Chapter 5

Parallel processes

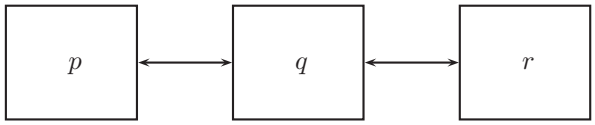
In chapter 4, we showed how it is possible to describe sequential processes that can interact with their environment. In this chapter we describe how to put these in parallel to describe and study the interaction between different processes.

The actions in two parallel processes happen independently of each other. Recall that we consider actions as atomic events in time. Hence, an action from the first process can happen before, after, or simultaneously with an action of the second process. This view on how parallel actions happen in called interleaving.

Below we first discuss processes that do not interact. In the subsequent sections we show how actions that happen simultaneously can be synchronised. By synchronising data values they can pass information to each other. In this way communication of data values is modelled.

5.1 The parallel operator

The parallel composition of two processes p and q is denoted by $p \parallel q$, and the binary (infix) operator $\parallel$ is called the *parallel operator*. This means that the actions in p happen independently of those in q. The process $p \parallel q$ terminates if both p and q can terminate. Three processes can simply be put in parallel by writing $p \parallel q \parallel r$. We require the parallel operator to be commutative and associative, and hence, the brackets around the parallel composition of several processes can be omitted. Typically, parallel processes are depicted as follows, where the arrows indicate how processes communicate.



In table 5.1 the axioms governing the behaviour of processes are found. It turns out that it is only possible to give a finite set of axioms, if auxiliary operators are introduced. The necessary operators are the leftmerge ($\between$) and the synchronisation merge ($|$).

M	$x \parallel y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + x \mid y$
LM1‡	$\alpha \lfloor\!\lfloor x = \alpha \cdot x$
LM2‡	$\delta \lfloor\!\lfloor x = \delta$
LM3‡	$\alpha \cdot x \lfloor\!\lfloor y = \alpha \cdot (x \parallel y)$
LM4	$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$
LM5	$(\sum_{d:D} X(d)) \lfloor\!\lfloor y = \sum_{d:D} X(d) \lfloor\!\lfloor y$
S1	$x \mid y = y \mid x$
S2	$(x \mid y) \mid z = x \mid (y \mid z)$
S3	$x \mid \tau = x$
S4	$\alpha \mid \delta = \delta$
S5	$(\alpha \cdot x) \mid \beta = \alpha \mid \beta \cdot x$
S6	$(\alpha \cdot x) \mid (\beta \cdot y) = \alpha \mid \beta \cdot (x \parallel y)$
S7	$(x + y) \mid z = x \mid z + y \mid z$
S8	$(\sum_{d:D} X(d)) \mid y = \sum_{d:D} X(d) \mid y$
TC1	$(x \lfloor\!\lfloor y) \lfloor\!\lfloor z = x \lfloor\!\lfloor (y \parallel z)$
TC2	$x \lfloor\!\lfloor \delta = x \cdot \delta$
TC3	$(x \mid y) \lfloor\!\lfloor z = x \mid (y \lfloor\!\lfloor z)$

Table 5.1: Axioms for the parallel composition operators

The process $p \lfloor\!\lfloor q$ (say p left merge q) is almost the same as the process $p \parallel q$ except that the first action must come from p.

The process $p|q$ (say p synchronises with q) is also the same as the process $p \parallel q$, except that the first action must happen simultaneously in p and q. Note that the symbol that we use for synchronisation between processes, is the same as the symbol used for combination of actions to multi-actions. Although, these are two different operators, we use them interchangeably as their meaning in both cases is the same. More concretely, $a|b$ both represents a multi-action and a synchronisation of two processes both consisting of a single action.

The axiom marked M in table 5.1 characterises our view on parallelism. The first action in $x \parallel y$ can either come from x, come from y or is an action that happens simultaneously in both of them. Axioms LM1‡ and LM3‡ state that the multi-action α must happen before any in the process x (and y) must do an action. Axiom LM2‡ expresses that a first action cannot come from δ. LM4 and LM5 indicate how parallel composition distributes over the sum and choice operator. The axioms that start with S allow to eliminate the communication merge.

As done elsewhere, the axioms that are only valid in an untimed setting are marked with a ‡. The variants valid in a timed setting can be found in chapter 8.

Consider the following process $a{\cdot}b \parallel c{\cdot}d$. Using the axioms in table 5.1, except TC1, TC2 and TC3, it is possible to remove parallel operators from expressions without variables and recursive behaviour in favour of the operators from chapter 4. This is called parallel expansion. We get:

