1' \equiv_{\mathrm{tr}} T_2 \cdot T_2'$;
- if $T_1 \leftrightarrows T_2$ and $T_1' \leftrightarrows T_2'$, then $T_1 \cdot T_1' \leftrightarrows T_2 \cdot T_2'$;
- if $T_1 \leftrightarrows_{\mathrm{b}} T_2$ and $T_1' \leftrightarrows T_2'$, then $T_1 \cdot T_1' \leftrightarrows_{\mathrm{b}} T_2 \cdot T_2'$;
- if $T_1 \cong T_2$, then $T_1{}^* \cong T_2{}^*$;
- if $T_1 \equiv_{\mathrm{l}} T_2$, then $T_1{}^* \equiv_{\mathrm{l}} T_2{}^*$;
- if $T_1 \equiv_{\mathrm{tr}} T_2$, then $T_1{}^* \equiv_{\mathrm{tr}} T_2{}^*$;
- if $T_1 \leftrightarrows T_2$, then $T_1{}^* \leftrightarrows T_2{}^*$;
- if $T_1 \cong T_2$, then $T_1^\omega \cong T_2^\omega$;
- if $T_1 \equiv_{\mathrm{l}} T_2$, then $T_1^\omega \equiv_{\mathrm{l}} T_2^\omega$;
- if $T_1 \equiv_{\mathrm{tr}} T_2$, then $T_1^\omega \equiv_{\mathrm{tr}} T_2^\omega$;
- if $T_1 \leftrightarrows T_2$, then $T_1^\omega \leftrightarrows T_2^\omega$;
- if $T_1 \leftrightarrows_{\mathrm{b}} T_2$, then $T_1^\omega \leftrightarrows_{\mathrm{b}} T_2^\omega$.

The following example illustrates that branching bisimulation equivalence is not a congruence for alternative composition.

Example 4.4.1. We consider again the transition systems from Example 3.1.2. We restrict ourselves to the case where only bits are involved, i.e. $D = \{0, 1\}$. Using atomic transition systems, alternative composition and sequential composition, they can be presented as follows:

$$\mathsf{r}_1(0) \cdot (\mathsf{s}_2(0) + \tau \cdot \mathsf{s}_3(0)) + \mathsf{r}_1(1) \cdot (\mathsf{s}_2(1) + \tau \cdot \mathsf{s}_3(1))$$

and

$$\mathsf{r}_1(0) \cdot (\mathsf{s}_2(0) + \mathsf{s}_3(0)) + \mathsf{r}_1(1) \cdot (\mathsf{s}_2(1) + \mathsf{s}_3(1)) \ .$$

The second case is the first case with $\tau \cdot \mathsf{s}_3(d)$ replaced by $\mathsf{s}_3(d)$ for $d \in \{0, 1\}$. The latter two transition systems are branching bisimulation equivalent, but the former two are not as explained in Example 3.1.2. Hence, branching bisimulation equivalence fails to be a congruence with respect to alternative composition.

This anomaly can simply be resolved by requiring that the initial states are related as in the case of standard bisimulation equivalence.

Definition 4.4.1. Let $T = (S, A, \rightarrow, \downarrow, s_0)$ and $T' = (S', A', \rightarrow', \downarrow', s'_0)$ be transition systems. If B is a branching bisimulation between T and T', then we say that a pair $(s_1, s'_1) \in S \times S'$ satisfies the *root condition* in B if the following conditions hold:

1. whenever $s_1 \xrightarrow{a} s_2$, then there is a state s'_2 such that $s'_1 \xrightarrow{a}' s'_2$ and $B(s_2, s'_2)$;
2. whenever $s'_1 \xrightarrow{a}' s'_2$, then there is a state s_2 such that $s_1 \xrightarrow{a} s_2$ and $B(s_2, s'_2)$;
3. whenever $s_1 \downarrow$, then $s'_1 \downarrow'$;
4. whenever $s'_1 \downarrow'$, then $s_1 \downarrow$.