$$
\begin{aligned}
&a{\cdot}b \parallel c{\cdot}d \overset{\mathrm{M}}{=} \\
&a{\cdot}b \lfloor\!\lfloor c{\cdot}d + c{\cdot}d \lfloor\!\lfloor a{\cdot}b + a{\cdot}b|c{\cdot}d \overset{\mathrm{LM3}‡,\mathrm{S6}}{=} \\
&a{\cdot}(b \parallel c{\cdot}d) + c{\cdot}(a{\cdot}b \parallel d) + a{\cdot}b|c{\cdot}d \overset{\mathrm{M}}{=} \\
&a{\cdot}(b \lfloor\!\lfloor c{\cdot}d + c{\cdot}d \lfloor\!\lfloor b + b|c{\cdot}d) + c{\cdot}(a{\cdot}b \lfloor\!\lfloor d + d \lfloor\!\lfloor a{\cdot}b + a{\cdot}b|d) + \\
&\qquad (a|c){\cdot}(b \parallel d) \overset{\mathrm{LM1}‡,\mathrm{LM3}‡,\mathrm{S6,M}}{=} \\
&a{\cdot}(b{\cdot}c{\cdot}d + c{\cdot}(b \parallel d) + (b|c){\cdot}d) + c{\cdot}(a{\cdot}(b \parallel d) + \\
&\qquad d{\cdot}a{\cdot}b + (a|d){\cdot}b) + (a|c){\cdot}(b \lfloor\!\lfloor d + d \lfloor\!\lfloor b + b|d) \overset{\mathrm{M,LM1}‡}{=} \\
&a{\cdot}(b{\cdot}c{\cdot}d + c{\cdot}(b{\cdot}d + d{\cdot}b + b|d) + (b|c){\cdot}d) + c{\cdot}(a{\cdot}(b{\cdot}d + d{\cdot}b + b|d) + \\
&\qquad d{\cdot}a{\cdot}b + (a|d){\cdot}b) + (a|c){\cdot}(b{\cdot}d + d{\cdot}b + b|d)
\end{aligned}
$$

In this expansion quite a number of axioms have been applied each time. Expansion is a very time consuming activity that shows how many options there are possible when parallel behaviour is involved. Later on, we treat ways to get rid of the parallel operator, without getting entangled in an axiomatic parallel expansion. Although not evident from the expansion above, parallel processes have a very typical structure, which becomes clear if the behaviour is plotted in a labelled transition system (see figure 5.1).

The axioms TC1, TC2 and TC3 are called axioms of true concurrency. They are useful to simplify expressions with parallel operators and variables.

The synchronisation operator binds stronger than all other binary operators. The parallel composition and left merge bind stronger than the sum and choice operator but weaker than the conditional operator.

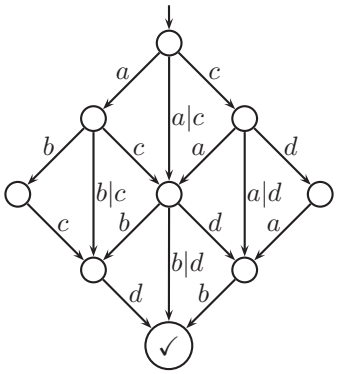


Figure 5.1: The behaviour of $a{\cdot}b \parallel c{\cdot}d$

Exercise 5.1.1. Expand the process $a{\cdot}b \parallel c$. Indicate precisely which axioms have been used.

Exercise 5.1.2. Give a rough estimate the size of the expansion of $a{\cdot}a{\cdot}a \parallel b{\cdot}b{\cdot}b \parallel c{\cdot}c{\cdot}c$.

Exercise 5.1.3. Prove that the parallel operator is both commutative and associative, i.e., $x \parallel y = y \parallel x$ and $x \parallel (y \parallel z) = (x \parallel y) \parallel z$.

5.2 Communication among parallel processes

Processes that are put in parallel can execute actions simultaneously, resulting in multi-actions. The communication operator $\Gamma_C(p)$ takes some actions out of a multi-action and replaces them with a single action, provided their data is equal. In this way it is made clear that these actions communicate or synchronise. Here C is a set of allowed communications of the form $a_1|\cdots|a_n{\to}c$, with $n{>}1$ and a_i and c are action names. For each communication $a_1|\cdots|a_n{\to}c$, the part of a multi-action consisting of $a_1(d)|\cdots|a_n(d)$ (for some d) in p is replaced by $c(d)$. Note that the data parameter must be equal for all communicating actions, and this data parameter is retained in action c. For example

$$\Gamma_{\{\,a|b\to c\,\}}(a(0)|b(0)) = c(0) \text{ and}$$
$$\Gamma_{\{\,a|b\to c\,\}}(a(0)|b(0)|d(0)) = c(0)|d(0).$$

If data is not equal no communication takes place: $\Gamma_{\{\,a|b\to c\,\}}(a(0)|b(1)) = a(0)|b(1)$. The axioms for the communication operator are given in table 5.2.

The function $\gamma_C(\alpha)$ applies the communications described by C to a multi-action α. It replaces every occurrence of a left-hand side of a communication it can find in α

C1 $\Gamma_C(\alpha) = \gamma_C(\alpha)$	C4 $\Gamma_C(x{\cdot}y) = \Gamma_C(x){\cdot}\Gamma_C(y)$
C2 $\Gamma_C(\delta) = \delta$	C5 $\Gamma_C(\sum_{d:D} X(d)) = \sum_{d:D} \Gamma_C(X(d))$
C3 $\Gamma_C(x{+}y) = \Gamma_C(x){+}\Gamma_C(y)$	

Table 5.2: Axioms for the communication operator

with the appropriate result. More precisely:

$$\begin{array}{lcl}
\gamma_\emptyset(\alpha) & = & \alpha \\
\gamma_{C_1 \cup C_2}(\alpha) & = & \gamma_{C_1}(\gamma_{C_2}(\alpha)) \\
\gamma_{\{a_1 \mid \cdots \mid a_n \to b\}}(\alpha) & = & \begin{cases} b(d) \mid \gamma_{\{a_1 \mid \cdots \mid a_n \to b\}}(\alpha \setminus (a_1(d)| \cdots |a_n(d))) \\ \qquad \text{if } a_1(d)| \cdots |a_n(d) \sqsubseteq \alpha \text{ for some } d. \\ \alpha \qquad \text{otherwise.} \end{cases}
\end{array}$$

For example, $\gamma_{\{a|b\to c\}}(a|a|b|c) = a|c|c$ and $\gamma_{\{a|a\to a, b|c|d\to e\}}(a|b|a|d|c|a) = a|a|e$.

An action cannot occur in two left-hand sides of allowed communications (e.g., $C = \{a|b{\to}c, a|d{\to}e\}$ is not allowed) and a right-hand side of a communication cannot occur in a left-hand side. Otherwise, $\gamma_{C_1}(\gamma_{C_2}(\alpha)) = \gamma_{C_2}(\gamma_{C_1}(\alpha))$ would not necessarily hold. In that case, $\gamma_{C_1 \cup C_2}(\alpha)$ is not uniquely defined and γ_C would not be a properly defined function.

When there are variables being used in actions, it cannot always directly be determined whether communication can take place. Consider

$$\Gamma_{\{a|b\to c\}}(a(d)|b(e)). \tag{5.1}$$

In order to determine whether a and b can communicate, it must be determined whether d and e are equal. Using $x = c'{\to}x{\diamond}x$ (see section 4.5) with $c' = d{\approx}e$, we rewrite (5.1) to

$$\Gamma_{\{a|b\to c\}}((d{\approx}e){\to}(a(d)|b(e)){\diamond}(a(d)|b(e))).$$

Now the communication can be applied to both sides, obtaining

$$(d{\approx}e){\to}c(d){\diamond}(a(d)|b(e)),$$

which can also be written as

$$(d{\approx}e){\to}c(d) + (d{\not\approx}e){\to}(a(d)|b(e)).$$

Exercise 5.2.1. Use the axioms for the communication operator to simplify the following process expressions.

1. $\Gamma_{\{a|b\to c\}}(a(1)|b(1)|d)$.
2. $\Gamma_{\{a|b\to c\}}(a(1)|b(2)|b(1))$.
3. $\Gamma_{\{a|b\to c\}}(a(d_1, d_2)|b(e_1, e_2))$.

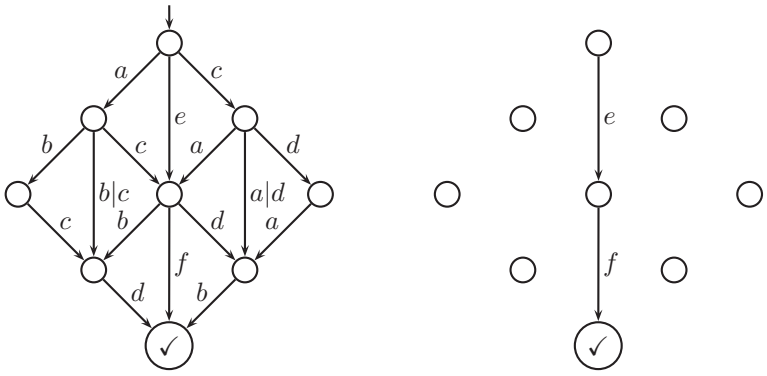


Figure 5.2: The behaviour of $\Gamma_{\{a|c\to e,b|d\to f\}}(a{\cdot}b\|c{\cdot}d)$, also with application of $\nabla_{\{e,f\}}$

Exercise 5.2.2. If the communications in the communication operator use the same actions in the left-hand sides, then the communication operator is not well defined. Show that

$$\Gamma_{\{a|b\to c,a|d\to e\}}(a|b|d)$$

can be simplified to two non bisimilar processes. Similarly, if the right-hand side of a communication overlaps with a left-hand side more outcomes are possible. Show also that

$$\Gamma_{\{a|b\to c,c|d\to e\}}(a|b|d)$$

can be simplified to non bisimilar processes.

5.3 The allow operator

The communication operator lets actions communicate when their data parameters are equal. But it cannot enforce communication. We must explicitly allow those actions that we want to see (namely the result of communications), and implicitly block other actions.

The *allow operator* $\nabla_V(p)$ is used for this purpose, where V is a set of multi-action names that specifies exactly which multi-actions from p are allowed to occur. For example $\nabla_{\{a,a|b\}}(a|b+a+b) = a+a|b$. The operator $\nabla_V(p)$ ignores the data parameters of the multi-actions in p, e.g., $\nabla_{\{b|c\}}(b(true,5)|c) = b(true,5)|c$. The empty multi-action τ is not allowed to occur in the set V because it cannot be blocked.

The axioms are given in table 5.3. The axioms V1 and V2 expresses how actions are allowed and blocked. Axiom TV1 allows to simplify complex constellations of allow expressions. All other axioms say that the allow operator distributes through a process expression.