The two transition systems T and T' are *rooted branching bisimulation equivalent*, written $T \leftrightarrows_{\mathrm{rb}} T'$, if there exists a branching bisimulation B between T and T' such that the pair (s_0, s'_0) satisfies the root condition in B.

Similar to the equivalences discussed before, rooted branching bisimulation equivalence is an equivalence.

Property 4.4.2. Rooted branching bisimulation equivalence is an equivalence relation.

Rooted branching bisimulation equivalence distinguishes strictly less transition systems than bisimulation equivalence and strictly more transition systems than branching bisimulation equivalence.

Property 4.4.3. Any two bisimulation equivalent transition systems are also rooted branching bisimulation equivalent. Any two rooted branching bisimulation equivalent transition systems are also branching bisimulation equivalent.

Just as branching bisimulation equivalence, rooted branching bisimulation equivalence is preserved by parallel composition, encapsulation and abstraction. Moreover, it is preserved by alternative composition, sequential composition, iteration and no-exit iteration.

Property 4.4.4. Let T_1, T_2, T'_1, T'_2 be transition systems, let γ be a communication function, and let H and I be sets of actions. Then the following holds:

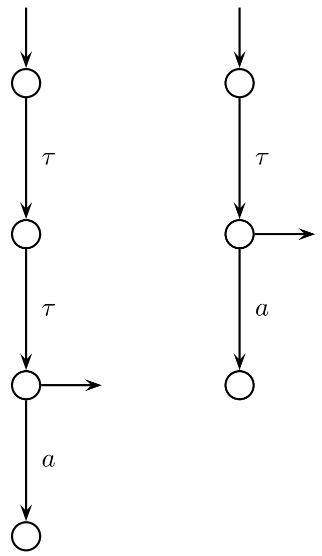
if $T_1 \leftrightarrows_{\mathrm{rb}} T_2$ and $T'_1 \leftrightarrows_{\mathrm{rb}} T'_2$, then $T_1 + T'_1 \leftrightarrows_{\mathrm{rb}} T_2 + T'_2$,
$T_1 \cdot T'_1 \leftrightarrows_{\mathrm{rb}} T_2 \cdot T'_2$, and $T_1 \parallel_\gamma T'_1 \leftrightarrows_{\mathrm{rb}} T_2 \parallel_\gamma T'_2$;
if $T_1 \leftrightarrows_{\mathrm{rb}} T_2$, then $\partial_H(T_1) \leftrightarrows_{\mathrm{rb}} \partial_H(T_2)$, $\tau_I(T_1) \leftrightarrows_{\mathrm{rb}} \tau_I(T_2)$,
${T_1}^* \leftrightarrows_{\mathrm{rb}} {T_2}^*$, and $T_1^\omega \leftrightarrows_{\mathrm{rb}} T_2^\omega$.

If we consider transition systems the same if they are rooted branching bisimulation equivalent, then both parallel composition and alternative composition are commutative and associative, and sequential composition is associative.

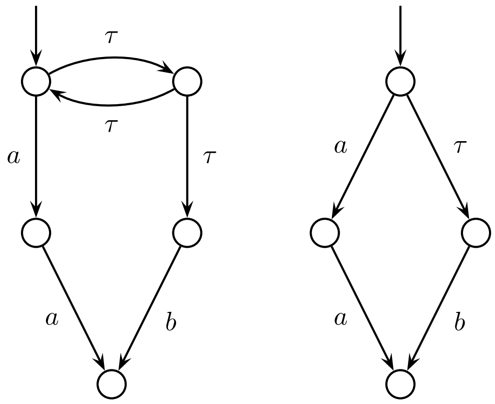
Property 4.4.5. Let T_1, T_2 and T_3 be transition systems. Let γ be a communication function. Then the following holds:

$$
\begin{aligned}
&T_1 + T_2 \leftrightarrows_{\mathrm{rb}} T_2 + T_1 \ , \\
&(T_1 + T_2) + T_3 \leftrightarrows_{\mathrm{rb}} T_1 + (T_2 + T_3) \ , \\
&(T_1 \cdot T_2) \cdot T_3 \leftrightarrows_{\mathrm{rb}} T_1 \cdot (T_2 \cdot T_3) \ , \\
&T_1 \parallel_\gamma T_2 \leftrightarrows_{\mathrm{rb}} T_2 \parallel_\gamma T_1 \ , \\
&(T_1 \parallel_\gamma T_2) \parallel_\gamma T_3 \leftrightarrows_{\mathrm{rb}} T_1 \parallel_\gamma (T_2 \parallel_\gamma T_3) \ .
\end{aligned}
$$