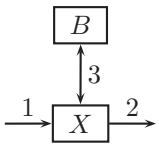
Consider again the process $a{\cdot}b\|c{\cdot}d$ as depicted in figure 5.1. Assume we want that action a communicates with c to e and b communicates with d to f. Then we

V1	$\nabla_V(\alpha) = \alpha$ if $\underline{\alpha} \in V \cup \{\tau\}$	V4	$\nabla_V(x + y) = \nabla_V(x) + \nabla_V(y)$
V2	$\nabla_V(\alpha) = \delta$ if $\underline{\alpha} \notin V \cup \{\tau\}$	V5	$\nabla_V(x \cdot y) = \nabla_V(x) \cdot \nabla_V(y)$
V3	$\nabla_V(\delta) = \delta$	V6	$\nabla_V(\sum_{d:D} X(d)) = \sum_{d:D} \nabla_V(X(d))$
TV1	$\nabla_V(\nabla_W(x)) = \nabla_{V \cap W}(x)$		

Table 5.3: Axioms for the allow operator

can first apply the operator $\Gamma_{\{a|c\to e,b|d\to f\}}$ to this process. We get the state space as depicted in figure 5.2 at the left. As a next operation we say that we only allow communications e and f to occur, effectively blocking all (multi-)actions in which an a, b, c or d occurs. The labelled transition system in figure 5.2 at the right belongs to the expression $\nabla_{\{e,f\}}(\Gamma_{\{a|c\to e,b|d\to f\}}(a \cdot b \parallel c \cdot d))$.

As a more realistic example we describe a system with a switching buffer X and a temporary store B. Data elements can be received by X via gate 1. An incoming datum is either sent on via gate 2, or stored in a one-place buffer B via gate 3. For sending an action via gate i we use the action s_i and for receiving data via gate i we use r_i.



The processes X and B are defined as follows:

act $r_1, s_2, s_3, r_3, c_3{:}D$;
proc $X = \sum_{d:D}(r_1(d) + r_3(d)) \cdot (s_2(d) + s_3(d)) \cdot X$;
$B = \sum_{d:D} r_3(d) \cdot s_3(d) \cdot B$;

Consider the behaviour $S = \nabla_{\{r_1,s_2,c_3\}}(\Gamma_{\{r_3|s_3\to c_3\}}(X \parallel B))$. In order to depict the labelled transition system we let D be equal to $\{d_1, d_2\}$. In figure 5.3 the behaviour of the processes X, B and S are drawn. Note that it is somewhat tedious to combine the behaviour of X and B and apply the communication and allow operator. In subsequent chapters we will provide different techniques to do this.

As we have the transition system of S we can answer a few questions about its behaviour. For instance, it is obvious that there are no deadlocks. It is also easy to see that reading at gate 1 and delivery at gate 2 does not necessarily have to take place in sequence. Reading more than two times at gate 1 without any intermediate delivery at gate 2 is also not possible. The system S can store at most two data elements.

Exercise 5.3.1. Data elements (from a set D) can be received by a one-place buffer X via gate 1, in which case they are sent on to a one-place buffer Y via gate 2. Y either

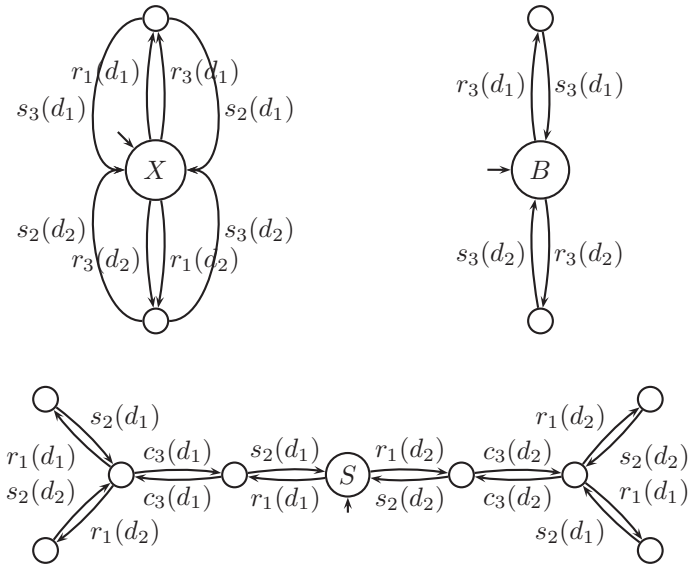
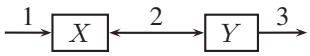


Figure 5.3: The LTSs of X, B and $\nabla_{\{r_1,s_2,c_3\}}(\Gamma_{\{r_3|s_3\to c_3\}}(X \parallel B))$

forwards an incoming datum via gate 3, or it returns this datum to X via gate 2. In the latter case, X returns the datum to Y via gate 2.



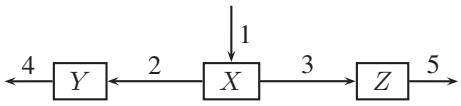
X and Y are defined by the following recursive specification:

act $r_1, s_2, r_2, c_2, s_3{:}D$;
proc $X = \sum_{d:D}(r_1(d) + r_2(d))\cdot s_2(d)\cdot X$;
$Y = \sum_{d:D} r_2(d)\cdot(s_3(d) + s_2(d))\cdot Y$;

Let S denote $\nabla_{\{r_1,c_2,s_3\}}(\Gamma_{\{s_2|r_2\to c_2\}}(X \parallel Y))$, and let D consist of $\{d_1, d_2\}$.

- Draw the state space of S.
- Are data elements read via gate 1 and sent in the same order via gate 3?
- Does $\nabla_{\{r_1,c_2\}}(S)$ contain a deadlock? If yes, give an execution trace to a deadlock state.

Exercise 5.3.2. Data elements (from a set D) can be received by a one-place buffer X via gate 1, in which case they are sent on in an alternating fashion to one-place buffers Y and Z via gates 2 and 3, respectively. So the first received datum is sent to Y, the second to Z, the third to Y, etc. Y and Z send on incoming data elements via gates 4 and 5, respectively.



(1) Specify the independent processes X, Y en Z and the parallel composition with the right communication and allow functions around it.

(2) Let D consist of a single element. Draw the state space.

5.4 Blocking and renaming

The *blocking operator* $\partial_B(p)$ (also known as the *encapsulation operator*) has the opposite effect of the allow operator. The set B contains action names that are not allowed. Any multi-action containing an action name in B is blocked. Blocking $\partial_B(p)$ does not have an effect on the data parameters of the actions in p when determining if an action should be blocked. E.g., $\partial_{\{b\}}(a(0) + b(true, 5)|c) = a(0)$. The blocking operator is sometimes used as an auxiliary operator, by blocking certain actions when analysing processes. For instance blocking the possibility to lose messages allows to get insight in the 'good weather' behaviour of a communication protocol more easily. The blocking operator is characterised by the axioms in table 5.4.

E1	$\partial_B(\tau) = \tau$	E5	$\partial_B(\delta) = \delta$
E2	$\partial_B(a(d)) = a(d)$ if $a \notin B$	E6	$\partial_B(x + y) = \partial_B(x) + \partial_B(y)$
E3	$\partial_B(a(d)) = \delta$ if $a \in B$	E7	$\partial_B(x \cdot y) = \partial_B(x) \cdot \partial_B(y)$
E4	$\partial_B(\alpha\|\beta) = \partial_B(\alpha)\|\partial_B(\beta)$	E8	$\partial_B(\sum_{d:D} X(d)) = \sum_{d:D} \partial_B(X(d))$
E10	$\partial_H(\partial_{H'}(x)) = \partial_{H \cup H'}(x)$		

Table 5.4: Axioms for the blocking operator

The *Renaming operator* ρ_R is used to rename action names. The set R contains renamings of the form $a \to b$. For a process $\rho_R(p)$ this means that every occurrence of action name a in p is replaced by action name b. Renaming $\rho_R(p)$ also disregards the data parameters. When a renaming is applied the data parameters are retained, e.g., $\rho_{\{a \to b\}}(a(0)+a) = b(0)+b$. To avoid ambiguities, every action name may only occur once as a left-hand side of a $a \to b$ in R. All renamings are applied simultaneously, i.e., a renamed action cannot be renamed twice in one application of the renaming operator. So $\rho_{\{a \to b, b \to c\}}$ renames action label a to b, not to c. The axioms are given in table 5.5.

Exercise 5.4.1. Simplify the following expressions.

1. $\partial_{\{a\}}(a + b + a|b)$.

R1	$\rho_R(\tau) = \tau$	
R2	$\rho_R(a(d)) = b(d)$	if $a{\rightarrow}b \in R$ for some b
R3	$\rho_R(a(d)) = a(d)$	if $a{\rightarrow}b \notin R$ for all b
R4	$\rho_R(\alpha\|\beta) = \rho_R(\alpha)\|\rho_R(\beta)$	
R5	$\rho_R(\delta) = \delta$	
R6	$\rho_R(x + y) = \rho_R(x) + \rho_R(y)$	
R7	$\rho_R(x{\cdot}y) = \rho_R(x){\cdot}\rho_R(y)$	
R8	$\rho_R(\sum_{d:D} X(d)) = \sum_{d:D} \rho_R(X(d))$	

Table 5.5: Axioms for the renaming operator

2. $\rho_{\{a\rightarrow b\}}(a + b + a|b)$.
3. $\rho_{\{a\rightarrow b\}}(\partial_{\{a,b\}}(a + b + a|b))$.

5.5 Hiding internal behaviour

As indicated in the previous chapter, hiding information is very important to obtain insight in the behaviour of processes. For this purpose the *hiding operator* τ_I is defined. The action names in the set I are removed from multi-actions. So, $\tau_{\{a\}}(a|b) = b$ and $\tau_{\{a\}}(a) = \tau$. The axioms for hiding are listed in table 5.6.

H1	$\tau_I(\tau) = \tau$	H5	$\tau_I(\delta) = \delta$
H2	$\tau_I(a(d)) = \tau$ if $a{\in}I$	H6	$\tau_I(x{+}y) = \tau_I(x) + \tau_I(y)$
H3	$\tau_I(a(d)) = a(d)$ if $a{\notin}I$	H7	$\tau_I(x{\cdot}y) = \tau_I(x){\cdot}\tau_I(y)$
H4	$\tau_I(\alpha\|\beta) = \tau_I(\alpha)\|\tau_I(\beta)$	H8	$\tau_I(\sum_{d:D} X(d)) = \sum_{d:D} \tau_I(X(d))$
H10	$\tau_I(\tau_{I'}(x)) = \tau_{I\cup I'}(x)$		

Table 5.6: Axioms for the hiding operator

It is convenient to be able to postpone hiding of actions, by first renaming them to a special visible action int which is subsequently renamed to τ. For this purpose the straightforward *pre-hide operator* Υ_U is defined where U is a set of action labels. All actions with labels in U are renamed to the action int and the data is removed. The important property of the pre-hide operator is that $\tau_{I\cup\{int\}}(x) = \tau_{\{int\}}(\Upsilon_I(x))$. The axioms for the pre-hide operator are in table 5.7.