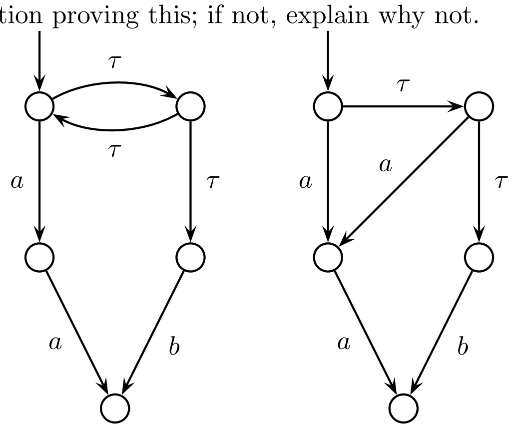
Exercise 4.4.1. Determine whether the following two transition systems are rooted branching bisimulation equivalent; if so, give a rooted branching bisimulation proving this; if not, explain why not.



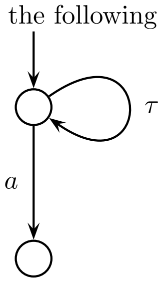
Exercise 4.4.2. Determine whether the following two transition systems are rooted branching bisimulation equivalent; if so, give a rooted branching bisimulation proving this; if not, explain why not.



Exercise 4.4.3. Determine whether the following two transition systems are rooted branching bisimulation equivalent; if so, give a rooted branching bisimulation proving this; if not, explain why not.



Exercise 4.4.4. Consider the following transition system:



Give an expression that is bisimulation equivalent with this transitions system. Then, give an expression without using iteration and no-exit iteration that is rooted branching bisimulation equivalent with the given transition system.

Exercise 4.4.5. Give a counterexample for the property that branching bisimulation equivalence is a congruence for iteration, i.e., give two transition systems T and T' such that $T \leftrightarroweq_{\mathrm{b}} T$" but not $T^* \leftrightarroweq_{\mathrm{b}} T'^*$.

Exercise 4.4.6. Consider the following pairs of compositions:

1. $a \cdot (\tau \cdot b + b)$ and $a \cdot b$
2. $a \cdot (\tau \cdot (b + c) + b)$ and $a \cdot (b + c)$
3. $\tau \cdot (a + b) + \tau \cdot a$ and $\tau \cdot (a + b)$
4. $a \cdot \tau \cdot (\tau \cdot b + \tau \cdot \tau \cdot b)$ and $a \cdot b$

For each of these, draw the corresponding transition systems and establish whether or not these are rooted branching bisimulation equivalent. If so, give a rooted branching bisimulation relation; if not, explain why not.

4.5 Miscellaneous

We have seen in Chapter 2 that it is slightly simpler to define parallel composition on nets than it is on transition systems. On the other hand, it is fairly complicated to define alternative composition, sequential composition and iteration on nets. For that reason, we will not show that alternative composition, sequential composition and iteration can be defined on nets as well.

A. Set theoretical preliminaries

In this appendix, we give a brief summary of facts from set theory used in this book. This will at least serve to establish the terminology and notation concerning sets. First of all, we treat elementary sets (Appendix A.1). After that, we look at relations, functions (Appendix A.2) and sequences (Appendix A.3).

A.1 Sets

A *set* is a collection of things which are said to be the *members* of the set. A set is completely determined by its members. That is, if two sets A and A' have the same members, then $A = A'$. We write $a \in A$ to indicate that a is a member of the set A, and $a \notin A$ to indicate that a is not a member of the set A. A set A is a *subset* of a set A', written $A \subseteq A'$ or $A' \supseteq A$, if for all x, $x \in A$ implies $x \in A'$.