As an example we apply hiding to the example in section 5.2 with a switching buffer and a temporary store. We may be interested in the communication at gates 1 and 2, but we are not interested how X and B exchange information on gate 3. So, we hide

U1	$\Upsilon_U(\tau) = \tau$	U5	$\Upsilon_U(\delta) = \delta$
U2	$\Upsilon_U(a(d)) = int$ if $a \in U$	U6	$\Upsilon_U(x+y) = \Upsilon_U(x) + \Upsilon_U(y)$
U3	$\Upsilon_U(a(d)) = a(d)$ if $a \notin U$	U7	$\Upsilon_U(x \cdot y) = \Upsilon_U(x) \cdot \Upsilon_U(y)$
U4	$\Upsilon_U(\alpha \mid \beta) = \Upsilon_U(\alpha) \mid \Upsilon_U(\beta)$	U8	$\Upsilon_U(\sum_{d:D} X(d)) = \sum_{d:D} \Upsilon_U(X(d))$
U10	$\Upsilon_U(\Upsilon_{U'}(x)) = \Upsilon_{U \cup U'}(x)$		

Table 5.7: Axioms for the pre-hiding operator

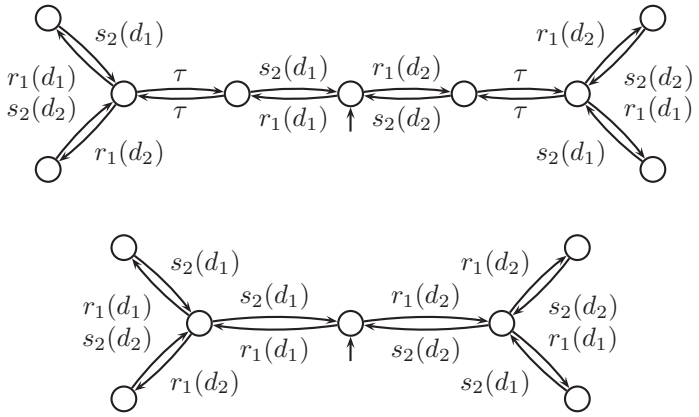


Figure 5.4: The full and reduced LTSs of $\tau_{\{c_3\}}(\nabla_{\{r_1,s_2,c_3\}}(\Gamma_{\{r_3|s_3 \to c_3\}}(X \parallel B)))$

the action c_3. So, we are interested in the behaviour of $\tau_{\{c_3\}}(S)$. In the first labelled transition system in figure 5.4 the hiding operator has been applied on the behaviour as given in figure 5.3. In the second transition system the states connected with τ's have been joined, because they are branching bisimilar.

Exercise 5.5.1. Consider the labelled transition system drawn for the system in exercise 5.3.1 where c_2 is hidden. Draw this transition system modulo branching bisimulation. Would it make sense to reduce this transition system further using weak bisimulation or weak trace equivalence?

5.6 ★Alphabet axioms

The parallel operator and its associated operators such as the hiding and the allow operator have many relations that can fruitfully be exploited. These are for instance useful when performing a parallel expansion. By distributing the communication and

allow operator as far as possible over the parallel operator, the generation of many multi-actions that will be blocked anyhow can be avoided, substantially reducing the size of the calculation.

These relations are characterised by the so called alphabet axioms. The reason is that they are very dependent on the actions labels that occur in a process. The set of action names in a process p is often called its alphabet and denoted by $\alpha(p)$ and is defined as follows on basic processes.

Definition 5.6.1. Let p be a process expression. We define the alphabet of p, notation $\alpha(p)$ inductively by:

- $\alpha(\alpha) = \{\underline{\alpha}\}$ if $\alpha \not\equiv \tau$.
- $\alpha(\tau) = \alpha(\delta) = \emptyset$.
- $\alpha(p + q) = \alpha(c{\rightarrow}p{\diamond}q) = \alpha(p{\cdot}q) = \alpha(p) \cup \alpha(q)$.
- $\alpha(\sum_{d:D} p(d)) = \alpha(p(d))$.
- $\alpha(p \parallel q) = \alpha(p \between q) = \alpha(p) \cup \alpha(q) \cup \{\beta_1|\beta_2 \mid \beta_1{\in}\alpha(p), \beta_2{\in}\alpha(q)\}$.
- $\alpha(p \mid q) = \{\beta_1|\beta_2 \mid \beta_1{\in}\alpha(p), \beta_2{\in}\alpha(q)\}$.
- $\alpha(\Gamma_C(p)) = \alpha(p) \cup \{b|\beta_1 \mid \beta_2{\rightarrow}b{\in}C \text{ and } \beta_1|\beta_2 \in \alpha(p)\}$.
- $\alpha(\nabla_V(p)) = V{\cap}\alpha(p)$.
- $\alpha(\partial_B(p)) = \{\beta \mid \beta{\in}\alpha(p) \text{ and no action in } B \text{ occurs in } \beta\}$.
- $\alpha(\rho_R(p)) = \{\rho_R(\beta) \mid \beta{\in}\alpha(p)\}$.
- $\alpha(\tau_I(p)) = \{\tau_I(\beta) \mid \beta{\in}\alpha(p)\}$.
- $\alpha(\Upsilon_I(p)) = \{\Upsilon_I(\beta) \mid \beta{\in}\alpha(p)\}$.

Here $\rho_R(\beta)$ is defined by $\rho_R(\tau){=}\tau$, $\rho_R(a){=}b$ if $a{\rightarrow}b{\in}R$ for some action name b. Otherwise, $\rho_R(a){=}a$. Furthermore, $\rho_R(\beta_1|\beta_2){=}\rho_R(\beta_1)|\rho_R(\beta_2)$. Similarly, $\tau_I(\beta)$ is defined by $\tau_I(\tau){=}\tau$, $\tau_I(a){=}\tau$ if $a{\in}I$, otherwise, $\tau_I(a){=}a$, and $\tau_I(\beta_1|\beta_2){=}\tau_I(\beta_1)|\tau_I(\beta_2)$. The definition of $\Upsilon_I(\beta)$ is exactly the same as $\tau_I(\beta)$ except that $\Upsilon_I(a){=}int$ if $a{\in}I$.

In table 5.8 the alphabet axioms are given (inspired by [176]). They depend heavily on operations of action labels. To phrase these action label constraints, we require the following notations.

Definition 5.6.2. Let V be a set containing multi-sets of action names. We define the set $\mathcal{N}(V)$ of actions as follows:

$$\mathcal{N}(V) = \{a \mid a \in \alpha{\wedge}\alpha \in V\}.$$

We define the set with multi-sets of action names $\Downarrow(V)$ by

$$\Downarrow(V) = \{\beta \sqsubseteq \alpha \mid \alpha \in V\}.$$

VL1	$\nabla_V(x)=x$	if $\alpha(x)\subseteq V$
VL2	$\nabla_V(x\|y)=\nabla_V(x\|\nabla_{V'}(y))$	if $\Downarrow(V)\subseteq V'$
DL1	$\partial_H(x)=x$	if $H\cap\mathcal{N}(\alpha(x))=\emptyset$
DL2	$\partial_H(x\|y)=\partial_H(x)\|\partial_H(y)$	
TL1	$\tau_I(x)=x$	if $I\cap\mathcal{N}(\alpha(x))=\emptyset$
TL2	$\tau_I(x\|y)=\tau_I(x)\|\tau_I(y)$	
CL1	$\Gamma_C(x)=x$	if $dom(C)\cap\Downarrow(\alpha(x))=\emptyset$
CL2	$\Gamma_C(\Gamma_{C'}(x))=\Gamma_{C\cup C'}(x)$	if $\mathcal{N}(dom(C))\cap\mathcal{N}(dom(C'))=\emptyset\wedge$ $\mathcal{N}(dom(C))\cap rng(C')=\emptyset$
CL3	$\Gamma_C(x\|y)=x\|\Gamma_C(y)$	if $\Downarrow(dom(C))\cap\Downarrow(\alpha(x))=\emptyset$
CL4	$\Gamma_C(x\|y)=\Gamma_C(x\|\Gamma_C(y))$	if $\mathcal{N}(dom(C))\cap rng(C)=\emptyset$
RL1	$\rho_R(x)=x$	if $dom(R)\cap\mathcal{N}(\alpha(x))=\emptyset$
RL2	$\rho_R(\rho_{R'}(x))=\rho_{R\cup R'}(x)$	if $dom(R)\cap dom(R')=\emptyset\wedge$ $dom(R)\cap rng(R')=\emptyset$
RL3	$\rho_R(\rho_{R'}(x))=\rho_{R''}(x)$	if $R''=\{a\rightarrow b \mid (a\rightarrow b\in R\wedge$ $a\notin(dom(R')\cup rng(R')))\vee$ $(c\rightarrow b\in R\wedge a\rightarrow c\in R')\vee$
RL4	$\rho_R(x\|y)=\rho_R(x)\|\rho_R(y)$	$(a\rightarrow b\in R'\wedge b\notin dom(R))\}$
VC1	$\nabla_V(\Gamma_C(x))=\nabla_V(\Gamma_C(\nabla_{V'}(x)))$	if $V'=\{\alpha\|\beta \mid C(\alpha)\|\beta\in V\}$
VC2	$\Gamma_C(\nabla_V(x))=\nabla_V(x)$	if $dom\|(C)\cap\Downarrow(V)=\emptyset$
VD1	$\nabla_V(\partial_H(x))=\partial_H(\nabla_V(x))$	
VD2	$\nabla_V(\partial_H(x))=\nabla_{V'}(x)$	if $V'=\{\alpha \mid \alpha\in V\wedge\mathcal{N}(\{\alpha\})\cap H=\emptyset\}$
VD3	$\partial_H(\nabla_V(x))=\nabla_{V'}(x)$	if $V'=\{\alpha \mid \alpha\in V\wedge\mathcal{N}(\{\alpha\})\cap H=\emptyset\}$
VR	$\nabla_V(\rho_R(x))=\rho_R(\nabla_{V'}(x))$	if $V'=\{\alpha \mid R(\alpha)\in V\}$
CD1	$\partial_H(\Gamma_C(x))=\Gamma_C(\partial_H(x))$	if $(\mathcal{N}(dom(C))\cup rng(C))\cap H=\emptyset$
CD2	$\Gamma_C(\partial_H(x))=\partial_H(x)$	if $\mathcal{N}(dom(C))\subseteq H$
CT1	$\tau_I(\Gamma_C(x))=\Gamma_C(\tau_I(x))$	if $(\mathcal{N}(dom(C))\cup rng(C))\cap I=\emptyset$
CT2	$\Gamma_C(\tau_I(x))=\tau_I(x)$	if $\mathcal{N}(dom(C))\subseteq I$
CR1	$\rho_R(\Gamma_C(x))=\Gamma_C(\rho_R(x))$	if $dom(R)\cap rng(C)=dom(R)\wedge$ $\mathcal{N}(dom(C))=rng(R)\wedge$ $\mathcal{N}(dom(C))=\emptyset$
CR2	$\Gamma_C(\rho_R(x))=\rho_R(x)$	if $\mathcal{N}(dom(C))\subseteq dom(R)\wedge$ $\mathcal{N}(dom(C))\cap rng(R)=\emptyset$
DT	$\partial_H(\tau_I(x))=\tau_I(\partial_H(x))$	if $I\cap H=\emptyset$
DR	$\partial_H(\rho_R(x))=\rho_R(\partial_{H'}(x))$	if $H'=\{\alpha\|R(\alpha)\in H\}$
TR	$\tau_I(\rho_R(x))=\rho_R(\tau_{I'}(x))$	if $I=\{R(a)\|a\in I'\}$