If a set has a finite number of members a_1, …, a_n, then the set is written as follows:

$$\{a_1, \ldots, a_n\} .$$

Let $P(x)$ be the statement that x has property P. Then the set whose members are exactly the things that have property P, if such a set exists, is written as follows:

$$\{x \mid P(x)\} .$$

If A is a set and $P(x)$ is the statement that x has property P, then there exists a subset of A of which the members are exactly the members of A that have property P. This set is denoted by $\{x \in A \mid P(x)\}$:

$$\{x \in A \mid P(x)\} = \{x \mid x \in A \text{ and } P(x)\} .$$

If A is a set, then there exists a set of which the members are exactly the subsets of A. This set is called the *powerset* of A and is denoted by $\mathcal{P}(A)$:

$$\mathcal{P}(A) = \{x \mid x \subseteq A\} .$$

If $\mathcal{A}$ is a set of sets, then there exists a set of which the members are exactly the members of the subsets of $\mathcal{A}$. This set is called the *union* of $\mathcal{A}$ and is denoted by $\bigcup \mathcal{A}$:

$$\bigcup \mathcal{A} = \{x \mid \text{for some } A \in \mathcal{A}\colon x \in A\} .$$

There exists a set with no members. This set is called the *empty set* and is denoted by $\emptyset$:

$$\emptyset = \{x \mid x \neq x\} .$$

Let A and A' be sets. Then the usual set operations *union* ($\cup$), *intersection* ($\cap$) and *difference* ($\setminus$) are defined as follows:

$$\begin{aligned} A \cup A' &= \{x \mid x \in A \text{ or } x \in A'\} , \\ A \cap A' &= \{x \mid x \in A \text{ and } x \in A'\} , \\ A \setminus A' &= \{x \mid x \in A \text{ and } x \notin A'\} . \end{aligned}$$

If A and A' are sets, then there exists a set of which the members are exactly A and A'. This set is called the *unordered pair* of A and A' and is denoted by $\{A, A'\}$. Let A be a set, $a \in A$ and $a' \in A$. Then the *ordered pair*, or shortly pair, with *first element* a and *second element* a', written (a, a'), is the set defined as follows:

$$(a, a') = \{\{a\}, \{a, a'\}\} .$$

Let A and A' be sets. Then the set operation *cartesian product* ($\times$) is defined as follows:

$$A \times A' = \{(x, x') \mid x \in A \text{ and } x' \in A'\} .$$

This is extended in the obvious way to the cartesian product of more than two sets. An *ordered n-tuple* ($n > 2$), or shortly *n-tuple*, with *first element* a_1, ..., *nth element* a_n, written $(a_1, \ldots, a_n)$, is the set defined as follows:

$$(a_1, \ldots, a_n) = ((a_1, \ldots, a_{n-1}), a_n) .$$

A pair is sometimes also called a 2-tuple. Let A_1, ..., A_n be sets. Then the *cartesian product* of more than two sets is defined as follows:

$$A_1 \times \ldots \times A_n = \{(x_1, \ldots, x_n) \mid x_1 \in A_1, \ldots, x_n \in A_n\} .$$

If a set has a finite number of members, the set is said to be *finite*. We use the following abbreviation. We write $\mathcal{P}_{\text{fin}}(A)$ for $\{x \in \mathcal{P}(A) \mid x \text{ is finite}\}$, the set of all finite subsets of A.

As usual, we write $\mathbb{N}$ to denote the set of all natural numbers, and $\mathbb{B}$ to denote the set $\{\mathsf{tt}, \mathsf{ff}\}$ of all boolean values.

A.2 Relations and functions

Let $A_1, \ldots, A_n$ be sets. An *n-ary relation R between* $A_1, \ldots, A_n$ is a subset of $A_1 \times \ldots \times A_n$. If $A_1 = \ldots = A_n$, R is called an n-ary relation *on* A_1. We often write $R(a_1, \ldots, a_n)$ for $(a_1, \ldots, a_n) \in R$.