Table 5.8: Alphabet Axioms

Let $C = \{a_1^1|\ldots|a_{m_1}^1 \rightarrow a^1, \ldots, a_1^n|\ldots|a_{m_n}^n \rightarrow a^n\}$ be a set of allowed communications or a set of renamings (in which case only one action occurs at the left-hand side of the arrow). We write $dom(C)$ and $rng(C)$ as follows:

$$\begin{array}{lcl} dom(C) & = & \{a_1^1|\ldots|a_{m_1}^1, \ldots, a_1^n|\ldots|a_{m_n}^n\}, \text{ and} \\ rng(C) & = & \{a^1, \ldots, a^n\}. \end{array}$$

If R is a renaming we write $R(a(d_1, \ldots, d_n)) = b(d_1, \ldots, d_n)$ if $a \rightarrow b \in R$. If α is a multi-action, we apply R to all individual action names. Ie., if α is an action then $R(\alpha)$ is as indicated above. If $\alpha = \alpha_1|\alpha_2$, then $R(\alpha_1|\alpha_2) = R(\alpha_1)|R(\alpha_2)$. For C a set of communications, we write $C(\alpha) = b$ if $\underline{\alpha} \rightarrow b \in C$.

Example 5.6.3. We show how the alphabet axioms can be used to reduce the number of multi-actions when simplifying a process. Consider

$$\nabla_{\{a,d\}}(\Gamma_{\{b|c\rightarrow d\}}(a \parallel b \parallel c)). \tag{5.2}$$

Straightforward expansion of the parallel operators yields the following term:

$$\begin{array}{l} \nabla_{\{a,d\}}(\Gamma_{\{b|c\rightarrow d\}}(a{\cdot}(b{\cdot}c{+}c{\cdot}b{+}b|c){+}b{\cdot}(a{\cdot}c{+} \\ \qquad c{\cdot}a{+}a|c){+}c{\cdot}(a{\cdot}b{+}b{\cdot}a{+}a|b){+}(a|b){\cdot}c{+}(a|c){\cdot}b{+}(b|c){\cdot}a{+}(a|b|c))). \end{array}$$

Via a straightforward but laborious series of applications of axioms this term can be shown to be equal to $a{\cdot}d + d{\cdot}a$. But using the alphabet axioms CL3 and VL2 we can rewrite equation (5.2) to:

$$\nabla_{\{a,d\}}(a \parallel \nabla_{\{a,d\}}(\Gamma_{\{b|c\rightarrow d\}}(b \parallel c))).$$

Expansion of the innermost $b \parallel c$ yields $b{\cdot}c{+}c{\cdot}d{+}b|c$ and application of the communication and allow operator shows that (5.2) is equal to

$$\nabla_{\{a,d\}}(a \parallel d).$$

This is easily shown to be equal to $a{\cdot}d + d{\cdot}a$, too.

Exercise 5.6.4. Simplify $\nabla_{\{e,f\}}(\Gamma_{\{a|b\rightarrow e,c|d\rightarrow f\}}(a \parallel b \parallel c \parallel d))$.

5.7 Historical notes

The history of parallel composition, as we know it in process algebra, goes back to the seminal work of Bekič [24] and Milner [135]. Various process algebras defined different variants of parallel composition, particularly with respect to synchronisation. In CCS [135], there is a hand-shaking (send and single receive) tied with hiding. In CSP [101], multi-party synchronisation is possible, together with an implicit block operator disallowing individual happening of common actions. In ACP [25, 18] (and later in TCP [12]) there is a generic synchronisation scheme which allows, among others, for both hand-shaking and multi-party synchronisation. Moreover, hiding and blocking (also called encapsulation) are separated from parallel composition. The synchronisation scheme introduced in this chapter is an extension of ACP synchronisation scheme, which enforces common data parameters; this is essential for communicating data values among processes.