Let A be a set and R be a binary relation on A. Then we define the following:

- R is *reflexive* if $R(x, x)$ for all $x \in A$;
- R is *symmetric* if $R(x, y)$ implies $R(y, x)$ for all $x, y \in A$;
- R is *transitive* if $R(x, y)$ and $R(y, z)$ implies $R(x, z)$ for all $x, y, z \in A$;
- R is an *equivalence relation* on A if R is reflexive, symmetric and transitive.

Let A be a set and R be an equivalence relation on A. Then, for each $a \in A$, the set $\{x \in A \mid R(a, x)\}$ is called an *equivalence class* with respect to R. The members of an equivalence class are said to be *representatives* of the equivalence class.

Let A and A' be sets. Then a *function from A to A'* is a relation f between A and A' such that for all $x \in A$ there exists a unique $x' \in A'$ with $(x, x') \in f$. This x' is called the *value of f at x*. We write $f : A \to A'$ to indicate that f is a function from A to A', and we write $f(x)$ for the value of f at x.

A function $f : A \to A'$ is called *bijective* if

- for all $x, y \in A$: $f(x) = f(y)$ implies $x = y$;
- for all $x' \in A'$: $x' = f(x)$ for some $x \in A$.

If A, A' and A'' are sets, $A \subseteq A'$ and $f : A' \to A''$, then there exists a set of which the members are exactly the values of f at the members of A. This set is denoted by $\{f(x) \mid x \in A\}$:

$$\{f(x) \mid x \in A\} = \{x' \mid \text{for some } x \in A\text{: } f(x) = x'\} .$$

Let $\mathcal{I}$ be a set, $\mathcal{A}$ be a set of sets. Then a *family* indexed by $\mathcal{I}$ is a function $A : \mathcal{I} \to \mathcal{A}$. The set $\mathcal{I}$ is called the *index* set of the family. We write A_i for $A(i)$. If A is a family indexed by $\mathcal{I}$, then we write $\bigcup_{i \in \mathcal{I}} A_i$ for $\bigcup\{A_i \mid i \in \mathcal{I}\}$.

We also use the following abbreviation. We write $\{f(x) \mid x \in A, P(x)\}$ for $\{f(x) \mid x \in \{x' \in A \mid P(x')\}\}$.

A.3 Sequences

Let A be a set and $n \in \mathbb{N}$. Then a (finite) *sequence* over A of *length* n, is a function $\sigma : \{i \in \mathbb{N} \mid 1 \leq i \leq n\} \to A$. If $n > 0$ and $\sigma(1) = a_1, \ldots, \sigma(n) = a_n$, then the sequence is written as follows:

$$a_1 \ldots a_n .$$

The sequence of length 0 is called the *empty* sequence and is denoted by ϵ.

Let A be a set. Then the set of all sequences over A is denoted by A^*, and the set of all nonempty sequences over A is denoted by A^+. For each $\sigma \in A^*$, we write $|\sigma|$ for the length of σ.

Let A be a set, and $\sigma, \sigma' \in A^*$. Then the sequence operation *concatenation* ($\frown$) is defined as follows. $\sigma \frown \sigma'$ is the unique sequence $\sigma'' \in A^*$ with $|\sigma''| = |\sigma| + |\sigma'|$ such that:

$$\begin{aligned} &\sigma''(i) = \sigma(i) && \text{if } 1 \leq i \leq |\sigma| , \\ &\sigma''(i) = \sigma'(i - |\sigma|) && \text{if } |\sigma| + 1 \leq i \leq |\sigma| + |\sigma'| . \end{aligned}$$

We usually write $\sigma\,\sigma'$ for $\sigma \frown \sigma'$.

Let A be a set, and $\sigma, \sigma' \in A^*$. Then σ' is a *prefix* of σ, written $\sigma' \preceq \sigma$, if there exists a $\sigma'' \in A^*$ such that $\sigma'\,\sigma'' = \sigma$; and σ' is a *proper prefix* of σ, written $\sigma' \prec \sigma$, if $\sigma' \preceq \sigma$ and $\sigma' \neq \sigma$.